Systemneutraler automatisierter Abruf von Daten aus dem Liegenschaftskataster

Diplomarbeit im Studiengang Geoinformatik

vorgelegt von
Ludwig Gentz
urn:nbn:de:gbv:519-thesis2008-120-1

Hochschule Neubrandenburg – University of Applied Sciences

Neubrandenburg, den 01.09.2008

Betreuer: Prof. Dr. Ing. Andreas Wehrenpfennig

Dipl. Ing. Christian Fietz

Erklärung

Ich versichere, dass ich meine Diplomarbeit selbständig und ohne fremde Hilfe erfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Ort, Datum Unterschrift

_____ _____

Kurzfassung

Diese Diplomarbeit soll das von der GISAL Anwendergemeinschaft geplante Abrufver fahren und in Diskussion mit dem Zweckverband „Elektronische Verwaltung in M V“ stehende Abrufverfahren nach der Liegenschaftskataster Abrufverordnung (LiKatAVO) M V beschreiben. Dabei werden die Liegenschaftskatasterdaten von einer daten haltenden Stelle der abrufenden Stelle online bereitgestellt. Durch Anfragen dieser können Daten individuell bezogen werden und in bestehende heterogene Systeme im portiert werden. Der Abrufprozess soll anhand eines Prototyps weitestgehend automa tisiert ausgeführt werden.

Abstract

Object of this dissertation is the demand process on real estate cadastre data which is discussed by the GISAL usergroup and the association „Elektronische Verwaltung in M V“ according to the Liegenschaftskataster Abrufverordnung (LiKatAVO) M V. All real estate cadastre data are provided by the responsible organization online. Requests of the data holding organization can be imported by the demanding organization in their existing heterogeneous system. The demand process will be automated as far as poss ible using a prototype.

1 Einleitung

Seit 1993 wird das Liegenschaftskataster in Mecklenburg Vorpommern weitgehend in elektronischer Form geführt. Anwender wie kommunale Verwaltungen und öffentliche Einrichtungen werden durch das Innenministerium autorisiert. Hierzu stellt das zu ständige Kataster und Vermessungsamt die Liegenschaftsdaten noch im WLDG(E) für das Automatisierte Liegenschaftsbuch und im EDBS Format für die Automatisierte Lie genschaftskarte postalisch auf Datenträgern zur Verfügung. Zukünftig werden die Da ten der automatisierten Liegenschaftsverwaltung durch das Amtliche Liegenschaftska taster Informationssystem (ALKIS) im NAS Format, der bundesweit vorgeschriebenen Normbasierten Austausch Schnittstelle, bereitgestellt. Durch die Migration von ALK und ALB ist eine integrierte Datenhaltung in einem Format möglich. Somit bilden Sach daten und Grafik auch physisch eine Einheit. Die Übergabe dieser Daten erfolgt zyk lisch auf Anforderung, in der Regel vierteljährlich.

Auf Grundlage der *„Verordnung über den automatisierten Abruf von Daten aus dem Liegenschaftskataster“* (Liegenschaftskataster Abrufverordnung LiKatAVO M V) vom 18. Juli 2007 und den Anforderungen der öffentlichen Einrichtungen soll dieser Prozess automatisiert werden. Dazu werden die Daten künftig über das Internet von den auto risierten Einrichtungen abgerufen. Die dort verwendete Liegenschaftsverwaltungs software ist jedoch nicht einheitlich, was eine systemneutrale Abrufsoftware erforder lich macht. Zusätzlich werden Schnittstellen benötigt, um die Daten nach dem Abruf weiterverarbeiten zu können und in das bestehende System zu importieren. Dadurch wird ein Einsatz in heterogenen Umgebungen möglich. Hierfür gelten jedoch strenge Anforderungen. So sollte der gesamte Vorgang so weit wie möglich automatisiert wer den. Profitieren werden dabei sowohl die Kataster und Vermessungsämter, als auch die abrufenden Einrichtungen. Wesentliche positive Effekte sind die Senkung der Per sonalkosten, verstärkte Sicherheit und höchstmögliche Aktualität der Daten für den Anwender. Die Automatisierte Liegenschaftskarte (ALK) ist derzeit vollständig bei den zuständigen Kataster und Vermessungsämtern gespeichert. Lediglich die Daten des Automatisierten Liegenschaftsbuchs (ALB) sind zentral beim DVZ (Datenverarbeitungs zentrum) M V Schwerin GmbH gespeichert (ausgenommen: Hansestadt Rostock, Landkreise Bad Doberan, Nordwestmecklenburg, kreisfreie Stadt Wismar) [DVZ M V].

Da das Liegenschaftskataster aus ALB und ALK besteht, beschränkt sich die datenhal tende Stelle unter Umständen nicht nur auf eine einzige Einrichtung. In der Liegen schaftskatasterabrufverordnung ist geregelt, welche Beschränkungen für den automa tischen Abruf gelten. Besonderen Stellenwert haben somit unter anderem die Rege lungen des Landesdatenschutzgesetzes.

Diese Arbeit soll das von der GISAL Anwendergemeinschaft geplante Abrufverfahren und in Diskussion mit dem Zweckverband *„Elektronische Verwaltung in M V“ (siehe* [EgoMV]*)* stehende Abrufverfahren nach der LiKatAVO M V beschreiben. Dazu sollen praktische Lösungsmöglichkeiten für den Einsatz in systemneutraler Systemumgebung aufgezeigt werden. Hierfür ist die Implementierung eines Prototyps notwendig, der durch den Einsatz von geeigneter Technologie Basisfunktionalitäten bereitstellt, wel che den Prozess weitestgehend automatisieren. Anhand dieses Prototyps können po tentielle Anwender ihre Bedürfnisse für die Weiterentwicklung formulieren.

2 Analyse und Anforderungen

Die Spezifikationserstellung erfordert eine sorgfältige Analyse des Abruffahrens. Um ein Grundverständnis zu vermitteln erfolgt eine Erläuterung der bedeutsamsten Be griffe. Dazu gehören *verteilte Systeme* sowie die *Liegenschaftskatasterdaten*. Diese Da ten sind die Objekte, die als Basis des Abrufverfahrens festgelegt sind. Zudem werden mögliche Konfliktszenarien untersucht, welche beim Abruf Verfahren auftreten kön nen. Ein zukunftsorientierter Ansatz bildet dabei die Grundlage für eine spätere, je doch vollständige Umsetzung. In den nachfolgenden Abschnitten wird das momentane Verfahren untersucht und dem geplanten Verfahren gegenübergestellt. Insbesondere die Vorteile des Online Verfahrens werden dabei aufgezeigt. Hierfür ist der Einsatzbe reich des neuen Verfahrens grundlegend zu klären. Da die Liegenschaftskataster Ab rufverordnung die Grundlage für die rechtlichen Rahmenbedingungen bildet, sind die wichtigsten Paragraphen erläutert. Durch die Arbeit mit personenbezogenen Daten (ALB) muss die Anwendung die Sicherheit der Daten gewährleisten. Die systematische Einteilung des Verfahrens unterstützt dabei die weiteren Arbeiten. Für die Auswahl der Komponenten sowie für den Entwurf des Prototyps sind allgemeine Softwarespezifika tionen unerlässlich.

2.1 Verteilte Systeme

Durch die örtliche Trennung von datenhaltender und abrufender Stelle handelt es sich bei dem Abrufverfahren um ein verteiltes System. Die datenhaltende Stelle (Server) hat dabei die Aufgabe die Daten in geeigneter Form der abrufenden Stelle (Client) be reitzustellen. Sowohl Client als auch Server arbeiten zur Erbringung des Abrufdienstes. Dieser Dienst wird auch als verteilte Anwendung bezeichnet. Das System besteht aus unabhängigen Rechnersystemen beliebiger Anzahl. Dabei können mehrere Clients in volviert sein. Der Informationsaustausch wird durch ein Kommunikationsnetz reali siert. Dieses Kommunikationsnetz und die Rechnersysteme bilden zusammen die Hardware des verteilten Systems, während die verteilte Anwendung die Software bil det (vgl. [Heinzl, et al., 2005]). Für den Anwender erscheint das System dennoch kohä rent. Dadurch kann er nicht unterscheiden, ob er gerade mit einer lokalen oder ent fernten Funktionalität arbeitet (vgl. [Tanenbaum, et al., 2008]).

2.2 Liegenschaftskatasterdaten

Die Daten aus dem Liegenschaftskataster sind durch bundesweit einheitliche Liegen schaftsverwaltungsformate realisiert. Derzeit sind dies das Automatisierte Liegen schaftsbuch (ALB) im WLDG(E) Format und die Automatisierte Liegenschaftskarte (ALK) im EDBS Format. Für das geplante Amtliche Liegenschafts Informationssystem (ALKIS) werden sämtliche Daten im NAS Format gehalten. Zusammen bilden sie eine hervor ragende Plattform für die Realisierung des systemneutralen Abrufs der Liegenschafts katasterdaten (vgl. [Fietz, 2008]). Sowohl das EDBS als auch das WLDG(E) Format sind im ASCII Format spezifiziert. Das zukünftige Format NAS liegt in einem leichtgewichti gen XML Format vor. Für den Datenabruf ist es technisch unerheblich wie die Daten intern strukturiert sind. Einzig der konsistente und sichere Transfer der Daten zur ab rufenden Stelle ist erforderlich. Dazu soll die Abruf Software mit Daten beliebiger Grö ße arbeiten können. Die weitere Datenverarbeitung wird vor Ort getätigt. Hierzu liegen die Daten beim Anwender in einer direkt nutzbaren Form vor.

2.3 Derzeitiges Verfahren

Für die sogenannten *„Auswertungen aus dem Automatisierten Liegenschaftsbuch bzw. Liegenschaftskarte“* können schriftlich Anträge bei der zuständigen datenhaltenden Stelle eingereicht werden (siehe Abbildung 2 1). Dabei wird festgelegt, ob nur Ände rungsdaten oder der Gesamtdatenbestand (sogenannte Grundausstattung) übermittelt werden sollen. Weiterhin sind der Beginn und das Intervall der Datenabgabe fest zustellen. Dabei ist eine monatliche, quartalsweise, halbjährige, jährliche oder un bestimmte Datenabgabe möglich. Als Datenträger kommen in der Regel Disketten und CD ROMs zum Einsatz. Allerdings ist der Einsatz von Disketten kritisch zu betrachten, da diese nicht für die sichere (Entmagnetisierung) und dauerhafte Speicherung (Abrieb durch den Schreib /Lesekopf) von Daten ausgelegt sind. Gegenüber Disketten sind CD ROMs auch wegen ihrer deutlich längeren Haltbarkeit zu bevorzugen. Alle Datenträger werden postalisch an den Antragteller zu den Stichtagen übermittelt. Ein erneuter An trag bei zyklischen Auswertungen ist dann nicht mehr erforderlich. Anschließend pflegt eine Fachkraft die erhaltenen Liegenschaftsdaten in die individuelle Software ein. Da tenimportwerkzeuge werden dabei bereitgestellt. Bis zum Einpflegen in das System

der abrufenden Stelle vergehen durch die postalische Übermittlung und die manuelle Importierung mehrere Tage bzw. Wochen. Außerdem ist ein hoher Personalaufwand notwendig, um die postalische Übersendung zu realisieren. Materialkosten und Transportkosten sind dabei sekundär einzustufen.

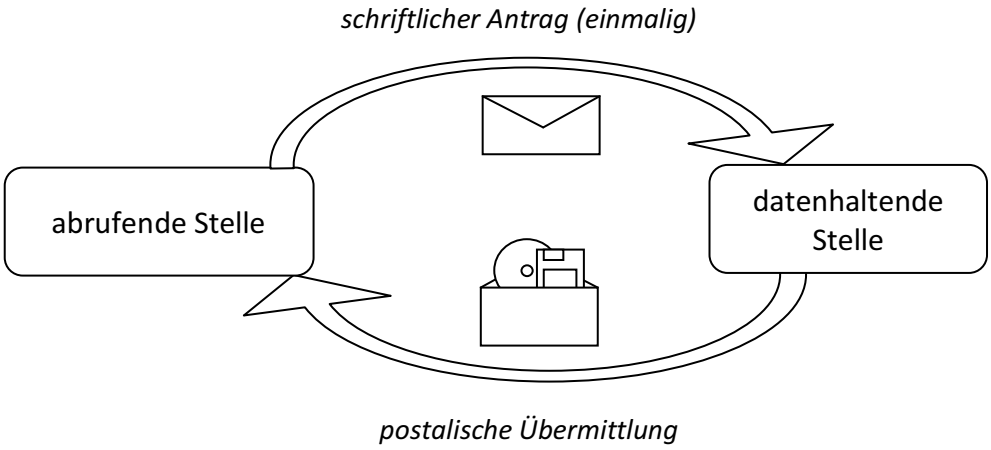


Abbildung 2-1 Derzeitiges Datenübermittlungsverfahren

2.4 Geplantes Verfahren

Zukünftig werden alle Auswertungen elektronisch arrangiert (siehe Abbildung 2 2). Ein schriftlicher Antrag muss dennoch einmalig eingereicht werden. Der permanente Zugriff auf die Liegenschaftsdaten ist nicht notwendig und daher nicht vorgesehen, da die Laufenthaltung auf Seiten der datenhaltenden Stelle ebenfalls nur zyklisch durchgeführt wird. Im Gegensatz zum Abrufverfahren über den Postweg werden die Daten aus dem Liegenschaftskataster direkt über das Internet gesendet. Der Gang zum ansässigen Postamt entfällt somit. Außerdem ist der gesamte Vorgang innerhalb kürzester Zeit abgeschlossen. Durch die höhere Effizienz des Verfahrens, ist eine Verkürzung der Aktualisierungsintervalle sinnvoll. Dies bedeutet, dass die abrufenden Einrichtungen aktuellere Daten zur Verfügung haben werden. Nachhaltig lassen sich Personal und Versandkosten verringern. Natürlich entstehen auf der anderen Seite Wartungs und Anschaffungskosten für entsprechende Hard und Software. Die Softwarepflege erfordert dabei immer Know How bei Mitarbeitern in der jeweiligen Einrichtung.

Das Transfervolumen kann bei einer Übertragung des Gesamtdatenbestandes etwa 100 Megabyte betragen. Für eine ISDN Anbindung ist dieses Transfervolumen noch in

vertretbarer Zeit herunterladbar (t < 4 h), da die Anwendung im Hintergrund des Betriebssystems und daher für den Anwender nicht sichtbar agiert. Der Eingriff durch einen Anwender ist dabei nicht notwendig. Die Größe des Transfervolumens stellt für DSL Anbindungen aufgrund der exzellenten Bandbreite kein Problem dar. Ein Datenabgleich über das Internet ist daher äußerst sinnvoll. Allerdings sollte der Zeitpunkt des Abrufs in der Einrichtung außerhalb der Betriebszeiten liegen (z.B. Sonntag), um die Internetverbindung nicht unnötig zu blockieren. Damit das gesamte Verfahren automatisiert werden kann, sind die heruntergeladenen Daten anschließend in das bestehende System autonom zu importieren. Es werden dazu die Werkzeuge der individuellen Software genutzt. Um die Funktionsfähigkeit der Anwendung zu gewährleisten sind Kontroll- und Wartungsarbeiten durch Fachpersonal notwendig. Alle anderen manuellen Arbeiten entfallen jedoch. Nachdem der gesamte Vorgang abgeschlossen ist, können die Daten von den Endanwendern sofort genutzt werden. Vergleicht man den Personalaufwand des derzeitigen Verfahrens mit dem des geplanten Verfahrens, so wird ein positives Ergebnis erzielt.

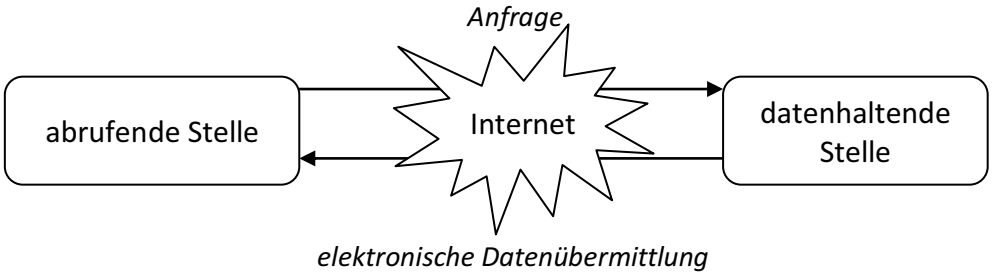


Abbildung 2-2 Geplantes Übermittlungsverfahren

Der Transfer über das Internet erfordert neue Rahmenbedingungen, die rechtlich definiert sind. Auf den nachfolgenden Seiten werden diese analysiert.

2.5 Rechtliche Rahmenbedingungen

Eine Durchführbarkeit des geplanten Abrufverfahrens ist insbesondere an rechtliche Einschränkungen gebunden. Diese Rahmenbedingungen für den automatisierten Abruf sind in der Liegenschaftskataster Abrufverordnung LiKatAVO M V geregelt.

§1 (1)

„Die Übermittlung von Daten des Liegenschaftskatasters im automatisierten Abruf verfahren ist an die in der Anlage zu § 4 genannten abrufberechtigten Stellen für ihren Zuständigkeitsbereich und zur Erfüllung ihrer Aufgaben zulässig. Die abrufberechtigte Stelle kann eine andere Stelle mit der Aufgabenerfüllung beauftragen. Bei der Ver arbeitung von personenbezogenen Daten im Auftrag findet § 4 des Landesdatenschutz gesetzes Anwendung." [LiKatAVO]

Nur abrufberechtigte Stellen sind folglich legitimiert auf die Daten des Liegenschaftska tasters zuzugreifen. Daher muss gewährleistet sein, dass nur diese Stellen Zugang er halten. Weiterhin dürfen Daten nur zur Erfüllung der legitimierten Aufgaben weiterver arbeitet werden. Datenschutzrechtliche Bestimmungen sind unbedingt einzuhalten. Der Datentransfer wird über das Internet realisiert, wo keine physische Trennung der Kommunikationsnetze erfolgt. Deshalb ist der Zugriff durch Unbefugte möglich. Für ei ne Implementierung bedeutet dies, dass alle Komponenten des Systems auf zuverläs sige Sicherheitsstandards beruhen müssen, um vor diesen Zugriffen geschützt zu sein. Ein ungesichertes Abrufverfahren wäre daher keinesfalls mit der LiKatAVO M V kon form.

§1 (2)

„Die Vermessungs und Katasterbehörden halten die in § 3 genannten Daten für den automatisierten Abruf bereit. Sie können sich dazu anderer Stellen bedienen." [LiKatAVO]

Es ist nicht zwingend erforderlich, dass die datenhaltende Stelle auch durch das Katas teramt und Vermessungsamt repräsentiert wird. Zum Beispiel werden derzeitig große Teile der ALB Datenbestände vom Datenverarbeitungszentrum Schwerin (DVZ M V Schwerin GmbH) gehalten. Sämtliche Daten werden dabei noch postalisch übermittelt.

§1 (3)

„Die abgerufenen Daten dürfen von den abrufenden Stellen auf eigenen Datenverarbei tungsanlagen weiterverarbeitet werden." [LiKatAVO]

Paragraph drei regelt die Weiterverarbeitung der Daten in der abrufenden Stelle. Es ist legitim, Daten in das eigene System der Einrichtung zu importieren und anschließend weiterzuverarbeiten (nur zur Erfüllung der Aufgaben). Trotzdem müssen hier ebenfalls Datenschutzrichtlinien eingehalten werden. Die Kontrolle über den rechtmäßigen Umgang mit den lokalen Daten kann nur bedingt technisch realisiert werden. Ein gewissenhafter Umgang mit den Daten ist daher durch die zuständigen Mitarbeiter zu gewährleisten.

§2 (1)

„Die Vermessungs und Katasterbehörden, die von ihnen beauftragten Stellen und die abrufende Stelle haben die nach den §§ 21 und 22 des Landesdatenschutzgesetzes erforderlichen technischen und organisatorischen Maßnahmen, insbesondere zur Zugangssicherung, Abrufberechtigung und Protokollierung, zu treffen." [LiKatAVO]

Die Zugangssicherung verhindert die Nutzung des Abrufverfahrens durch Unbefugte. Melden sich Anwender am System an, wird geprüft, ob eine Abrufberechtigung vorliegt. Dies wird in der Regel durch einen vorherigen schriftlichen Antrag genehmigt. Die Protokollierung bietet weiterhin die Kontrolle gegen Missbrauch. Jeder Abruf wird registriert und für einen Zeitraum von sechs Monaten gespeichert (siehe [LiKatAVO], §2 (2)). In §2 (1) ist außerdem geregelt, welche Daten zur Protokollierung verwendet werden dürfen. Hierzu gehören Benutzerkennung, Datum, Uhrzeit und Ordnungsmerkmale der abgerufenen Datensätze.

§2 (5)

„Es ist sicherzustellen, dass für die abrufende Stelle ein ändernder Zugriff auf Daten des Liegenschaftskatasters ausgeschlossen ist." [LiKatAVO]

Das Verfahren darf demnach keine Mechanismen bereitstellen, die Schreibrechte auf die Daten gewähren.

2.6 Einsatzgebiet der Software

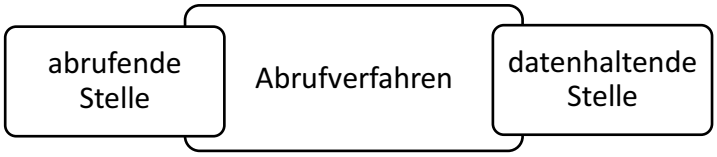


Abbildung 2-3 Schichtmodell des Abrufverfahrens

Der zu entwickelnde Prototyp wird sowohl auf Seiten der abrufenden Stelle als auch auf Seiten der datenhaltenden Stelle eingesetzt (siehe Abschnitt 2.1). Die Softwarekomponente der datenhaltenden Stelle wird dabei als Server bezeichnet, die Softwarekomponente der abrufenden Stelle als Client. Für die Nutzung des Dienstes können mehrere Clients zeitgleich auf den Server zugreifen. Beide Komponenten sind durch eine Middleware verbunden (siehe Abbildung 2 3). Dieses anwendungsneutrale Programm vermittelt zwischen den beiden Stellen und verbirgt die Komplexität und Infrastruktur der zugrunde liegenden Anwendung (vgl. [Abts, 2007]). Dabei wird die gesamte Kommunikation und der Datentransfer zwischen den beiden Stellen übernommen. Darüber hinaus werden externe Werkzeuge bzw. Programme verknüpft, um eine flexible Automatisierung zu erzielen.

2.7 Systemmodellierung

Das System muss aus zwei unabhängigen Modulen bestehen (siehe Abschnitt 0): datenhaltende Stelle und abrufende Stelle bzw. Abruf Client und Transfer Server.

2.7.1 Abrufende Stelle

Die abrufende Stelle ist die Einrichtung, die zyklisch Daten von der datenhaltenden Stelle anfordert (siehe [LiKatAVO], §4). Dies können z.B. kommunale Verwaltungen oder öffentliche Einrichtungen sein. Ferner kommen als potentielle Anwender alle autorisierten Stellen des Innenministeriums (siehe [LiKatAVO], §4) infrage (z.B. Gemeinde- und Amtsverwaltungen). Wurde durch das zuständige Amt eine andere Stelle mit der Aufgabenerfüllung beauftragt, so ist diese Stelle ebenfalls ein denkbarer Anwender. Für das Abrufverfahren wird ein Abruf Client eingesetzt, der die Anforderungen

selbstständig organisiert. Zur Verwaltung der Liegenschaften werden in den Einrich tungen individuelle Softwarelösungen eingesetzt, welche für die eigentliche Verarbei tung der amtlichen Liegenschaftsdaten vorgesehen sind. Die Datenhaltung ist übli cherweise zentral organisiert, um Redundanzen zu vermeiden. Anwender der Liegen schaftsverwaltungssoftware können so gemeinsam auf die Daten zugreifen.

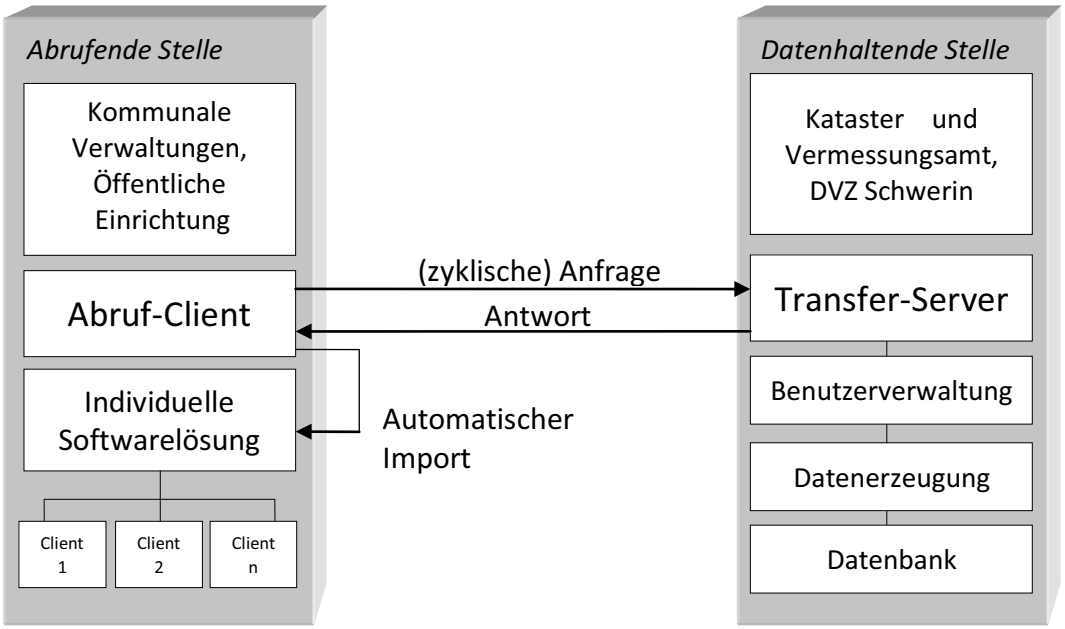


Abbildung 2-4 Systemaufbau mit Komponenten

Objektabfragen

In Gesprächen mit potentiellen Anwendern wurde festgestellt, dass für spezielle An wendungsfälle die zyklische allgemeine Aktualität nicht ausreicht. Daher wird folglich eine Implementierung für das Abfragen von einzelnen Liegenschaftsobjekten in Erwä gung gezogen. Dadurch kann ein einzelnes Objekt (z.B. Flurstück) anhand seines ein deutigen Schlüssels aus dem aktuellen Datenbestand der datenhaltenden Stelle abge fragt werden. Entsprechende Funktionen sind im Entwurf mit einzubeziehen.

Zeitplanung

Sowohl Zeitpunkt als auch Zyklus der Anforderungen werden vom Anwender festge legt. Sinnvolle Konfigurationsmöglichkeiten sind daher erforderlich.

Schnittstelle zur individuellen Software

Abgerufene Daten werden in das bestehende System automatisch importiert und dem Anwender bereitgestellt (siehe Abbildung 2 4, Client 1 n). Dafür werden die abgerufe nen Daten in ein vorher festgelegtes Verzeichnis abgelegt. Der Verweis zu den Dateien wird der individuellen Softwarelösung über eine entsprechende Schnittstelle bereit gestellt. Diese Liegenschaftsverwaltungssoftware hat geeignete Werkzeuge, um die bundesweit einheitlichen Formate in das System zu importieren.

2.7.2 Datenhaltende Stelle

Der Transfer Server hat die Aufgabe, die Daten aus dem Liegenschaftskataster an die abrufende Stelle zu transferieren. Sämtliche Daten werden dazu über das Internet den Anwendern bereitgestellt. Dabei ist eine Kommunikation mit den externen Modulen Benutzerverwaltung und Datenerzeugung notwendig, wobei die Benutzerverwaltung die Abrufberechtigung und den Zugriffsschutz regelt. Die Datenerzeugung generiert aus der Datenbank die Daten aus dem Liegenschaftskataster. Diese werden individuell zusammengestellt (abhängig von Benutzer) und dem Transfer Server für die Übermitt lung übergeben. Dabei kann es sich um eine automatische (autonome) als auch um ei ne manuelle Lösung handeln. Hierzu bearbeitet ein Mitarbeiter der datenhaltenden Stelle die elektronischen Anfragen und erzeugt mit entsprechenden Werkzeugen die Daten, welche abgelegt und der abrufenden Stelle bereitgestellt werden. Die erforder lichen Schnittstellen werden hierfür geschaffen. Auf die Datenerzeugung wird noch ge sondert eingegangen.

2.7.3 Alternativen

E-Mail

Das einfachste Verfahren zur Übermittlung der Daten ist per E Mail. Dazu versendet die Software zyklisch E Mails, die als Anhang die angeforderten Daten enthalten. Die Zieladresse ist dabei die Mailadresse der abrufenden Stelle. Auf der Gegenseite muss das Postfach regelmäßig abgerufen werden. Anschließend sind die Daten ebenfalls über einen automatischen Aufruf mit der individuellen Softwarelösung zu importieren. Der gesamte Prozess muss autonom und vollautomatisch ablaufen, wobei es möglich ist den Datenverkehr zu sichern. Zu beachten ist außerdem die Größe der Nachrichten.

Diese dürfen die zulässige maximale Größe des Posteingangsservers nicht überschrei ten. Im Notfall wäre es aber auch möglich die Nachrichten aufzuteilen. Besonders zu beachten ist in diesem Verfahren auch die Komponente des Mailservers. Rechtliche Rahmenbedingungen müssen dort ebenfalls eingehalten werden. Wird dieser Dienst durch einen privaten Anbieter gestellt, ist kein Einfluss auf die Datensicherheit gege ben. Hier muss sichergestellt sein, dass Sicherheits und Datenbestimmungen der LiKatAVO M V eingehalten werden. Die Daten enthalten personen und eigen tumsbezogene Daten, so dass durch Missbrauch ein erheblicher Schaden entstehen kann. Unter Umständen würde ein Missbrauch nicht einmal bemerkt werden. Um dies zu umgehen ist der Mailserver von der datenhaltenden Stelle selbst bereitzustellen (Hard und Software). Komplexität und Anzahl der Hard und Softwarekomponenten sind dann weitaus höher als bei der verteilten Anwendung. Dies erhöht zudem Hard ware und Wartungskosten, im schlimmsten Fall sogar Personalkosten. Der größte Vor teil des E Mail basierten Dienstes ist die monologe Arbeitsweise (nachrichtenorientier ter Ansatz). Dabei tritt die abrufende Stelle nur passiv in Erscheinung.

VPN

Mit einem *Virtual Private Network* können mehrere lokale Netzwerke über das Inter net miteinander verbunden werden. Sind die Rechner über die VPN Software verbun den, kann ein Rechner den anderen direkt adressieren. Für den Anwender scheint es dann so, als seien die beiden Rechner einem Netzwerk angehörig. Hierzu würde man den Transfer Server der datenhaltenden Stelle mit dem Abruf Client der abrufenden Stelle verbinden. Eine Verbindung ist so auch nur zwischen diesen beiden Rechnern möglich. Sind in der datenhaltenden Stelle Aktualisierungen am Datenbestand erfolgt, so werden diese in ein Eingangsverzeichnis beim Abruf Client abgelegt. Dabei müssen die Daten abhängig vom jeweiligen Benutzer erzeugt werden. Ein Skriptprogramm auf dem Abruf Client würde nun prüfen, ob neue Daten abgelegt wurden und ggf. den Im port initialisieren. Für das VPN existieren genügend Implementierungen, welche si cherheitstechnisch die Anforderungen erfüllen. Es besteht weiterhin die Möglichkeit einer Tunnelung über HTTP, was Firewallkonflikte verringern würde. In welcher Proto kollkombination VPN genutzt wird, ist von der Implementierung abhängig. Nachteilig ist die permanente Verbindung zwischen den Einrichtungen. Befindet sich nur eine ab

rufende Stelle in dem VPN, so ist die Datenlast nicht besonders hoch. Der Sachverhalt ändert sich jedoch, wenn mehrere Teilnehmer dem Netzwerk beitreten. Dabei muss verhindert werden, dass abrufende Stellen untereinander Zugriff besitzen. Entweder wird ein separates virtuelles Netzwerk für jede abrufende Stelle erstellt oder entsprechende Mechanismen verhindern ungewollten Datenverkehr zwischen diesen Stellen. Unnötiger Datenverkehr kann dadurch verhindert werden. Durch die Nutzung des VPNs ist keine komplexere Software mehr notwendig. Beachtet man jedoch, dass eine Protokollierung erfolgen muss, ist dieser Vorteil nicht mehr gegeben. Ohne weitere Software ist daher keine Protokollierung möglich und der Aufwand des VPNs ist höher als bei einer unabhängigen Implementierung.

2.8 Softwareanforderungen

Die Anforderungen beschreiben die Fähigkeiten der Abrufsoftware für einen erfolgreichen Einsatz in der Praxis. Damit werden sowohl funktionale als auch nicht funktionale Anforderungen tangiert.

2.8.1 Zielstellung des Prototyps

Für den Prototyp wird ein allgemeingültiger Ansatz gewählt, damit er in die bestehenden Systemumgebungen homogen eingebunden werden kann. Die Entwicklung als Komplettpaket mit allen Funktionalitäten von Datenerzeugung bis Import ist nicht Aufgabe des Prototyps. Daher werden keine Aufgaben der Benutzerverwaltung übernommen. Es ist davon auszugehen, dass in den datenhaltenden Stellen entsprechende Module existieren, die diese Aufgaben übernehmen können. Abrufsoftware und Benutzerverwaltung müssen jedoch über eine Schnittstelle miteinander kommunizieren. Die abzurufenden Daten müssen der Abrufsoftware durch ein externes Modul zur Verfügung gestellt werden. Die Abrufsoftware selbst erzeugt keine Dateien, daher ist eine Vermittlung unerlässlich. Sind die Dateien abgerufen worden, werden sie in das bestehende System importiert. Da jede Einrichtung ihre individuelle Lösung für den Import einsetzt, ist hierfür ebenfalls keine Implementierung vorgesehen. Nach ausführlichen Konsultationen mit den potentiellen Anwendern, müssen nun diese Betrachtungen konkretisiert werden. Dafür werden grundlegende Funktionalitäten im Prototyp rudimentär implementiert. Dies hat den Vorteil, dass technische Probleme schon im Vor

feld bekannt sind. Zudem hat der Anwender die Möglichkeit Einfluss auf die Entwick lungen des späteren Programms zu nehmen. Das Hauptziel ist demnach der system neutrale und universelle Einsatz.

2.8.2 Systemsicherheit

Das System muss den Sicherheitsanforderungen von §2 (1) LiKatAVO M V entsprechen. Nach heutigen Sicherheitsstandards gehören dazu Authentizität, Vertraulichkeit und Integrität der Daten. Damit ist gesichert, dass die Nachricht vom richtigen Sender aus gegangen ist (Authentizität), die Nachricht vom Zugriff Unbefugter geschützt ist (Ver traulichkeit) und eine Veränderung der Daten ausgeschlossen ist (Integrität). Der di rekte Schreibzugriff auf dem Server wird außerdem ausgeschlossen. Die verwendeten Sicherheitstechnologien sollten eine hohe Marktakzeptanz aufweisen und standardi siert sein. Dadurch wäre gewährleistet, dass die Technologien die Anforderungen nicht nur theoretisch erfüllen, sondern zusätzlich einer kompetenten Kontrolle der Anwen dergemeinschaft unterliegen.

2.8.3 Kosten

Aufgrund der angespannten Haushaltslage in öffentlichen Einrichtungen sind ressour censchonende Technologien zu verwenden. Somit sind Lizenzgebühren für Software zu vermeiden. Dagegen sind Kosten im Wartungsbereich unabwendbar: Aktualisierung, Wartung und Pflege der Software verursachen zumindest Personalkosten. Auf der an deren Seite ist die Anschaffung neue Hardware nicht zwangsläufig notwendig. Eine in dividuelle Sichtung der vorhandenen Hardware in der Einrichtung kann dabei weiteren Aufschluss geben. Die Anforderungen an die Hardware sind gesondert in Abschnitt 2.10 beschrieben.

2.8.4 Systemumgebung

Die Abrufsoftware ist sowohl unter Windows als auch unter Linux ohne große Anpas sungen lauffähig. Dadurch wird die Systemneutralität verstärkt und das Programm kann von einer großen Anwendergemeinschaft eingesetzt werden. Eine Standard schnittstelle zu allen bedeutenden relationalen Datenbanken wird jedoch benötigt. Dadurch ist eine Integration der Protokollierungsdatenhaltung in eine bestehende Da

tenbank gewährleistet. Wartungstechnisch ist damit ein Kompromiss in der Kompo nentenanzahl gewahrt. Ist kein bestehendes Datenbanksystem vorhanden, so ist die ses vom Abrufprogramm mitzuliefern.

2.8.5 Funktionalitäten

Die beschriebenen Funktionen befassen sich mit der internen Funktionalität der An wendung. Insbesondere solche, welche zum erfolgreichen Betrieb in der heterogenen Systemumgebung notwendig sind.

Schnittstellen zu externen Diensten

Um die Zugangssicherung zu gewährleisten, wird eine Schnittstelle zur individuellen Benutzerverwaltung in der Einrichtung bereitgestellt. Eingehende Anfragen werden über die Abrufsoftware an die Benutzerverwaltung weitervermittelt, während Ver zeichnisdienste dabei die Autorisierung des Anwenders übernehmen. Liegt keine Be rechtigung vor, wird der Anwender abgewiesen. Andernfalls wird die Anfrage zugelas sen. Dieses Vorgehen schafft die Grundvoraussetzungen, um das Programm gegen fremden Zugriff zu schützen. Angeforderte Daten werden für jeden Zugriff individuell zusammengestellt. Dadurch bekommt jede Einrichtung genau die Daten, die zur Erfül lung ihrer Aufgaben benötigt werden. Es werden daher nur Daten für den jeweiligen Zuständigkeitsbereich bereitgestellt (in der Regel Gemeinde bzw. Amtsbezogen). Da bei ist eine Schnittstelle erforderlich, die Anfragen an die Datenerzeugung weiterleitet. Über die Anwenderinformationen (z.B. Benutzername) sind dabei die Daten individuell zusammenzustellen. Diese Daten werden der Abrufsoftware in einem Verzeichnis für den Datentransfer bereitgestellt. Der Anwender muss über den Status seiner Anfrage in Kenntnis gesetzt werden, wobei es auch denkbar ist, dass er selbst Informationen über den Status anfordern kann.

Anfragen zu beliebig vielen datenhaltenden Stellen

Aufgrund der dezentralen Datenhaltung der Liegenschaftsdaten ist es notwendig, die Verbindung zu mehreren Servern gleichzeitig konfigurieren zu können. Damit ist es möglich die ALB Daten vom DVZ M V GmbH Schwerin und die ALK Daten vom zustän digen Kataster– oder Vermessungsamt zu beziehen. Die Anwendung muss somit eine entsprechende Verbindungs Organisation besitzen.

Protokollierung von Zugriffen

Werden Daten von der datenhaltenden Stelle bezogen, so werden diese protokolliert. Zu den in Abschnitt 2.5 genannten Merkmalen wird auch die IP Adresse des Nutzers zur Erweiterung der Protokollierung gespeichert, da die Verordnung nur „Mindestan gaben“ (siehe [LiKatAVO], §2 (2)) beschreibt. Dem gegenüber steht einer der wichtigs ten Grundsätze des Bundesdatenschutzgesetzes (BDSG): das „Prinzip der Datenspar samkeit“ ([BDSG], §3a). Hierzu ist zu prüfen, ob die Speicherung der IP Adresse geset zeskonform ist, da es sich um personenbezogene Daten laut BDSG handelt und keine separate Regelung in der LiKatAVO M V existiert. Ein anonymer Zugriff auf die Daten muss jedoch weitgehend ausgeschlossen werden.

Erreichbarkeit des Dienstes

Die datenhaltende Stelle betreibt den Server permanent und sichert die Erreichbarkeit des Moduls über das Internet. Anfragen des Abruf Programms können somit jederzeit entgegengenommen und verarbeitet werden. Außerordentlich wichtig ist dabei die Adressvergabe für den Transfer Server. Diese sollte statisch erfolgen, damit der Server immer dieselbe IP Adresse behält. Der Client hat ansonsten keine Referenz auf die IP Adresse, wie es beispielsweise bei einer Namensauflösung der Fall ist. Die nebenläufi ge Arbeit von mehreren abrufenden Stellen auf dem Server der datenhaltenden Stelle wird durch einen Multithread Server[1] realisiert.

Abrufplanung

Auf Seiten der abrufenden Stelle werden Funktionen bereitgestellt, die es dem An wender erlauben, einen Abruf einmalig oder zyklisch durchzuführen. Die Abrufe sind also durch zeitgesteuerte Ereignisse ausführbar und werden in bestimmten Intervallen wiederholt. Somit obliegt es dem Anwender, die Parameter individuell für jede Verbin dung zu konfigurieren, um den flexiblen Einsatz des Clients zu gewährleisten.

Performance und Optimierung der Datengrößen

Zunächst müssen technologische Voraussetzungen vorhanden sein um große Dateien über das Internet zu übertragen. Die Daten aus dem Liegenschaftskataster liegen im

[1] Der Serverprozess nutzt dabei mehrere Threads, um den Abruf bei konkurrierenden Zugriffen nicht zu blockieren.

ASCII Format vor, wodurch die erzeugten Dateien verhältnismäßig groß sind. Um das Transfervolumen gering zu halten, werden sämtliche Daten zuvor komprimiert. Die Ab rufanwendung muss auf der Gegenseite Funktionen bereitstellen, um die Daten au tomatisch dekomprimieren zu können. Die Dauer des Abrufs kann dadurch minimiert werden.

Graphische Oberfläche und Benutzerfreundlichkeit

Aus Gründen der Benutzerfreundlichkeit und einer bestmöglichen Integration in das Betriebssystem werden alle Interaktionen über eine graphische Oberfläche (engl. *Graphical User Interface – GUI*) realisiert. Der Zugriff mittels GUI ist als separates Mo dul zu implementieren. Angestrebt wird demnach ein modularer Aufbau des Systems. Unterliegende Daten werden dem Anwender in geeigneter Form dargestellt. Die Kos ten können dadurch zumindest im Bereich der Wartung und Pflege der Software opti miert werden.

Die graphische Benutzeroberfläche erschließt sich dem Anwender intuitiv. Anwen dungsmodule können in der Symbolleiste des Betriebssystems eingebunden werden. Sämtliche Interaktionen, dazu gehören Modulstarts, Konfigurationen oder auch Kon trollen des Abrufverlaufs, werden über die GUI realisiert. Eingabefehler durch den An wender werden zur Laufzeit abgefangen. Die Hinweise und Fehlermeldungen werden zusätzlich an die GUI weitergeleitet.

2.9 Anwendungsfälle

Um konkrete Anwendungsfälle festzulegen, wird zunächst die Verfahrensweise bei Anfragen beschrieben. Im zweiten Abschnitt werden lediglich Anwendungsfälle der ab rufenden Stelle untersucht. Auf die datenhaltende Stelle wird im Anschluss eingegan gen.

Datenanfrage und Datenholung

Leider konnte nach bisherigem Stand noch nicht geklärt werden, ob die Datenerzeu gung vollautomatisch durchgeführt werden darf. Rechtliche Bedenken sind dazu noch nicht ausgeräumt. Dieser Sachverhalt wird zukünftig in Konsultationen zwischen dem Innenministerium und den zuständigen Einrichtungen behandelt. Aufgrund dieser Ein

schränkung ist die Datenanfrage derzeit losgelöst von der Datenholung zu implementieren. Für den Abruf sind daher zwei Prozeduren denkbar. Eine Möglichkeit ist der vollautomatische und atomare Abruf. Hierbei wird eine Datenanfrage gestellt und anschließend die Daten erzeugt und heruntergeladen. Die andere Möglichkeit ist gestattet nach der Anfrage einen Mitarbeiter mit dem Zusammenstellen der Dateien zu beauftragen. Zu einem späteren Zeitpunkt muss die abrufende Stelle die Daten herunterladen. Der Anwender kann dabei zu jedem Zeitpunkt den Status der Anfrage einsehen. Dadurch kann die abrufende Stelle prüfen, ob die Anfrage bereits bearbeitet wurde (die Daten sind erzeugt und stehen zum Herunterladen bereit) oder nicht. Wie bei einer Bestellung im World Wide Web muss die Anfrage gespeichert werden und der Kunde in sein Konto einsehen können.

Optimaler ist allerdings eine automatische Datenerzeugung, da der Client seine Daten innerhalb einer Verbindung herunterladen kann. Die Lösung ist sowohl die einfachste als auch die kostengünstigste. In Abbildung 2 5 wird der atomare Prozess schematisch dargestellt.

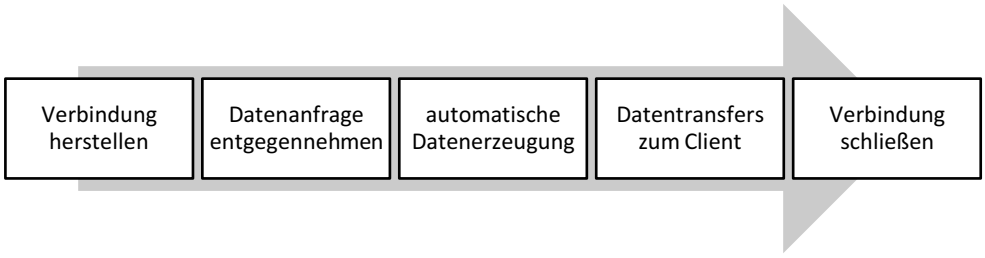
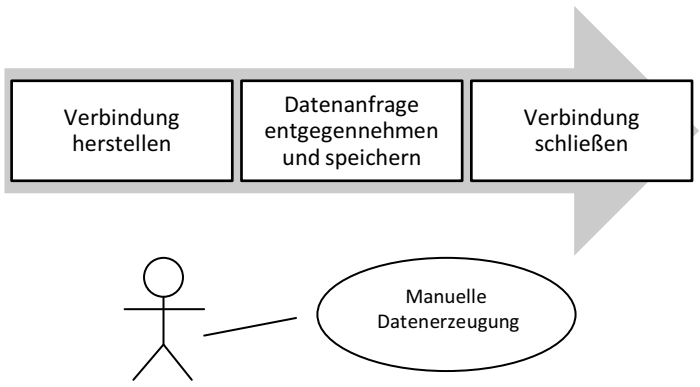


Abbildung 2-5 Vollautomatische Datenerzeugung

Im Gegensatz dazu wird bei einer getrennten Implementierung mindestens zweimal eine Verbindung hergestellt (siehe Abbildung 2 6). Wurde die Anfrage noch nicht bearbeitet, so sind auch noch weitere Anfragen notwendig. Erst nach endgültiger Zusammenstellung der Daten, kann die abrufende Stelle mit dem Herunterladen beginnen. Dies erhöht allerdings die Komplexität und damit das Fehlerrisiko des Prozesses. Zusätzlich verschlechtert sich die Performance des Systems, da deutlich mehr Schritte notwendig sind. Ferner muss das Problem des Zeitmanagements gelöst werden. Wann und wie oft sollen Datenholungsanfragen gestellt werden? Dies kann mit einem Intervall oder einem nachrichtenorientierten Konzept realisiert werden. Bei einem Intervall

muss definiert werden, wie viel Zeit zwischen Datenanfrage und Datenholungsanfrage verstreichen soll. Diese manuellen Prozesse werden durch Mitarbeiter durchgeführt, daher ist eine genaue Vorhersage für die Dauer des Vorgangs nicht gegeben. Eine Kopplung an die Bearbeitungszeiträume für die Datenerzeugung wäre denkbar. Es ist jedoch fragwürdig, ob personalinterne Absprachen in festgelegten Zeitrahmen einge halten werden. Ausnahmen müssen hier ebenfalls behandelt werden. Um dem entge genzuwirken sind Betriebsanweisungen in den Einrichtungen unerlässlich, damit das Intervall eingehalten wird. Wenn Daten niemals erzeugt werden, sendet die Abruf software ständig neue Datenholungsanfragen und blockiert damit zusätzliche Ressour cen. Ein besserer Ansatz wäre ein nachrichtenorientiertes Konzept. Wurde eine Anfra ge bereits bearbeitet und Daten sind erzeugt, so wird eine Nachricht an die abrufende Stelle gesendet. Die abrufende Stelle kann nun eine Datenholungsanfrage stellen und mit dem Herunterladen beginnen. Dies widerspricht jedoch dem Client Server Konzept. Extern scheint es sich um eine Antwort des Servers zu handeln. Dieser tritt jedoch aktiv in Erscheinung, da er nach Beendigung der Verbindung Kontakt mit dem Client aufnimmt. Die technische Umsetzung dieses Problems erhöht die Komplexität des Systems deutlich, da der Client über geeignete Mechanismen (z.B. *DNS*[2] *Domain Name System*) eindeutig ansprechbar sein muss. Außerdem muss die Netzwerkinfra struktur für einen solchen Einsatz konfiguriert sein. Probleme mit Firewalls und Ad ressumsetzungen (*NAT Network Address Translation*) sind ebenfalls zu erwarten.



[2] DNS beschreibt einen Dienst zur Namensauflösung (Ein Hostname besitzt genau eine IP-Adresse).

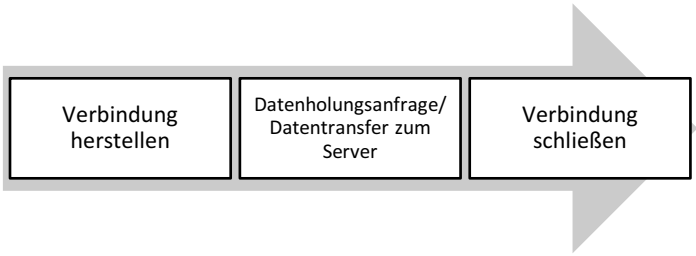


Abbildung 2-6 Manuelle Datenerzeugung

Ein alternatives Verfahren ist die Übermittlung des Bearbeitungszustandes per E Mail (siehe Abschnitt 2.7.3). Der Server sendet nach der Datenerzeugung eine E Mail an die abrufende Stelle. Der Client kann dann in bestimmten Zeiträumen das Postfach nach Antworten des Servers überprüfen und ggf. die Datenholungsanfrage absenden. Dieser Ansatz widerspricht wie die manuelle Datenerzeugung ebenfalls einer tatsächlichen Automatisierung, da der Kernprozess unterbrochen wird.

2.9.1 Abrufende Stelle

Die Verwaltung der Anwendung auf Seiten der abrufenden Stelle wird durch einen Administrator betreut. Konfigurationen werden dabei über die graphische Oberfläche durchgeführt. In bestimmten Zeiträumen wird ebenfalls der korrekte Betrieb der An wendung kontrolliert (siehe Abbildung 2 7).

Konfiguration

Der Anwender kann über die graphische Oberfläche Verbindungen erstellen. Dazu wird jeder Verbindung ein eindeutiger Name zugeordnet (z.B. „Verbindung zum Kataster amt“). Es ist möglich die Verbindung wieder zu löschen oder einfach nur umzubenen nen. Grundlage für eine Verbindung zum Transfer Server ist eine korrekte Konfigura tion der Verbindungseinstellungen. Dazu wird die Internetadresse zum Transferserver konfiguriert. Zur Authentifizierung werden lediglich der Benutzername und das Pass wort eingetragen. Außerdem wird der Ablageort für heruntergeladene Dateien festge legt und der Pfad später an die Importsoftware weitergeleitet. Um den Prozess weit reichend zu automatisieren ist zusätzlich die Erstellung eines Zeitplans erforderlich. Darüber hinaus besitzt jede Verbindung ein Intervall für Datenanfragen. Damit der Anwender den Zeitplan flexibel konfigurieren kann, wird jede Verbindung einzeln kon figuriert, wobei die Zeitplanung im Hintergrund des Betriebssystems agiert. Weiterhin

ist es notwendig einen Dienst zu registrieren, welcher über einen Button gestartet und gestoppt werden kann. Dieser Dienst muss mit den aktuellen Konfigurationsparame tern arbeiten, die bei Änderungen dadurch einen Neustart dieses Dienstes erforderlich machen. Dies geschieht für den Anwender jedoch unsichtbar.

Abbildung 2-7 Anwendungsfalldiagramm (engl. *Use-Case-Diagram*) der abrufenden Stelle

Wartung

Bei der Wartung erfolgt die Kontrolle der Prozesse, die für die Lauffähigkeit des Sys tems nötig sind. Zudem ist die Weiterentwicklung und Verbesserung der Software auch zukünftig erforderlich. Hierdurch soll die Marktakzeptanz der Software erzielen wer den. Zudem sind die Wartungskosten so gering wie möglich zu halten. Am einfachsten ist dies durch eine funktionelle Implementierung zu erreichen, die eine geringe Kom plexität aufzeigt. Alle Komponenten sollen eine hohe Zuverlässigkeit aufweisen, wobei die flexible Integration als systemneutrale Anwendung unerlässlich ist.

Eine hohe Benutzerfreundlichkeit kann Personalkosten einsparen (durch Zeiterspar nis). Da der Abruf zeitgesteuert abläuft, muss der Anwender die Kontrolle über die Programmaktivitäten haben. Dazu gehört ebenfalls die spätere Kontrolle der einge stellten Aufgaben. Über die Konsole kann der Anwender die letzten protokollierten Ak

tionen überprüfen. Anwendungsmeldungen werden zur Laufzeit in eine Log Datei ge schrieben. Dies ist insbesondere für die Kontrolle des Abrufverfahrens unerlässlich.

2.9.2 Datenhaltende Stelle

Auf Seiten der datenhaltenden Stelle erfolgt die Protokollierung. Die Interaktion des Administrators ist primär darauf ausgelegt, die Abrufe der Clients über das Frontend der Protokollierung zu prüfen. Findet eine manuelle Datenerzeugung statt, so wird der Administrator oder ein anderer Mitarbeiter die Daten erzeugen und den Anfragen zu ordnen (siehe Abbildung 2 8).

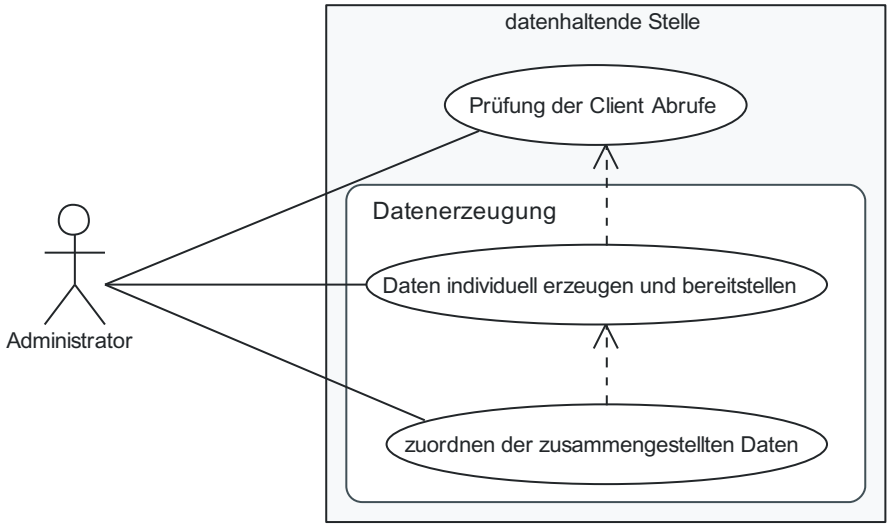


Abbildung 2-8 Use-Case-Diagramm datenhaltende Stelle

2.10 Hardwareanforderungen

Die Hardware sollte sich primär durch Zuverlässigkeit auszeichnen. Sowohl für den Ab ruf Client, als auch für den Transfer Server werden Rechner eingesetzt, die für den per manenten Betrieb ausgelegt sind. Handelt es sich beim Abruf Client auch gleichzeitig um den Netzwerkserver, so sollte der Rechner über genügend Rechenleistung verfü gen. Während des Datentransfers können die Zugriffe der Clients hohe Leistung erfor dern. Oftmals betreibt der Netzwerkserver in kleinen Netzen auch parallel die Daten bank. Dadurch wird zusätzlich Rechenleistung eingenommen. Auf der anderen Seite ist die Anzahl der Anwender des Transfer Servers beim automatischen Abrufverfahren durch die relativ geringe Anzahl der kommunalen Verwaltungen und Einrichtungen (abrufberechtigte) definierbar. Ein zeitgleicher Zugriff von abrufenden Stellen ist daher

sehr unwahrscheinlich. Auch wenn der Abruf der Daten automatisch und ohne Eingriff des Nutzers abläuft, so sollen die Antwortzeiten vertretbar sein. Beim Transfer Server sollte daher mindestens eine Breitband Internetanbindung eingesetzt werden. Speziell auf die Upload Geschwindigkeit der Anbindung ist zu achten. Der Server steht im Netzwerk in Verbindung mit den Modulen „Benutzerverwaltung“ und „Datenerzeu gung“. Eine Firewall sichert den Transfer Server und Abruf Client gegen Angriffe aus dem Internet ab. Allerdings muss hier der Datenverkehr des Servers nach außen frei gegeben sein. Dies betrifft das genutzte Internetprotokoll und die genutzten Ports. Ab bildung 2 9 zeigt ein Beispiel für den Aufbau der Hardwarekomponenten.

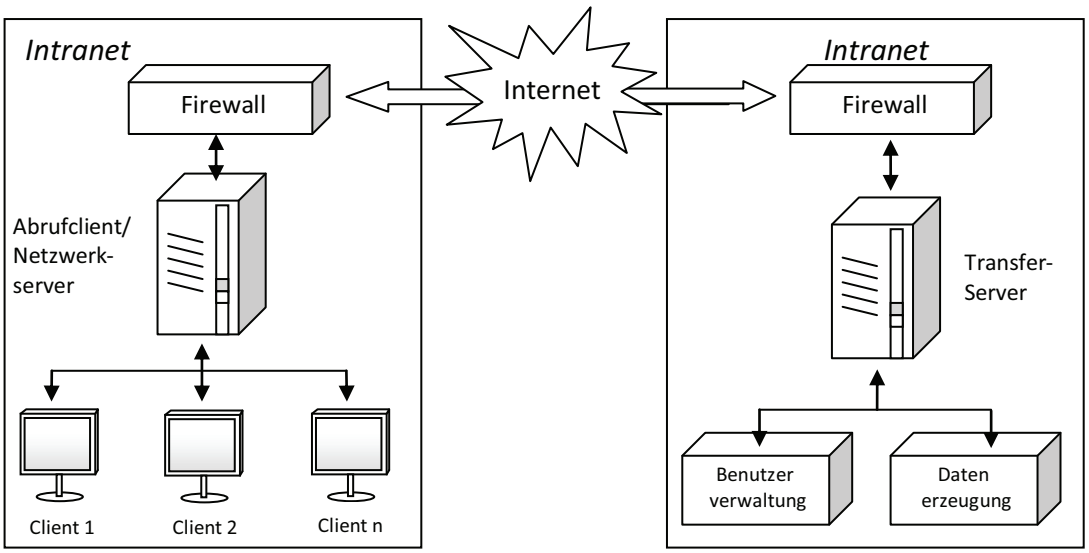


Abbildung 2-9 Hardwaresystemkomponenten

Die Kommunikationsinfrastruktur (z.B. durch ein lokales Netzwerk (LAN)) sollte für den derzeitigen Betrieb in abrufender und datenhaltender Stelle vorliegen, wobei die ver fügbaren Hardwarekomponenten in das System integriert werden können.
Reicht die Rechenleistung für eine Funktionalität nicht mehr aus, so muss eine Lasten verteilung erfolgen *(*engl. *load balancing)*. Die Datenbank kann z.B. für die Erzeugung der Daten auf einem separaten Server betrieben werden. Ein Clustering[3] innerhalb ein zelner Funktionalitäten ist nicht vorgesehen. Diese würden zu Inkonsistenzen und Syn chronisationsproblemen führen, die nur durch komplexe Implementierungen ausge

[3] Clustering bezeichnet die Vernetzung mehrere Computer, um Ausfallsicherheit und Anzahl der Re-chenoperationen zu erhöhen.

schlossen werden können. Mehrere Threads können dennoch parallel ausgeführt wer den (siehe Abschnitt 2.8.5, Funktionalitäten). Dadurch können beliebig viele Clients zeitgleich bedient werden. Der Einsatz von Mehrkernprozessoren kann die Leistung des Systems zusätzlich verbessern. Um den Datendurchsatz auf dem Server signifikant zu erhöhen ist der Einsatz eines RAID Systems empfehlenswert. Durch die Spiegelung der Daten auf mehrere Festplatten kann die Datensicherheit und der Datendurchsatz überdies erhöht werden. Für die ausreichende Datensicherheit der Liegenschaftsbasis daten hat die zuständige Einrichtung der datenabrufenden Stelle zu sorgen. Entspre chende hard und softwaretechnische Maßnahmen sind zur Absicherung zu veranlas sen (z.B. Bandsicherungslaufwerk mit Backupsoftware).

3 Technologieanalyse

In diesem Abschnitt wird analysiert, welche Technologien für die Implementierung ei ner verteilten Anwendung genutzt werden können. Die Vor und Nachteile werden un tersucht und gegenübergestellt. Da es viele Möglichkeiten zur Lösung des Sachverhalts gibt, wurde eine Vorauswahl getroffen. Dabei sind Technologien ausgewählt worden, die sich auf dem Softwaremarkt behauptet haben und weit verbreitet sind. Im Zent rum der Untersuchung steht die Umsetzung der Middleware, da diese die Grundlage für die Realisierung der weiteren Anwendungskomponenten bildet. Die Wahl der Pro grammiersprache sowie die Sicherheitsverfahren sind dabei ebenfalls von Bedeutung. Die Middleware Technologie sollte dabei folgende Eigenschaften aufweisen:

- Geringer Implementierungs und Wartungsaufwand,
- Zuverlässigkeit,
- Unterstützung geeigneter Programmiersprachen in Hinsicht auf weitere Funktionalitäten und Plattformunabhängigkeit,
- Gute Performance für den Interneteinsatz und
- Tunnelung des Protokolls durch Firewalls.

3.1 CORBA

„Die *Common Object Request Broker Architecture* (CORBA) ist ein Standard der Object Management Group (OMG) und definiert eine Kommunikation für verteilte Objekte. Die Struktur von CORBA wird durch das CORBA Referenzmodell verdeutlicht...“ (siehe Abbildung 3 1) [OMG].

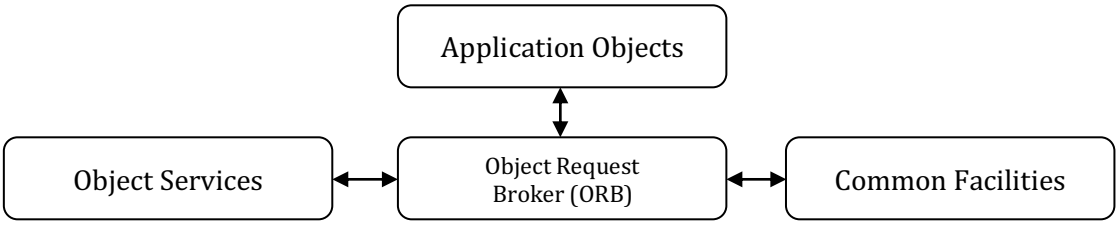


Abbildung 3-1 CORBA Referenz Modell (nach [Heinzl, et al., 2005])

3.1.1 Object Request Broker

In einer Umgebung von verteilten Objekten ist der *Object Request Broker* (ORB) diejenige Komponente, welche die Anfragen entgegennimmt und weiterleitet. Anwendungen von heterogenen und homogenen Umgebungen sind dadurch interoperabel (vgl. [OMG]).

3.1.2 Object Services

Object Services sind domänenunabhängige Basisdienste, die für die Umsetzung von verteilten Anwendungen genutzt werden. Die wichtigsten Dienste sind:

Naming Service

Der *Namensdienst (Naming Service)* wird von einem Client verwendet um Informationen über die Adresse des Zielrechners zu erhalten. Ohne die erforderliche Adresse kann demnach keine Verbindung aufgebaut werden.

Interface Repository

Im *Repository* werden die *Interfaces* der entfernten Objekte abgelegt. Sowohl der Client als auch der Server kennen die Methoden des Interfaces. Auf benötigte Objekte wird dynamisch zugegriffen (vgl. [OMG]).

RootPOA

Der *Program Object Adapter* regelt die Zugriffe auf die Skeletons (siehe Abbildung 3 3). Der Standardadapter wird *RootPOA* genannt. Sämtliche Klassen werden durch den Adapter registriert und definierte Sicherheitsrichtlinien (engl. *security policies*) werden umgesetzt. Dazu erzeugt der Adapter ein Kind Objekt (vgl. [Heinzl, et al.]).

Life Cycle Service

Der *Life Cycle Service* legt Konventionen zur Erzeugung, zum Kopieren, Löschen und Verschieben von Objekten fest. Es wird jedoch nicht festgelegt, wie die Objekte in der Anwendung implementiert sind (vgl. [OMG]).

Concurrency Service

Durch den *Concurrency Service* ist es möglich, konsistent und parallel auf ein Objekt zu zugreifen. Dabei wird keine Spezifikation des Objekts gegeben. Besonders wichtig ist dieser Dienst für den Einsatz von Transaktionen (vgl. [OMGCon]).

3.1.3 Common Facilities

Common Facilities sind Sammlungen von Diensten, die von einer Vielzahl von Anwen dungen verwendet werden, aber einheitlich definiert sind. Im Gegensatz zu den Object Services sind die *Common Facilities* jedoch weniger relevant. Ein Beispiel für eine sol che Spezifikation ist die Facility „Internationalization and Time“. Datum und Uhrzeit, sowie Informationen über kulturelle Gewohnheiten der Anwender werden organisiert (vgl. [OMG]).

3.1.4 Application Objects

Für die *Application Objects* existiert keine Spezifikation. Sie beschreiben die vom An wender implementierten Objekte. Es handelt sich demnach um die eigentlichen Funk tionalitäten der Anwendung. Die Object Mangement Group reiht die *Application Object* als höchste Ebene im CORBA Referenz Model ein (siehe Abbildung 3 1).

Spezifikationen von weiteren Diensten lassen sich auf der Internetpräsenz der Object Management Group finden (siehe [OMG]).

3.1.5 Kommunikation in CORBA

Die Kommunikation in CORBA setzt auf *Ortstransparenz*. Der Client weiß bei einem Aufruf nicht, ob er gerade auf eine lokale oder auf eine entfernte Methode zugreift. Wie in Abbildung 3 2 dargestellt, wendet sich ein Client nicht direkt an die Objekt implementierung, sondern an den Object Request Broker. Die Interaktion zwischen Client und Objekt ist ein objektorientierter Remote Procedure Call (RPC). Der ORB weiß, wo sich das Objekt befindet (siehe 3.1.2, Interface Repository) und sendet die Anfrage an das entsprechende Objekt weiter.

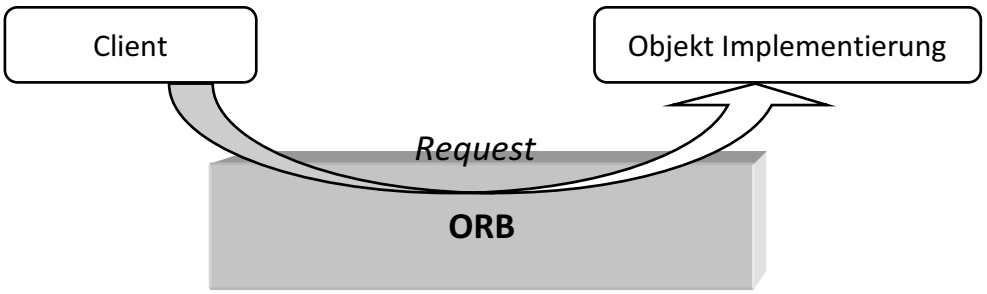


Abbildung 3-2 Object Request Broker (nach [OMG])

CORBA bietet dem Client drei Anfragemöglichkeiten das entfernte Objekt über ein Interface anzusprechen (siehe Abbildung 3 3), wobei das Objekt immer über den im plementierten Stellvertreter angesprochen wird. Damit der Client das Interface des Objekts nicht kennen muss, kann er das Objekt über das Interface Repository anfragen. Dazu sendet er eine Anfrage über das *Dynamic Invocation Interface*, wo die notwendi gen Objekt Informationen übermittelt werden. Eine andere Möglichkeit ist die Nut zung des vom IDL Compiler erzeugten Stubs. Der Stub übernimmt dort ähnliche Funk tionen wie ein Proxy zwischen Client und Server. Dabei hängt die verwendete Pro grammiersprache vom verwendeten IDL Compiler ab. Das ORB kann jedoch auch für einige Funktionen direkt angesprochen werden, wozu das ORB Interface genutzt wird (vgl. [OMG]).

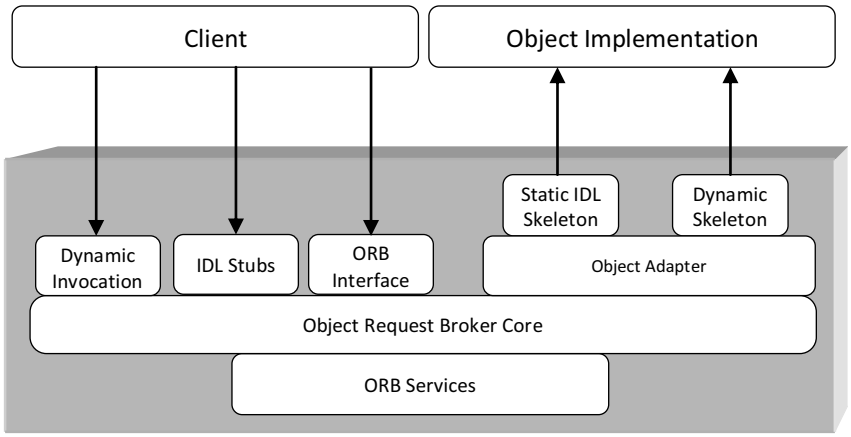


Abbildung 3-3 Die Struktur des Object Request (nach [OMG])

Die Kommunikation innerhalb einer CORBA Implementierung wird über ein hersteller spezifisches Protokoll geregelt. Damit auch unterschiedliche Hersteller untereinander kommunizieren können wurde in der Version 2.0 das General Inter ORB Protocol

(GIOP) festgelegt. Die größte Verbreitung hat der Einsatz des GIOP über TCP/IP (das Internet Inter ORB Protocol (IIOP)). IIOP ist also eine Spezialisierung von GIOP. Für Java gibt es eine native Implementierung. Grundsätzlich ist es für den Client jedoch unerheblich, welche Kommunikationstechnik vom Object Request Broker genutzt wird. Dies wird auch als *Transparenz der Kommunikationstechnik* bezeichnet (vgl. [Hofmann, et al., 2001]).

Bei CORBA werden Verbindungen standardmäßig nicht über HTTP realisiert. Durch die Kombination mit Firewalls können hier nicht zu unterschätzende Probleme auftreten. Im Abschnitt 3.3.2 über RMI wird auf diesen Sachverhalt einmalig eingegangen, da für RMI und CORBA dieselben Schwierigkeiten existieren.

3.1.6 Interface Definition Language

Mit CORBA ist eine Umsetzung in allen gängigen Programmiersprachen wie C, C++, Java, Smalltalk, Lisp, Cobol usw. möglich. Damit CORBA unabhängig von der Programmiersprache ist, wurde eine eigene Schnittstellenbeschreibung geschaffen (*Transparenz der Implementierungssprache*). Die *Interface Definition Language* (IDL) definiert eine Basis für alle Objekte, die in verschiedenen Programmiersprachen implementiert wurden. Wird ein Objekt beispielsweise in Java benötigt, so wird es von der IDL in einen Java Code übersetzt. Auf der anderen Seite werden alle Klassen und Schnittstellen, die für die Kommunikation benötigt werden, in die IDL übersetzt (vgl. [Heinzl, et al., 2005]). Die *Interface Definition Language* (IDL) bildet einen Vertrag zwischen Objekt und der Schnittstelle des Clients. Der Client kann über IDL die entsprechenden Objekte über die Schnittstelle nutzen. Der Vertrag wird in einer Textdatei die meist die Endung „.idl“ trägt gespeichert. Die in IDL definierten Schnittstellen sind den Klassendeklarationen in C++ und den Interfaces in Java sehr ähnlich (vgl. [Hofmann, et al., 2001]).

```
//IDL-Vertrag zwischen Client und Server
module Abruf{
        interface Anfrage{
                String getRequestKey();
        };
};
```

Tabelle 3-1 Einfache Schnittstellenbeschreibung mit IDL

Tabelle 3 1 zeigt ein sehr einfaches Beispiel für eine Schnittstellenbeschreibung in der *Interface Definition Language*. Dabei wird eine Schnittstelle bereitgestellt, die dem Client eine Anfrage ermöglicht. Als Rückgabewert erhält er einen Schlüssel zurück, der die Anfrage eindeutig registriert. Das Element *module* dient zur Schaffung eines Na mensraums und hilft so Namenskonflikte zu vermeiden. Durch die Schnittstellen ist ei ne Mehrfachvererbung möglich (vgl. [Hofmann, et al., 2001]).

3.1.7 Datentypen

In CORBA lassen sich numerische Werte, Boolesche Werte und Buchstaben darstellen. Da es zu Problemen beim Austauschen von Daten zwischen unterschiedlichen Platt formen kommen kann, sind die Wertebereiche der Basisdatentypen genau festgelegt. Für weiterführende Informationen siehe [Hofmann, et al., 2001 S. 251]. Zu finden ist dort eine Tabelle mit den Basisdatentypen und weiteren Erklärungen.

3.2 SOAP

SOAP beschreibt ein „leichtgewichtiges“ Austauschprotokoll für Daten in verteilten Umgebungen sowie für das Ausführen von Remote Procedure Calls. Das Protokoll ba siert auf dem XML Standard und besteht aus drei Teilen. Im Hauptteil befindet sich der *SOAP Envelope*, ein Konstruktor, der den Inhalt einer SOAP Nachricht in einem Fra mework widerspiegelt. Darin enthalten sind Entschlüsselungsregeln für die Teile einer Nachricht, die anwendungsspezifische Datentypen definieren und Konventionen für den Aufruf von entfernten Prozeduren sowie die entsprechende Prozedur selbst bieten (vgl. [SOAPP0]).

Zwar gibt es verschiedene Kombinationen der Transport und Anwendungsschicht, aber die größte Verbreitung hat der Einsatz von SOAP über HTTP und TCP. Zudem wird die Spezifikation von der W3C[4] empfohlen. Der historische Vorgänger ist XML RPC (siehe [Abts, 2007 S. 191]). Derzeit liegt SOAP in der Version 1.2 vor. Sämtliche Beispie le beziehen sich daher ausschließlich auf diese Version.

[4] Das W3C (*World Wide Web Consortium*) ist das Gremium zur Standardisierung des World Wide Webs. Beispiele für solche Standardisierungen sind z.B. HTML, XML, CSS, SVG, RSS und WCAG (vgl. [WikiW3C]).

3.2.1 Aufbau

Dem Envelope muss zuerst ein Namensraum Attribut referenziert werden (http://www.w3.org/2003/05/soap envelope). Als Kind muss der Body definiert sein. Ein Header Element ist (siehe Tabelle 3 2) optional. Er dient dazu Meta Informationen (z.B. Routing, Verschlüsselung, Transaktionsidentifizierungen) zu speichern. Im Body Element sind die eigentlichen Nutzdaten untergebracht. Dazu gehören sowohl einfa che Übertragungsdaten aber auch Informationen für den Prozeduraufruf (vgl. [WikiSOAP]).

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2001/12/soap-envelope">
  <s:Header>
  </s:Header>
  <s:Body>
  </s:Body>
</s:Envelope>
```

Tabelle 3-2 Struktur einer SOAP-Nachricht

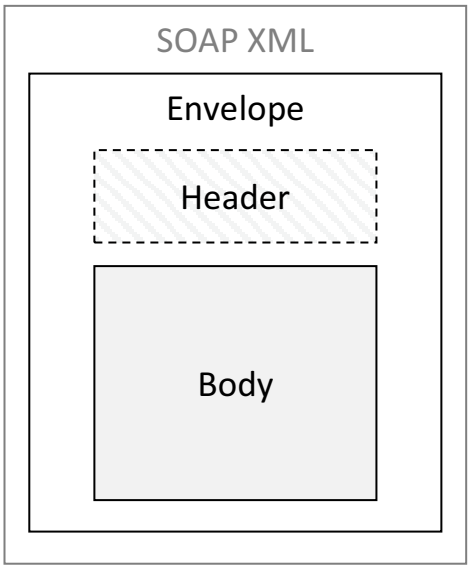


Abbildung 3-4 Bildliche Darstellung der SOAP-Nachricht

Abbildung 3 4 zeigt die bildliche Darstellung einer SOAP Nachricht. Dabei wird deut lich, dass der Envelope die Basis für die Metainformationen als auch für die eigentli chen Daten bildet.

3.2.2 Kommunikation

Die SOAP Anwendung spricht eine Referenz des entfernten Objektes an. Die XML Nachricht wird dazu über die POST Methode des asynchronen HTTP Protokolls über tragen. SOAP selbst hat keine Spezifikation für die Semantik zwischen den Anwendun gen. Eigentlich sind Nachrichten in SOAP zustandslos. Trotzdem stellt SOAP ein Framework zur Verfügung, um beliebige applikationsspezifische Informationen zu übertragen. Durch sogenanntes „Session Management“ wird für jeden Client ein ein deutiger Schlüssel generiert. Anhand dieses Schlüssels ist der Client bei dem nächsten zustandslosen Aufruf wieder identifizierbar. Dadurch können Sicherheitslücken (engl. *session stealing*) entstehen, die aber durch einen guten Schlüsselerzeugungsalgorith mus hinreichend gesichert werden können. Eine intelligente Interaktion zwischen den Applikationen erlaubt es, eine komplexe Kommunikation zu realisieren. Durch Ausnut zung der darunterliegenden Protokolle können neben „one way“ Anfragen auch Multi castanfragen[5] gestellt werden. Festgelegt ist allerdings, wie ein SOAP Knoten agiert, wenn eine Nachricht eintrifft. Dabei wird einen sogenannter "Message Path" verfolgt. Jede Nachricht kann so an Knotenpunkten verarbeitet werden, bevor sie beim eigentli chen Ziel ankommt (vgl. [SOAPP0]).

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2001/12/soap-envelope">
<s:Header>
        <user:login>
                <user:benutzername>meinNutzername<user:/benutzername>
                <user:passwort>meinPasswort<user:/passwort>
        <user:/login>
<s:/Header>
<s:Body>
        <m:GetKey  xmlns:m="http://www.abruf-von-liegenschaftsdaten.de/soap">
        </m:GetKey>
   </s:Body>
</s:Envelope>
```

Tabelle 3-3 SOAP-Anfrage

Die SOAP Anfrage in Tabelle 3 3 hat zwei Kind Elemente von Envelope: den Header und den Body. Der Header ist zwar optional, jedoch hilft er bei der Übertragung von

[5] Bei Multicastanfragen wird eine einzelne Nachricht an mehrere Empfänger gesendet. Diese wird bei den Empfängern automatisch weitergeleitet (vgl. [Tanenbaum, et al., 2008]).

Metainformationen. Diese enthalten Zusatzinformationen über die eigentlichen Daten der Nachricht. Als Zieladresse wird der Namensraum (engl. *„namespace“*) adressiert (*xmlns*). Die Anfrage soll von einem Server einen Schlüssel anfordern (z.B. Daten anfrage für den Abruf der Liegenschaftsdaten). Im Header werden gleichzeitig die Au torisierungsinformationen eingebettet. Tabelle 3 4 zeigt beispielhaft eine Antwort des Servers.

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2001/12/soap-envelope">
<s:Body>
        <m:AbrufResponse xmlns:m=
          http://www.abruf-von-liegenschaftsdaten.de/soap"
        <m:Key  value="2ed2fa61-8213-413c-9cf2-47fdc3f7970d"></m: Key>
        </m: AbrufResponse>
</s:Body>
</s:Envelope>
```

Tabelle 3-4 SOAP-Antwort

Der zurückgegebene Schlüssel befindet sich im Tag “AbrufResponse”, welcher wiede rum ein Kind Element vom Body ist. Der Key Tag hat als Attribut mit dem Namen „Value“ den Inhalt des angeforderten Schlüssels.

role-Attribut

Durch das Prinzip des „Message Paths“ ist es möglich, dass eine SOAP Nachricht even tuell auf dem Weg vom Sender zum endgültigen Empfänger noch mehrere Zwischen stationen passieren muss. Jede Zwischenanwendung ist dabei eine SOAP Applikation, die Nachrichten empfangen und verarbeiten kann. Durch Setzen des role Attributs kann der Pfad allerdings konfiguriert werden. Tabelle 3 5 zeigt die möglichen Attribute, die gesetzt werden können. Ist das optionale role Attribut nicht gesetzt, wird die Nach richt automatisch erst beim endgültigen Empfänger verarbeitet (siehe Tabelle 3 6). Folglich ist es möglich eine Nachricht auch einfach nur weiterzuleiten. Ist es jedoch er wünscht, dass auch die Zwischenstationen die Nachricht verarbeiten, so muss das role Attribut global festgelegt sein. Damit jede Station eindeutig identifizierbar ist, werden URIs[6] gesetzt (siehe Tabelle 3 5).

[6] Ein URI (engl. *Uniform Resource Identifier*) dient der Identifizierung einer virtuellen oder physischen Ressource.

"http://www.w3.org/2003/05/soap-envelope/role/none"

"http://www.w3.org/2003/05/soap-envelope/role/next”

"http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver"

Tabelle 3-5 Role-Attribute (nach [SOAPP0])

Tabelle 3 6 zeigt die Abhängigkeit der Verarbeitung vom gesetzten role Attribut. „Ja“ bedeutet, die Nachricht wird von der entsprechenden SOAP Applikation verarbeitet. „Nein“ bedeutet, die Nachricht wird nur weitergeleitet.

Knoten	/	none	next	Ultimate Reveiver
Zwischenstation	Nein	Nein	Ja	Nein
Endgültiger Empfänger	Ja	Nein	Ja	Ja

Tabelle 3-6 Abhängigkeit des „Message Paths“ von role-Attributen (nach [SOAPP0])

Für Nachrichtenorientierte Systeme kann dieses Verfahren enorme Vorteile in der Ausnutzung der Bandbreite erzielen. Da es sich bei abrufender Stelle und datenhalten der Stelle nur um zwei Stationen handelt, wird die Thematik nicht weiter vertieft. SOAP würde daher als Punkt zu Punkt Verbindung eingesetzt werden. Die offizielle Dokumentation von SOAP kann jedoch noch weiterführende Informationen bieten (siehe [SOAPP1]).

3.2.3 Ausnahmebehandlung

Die Ausnahmebehandlung beschreibt, wie fehlerhafte Programmzustände während der Laufzeit des Programms an andere Komponenten weitergeleitet werden. Eine gute Ausnahmebehandlung kann dem Anwender oder Entwickler Aufschluss über die Ursachen des Problems geben. Bei verteilten Anwendungen existiert ein weiteres Problem – die Ortstransparenz. Daher muss der Ort des fehlerhaften Zustandes ebenfalls bekannt sein.

mustUnderstand-Attribut

Wird eine Nachricht vom entsprechenden Empfänger verarbeitet, so ist es möglich, dass nicht alle Ausdrücke der SOAP Nachricht richtig interpretiert werden. Zu diesem Zweck existiert das mustUnderstand Attribut. Es legt fest, ob eine korrekte Verarbei

tung optional oder obligatorisch ist. Mögliche Werte für das Attribut sind „false“ und „true“. Ist kein Wert gesetzt, so wird standardmäßig „false“ angenommen. Kann ein Ausdruck nicht verarbeitet werden, wird dementsprechend ein Fehler zurückgegeben. Zusätzlich wird die Nachricht nicht mehr weitergeleitet.

Fault-Element

Entsprechende Fehler werden über ein *Fault Element* zurückgegeben. Das Element muss im Body stehen und darf nur einmal in der Nachricht vorkommen. Tabelle 3 7 zeigt mögliche Fault Codes. Der Entwickler kann so die Fehlerquelle kategorisch oder örtlich klassifizieren.

Name	Bedeutung
VersionMismatch	Beim entsprechenden Knoten konnte ein Namensraum nicht verarbeitet werden.
MustUnderstand	Ein Header-Element konnte nicht verarbeitet werden, obwohl es als „mustUnderstand“ deklariert wurde.
DataEncodingUnknown	Die Datentypen können nicht in eine SOAP-Nachricht übersetzt werden.
Sender	Die Nachricht wurde nicht richtig gebildet oder enthielt nicht die erforderlichen Informationen.
Receiver	Der Inhalt der Nachricht kann zwar verarbeitet werden, allerdings gibt es einen Fehler beim weiteren Prozess.

Tabelle 3-7 Fault-Codes (vgl. [SOAPP1])

Die Fehleranalyse bei verteilten Anwendungen gestaltet sich besonders schwierig, da eine Netzwerkverbindung die Anzahl der Fehlerquellen (Rechnerabstürze, Netzausfälle bzw. Überlastungen, Versionskonflikte) erhöht. Der Einfluss auf die entfernte An wendung ist meist nicht gegeben.

3.2.4 Datentypen

SOAP bietet über 40 Datentypen an. Neben Basisdatentypen wie Byte, Short, Integer, Long, Double, Float und String, werden Datumstypen und Spezialisierungen der primi tiven Datentypen angeboten. Zusätzlich existieren zusammengesetzte Datentypen.

Dabei wird zwischen Structs (siehe Tabelle 3 8) und Arrays (siehe Tabelle 3 9) unter schieden.

Structs werden einzig durch ihren Namen eindeutig identifiziert, Elementnamen dürfen daher nur einmalig existieren. Ein Struct ist aus mehreren Unterelementen zusammen gesetzt, welche wiederum aus Structs bestehen können. Die Bestandteile sind lediglich lokal sichtbar (vgl. [SOAPData]).

```
<e:Login>
        <username>mustername</username>
        <password>musterpasswort</password>
</e:Login>
```

Tabelle 3-8 Struct Beispiel

Ein Array kann jegliche Datentypen beinhalten, wobei Mehrdimensionale Arrays eben falls möglich sind. Dabei spielt der Name dieser Elemente keine Rolle. Nur die Position der Elemente dient dabei als Identikator.

```
<keys SOAP-ENC:arrayType="xsd:string[3]">
        <key>d6e2e8e0-3b8f-11dd-ae16-0800200c9a66 </key>
        <key>e0d2c89a-1468-41db-ab9d-071e2047230b</key>
        <key>fb09acf9-ee5f-4ba0-a7dd-3df6330b6d45</key>
</keys>
```

Tabelle 3-9 Array in SOAP

Für eine Gesamtübersicht der verfügbaren Datentypen siehe [SOAPData]. Falls es er forderlich ist, können eigene Datentypen definiert werden.

3.2.5 Performance

Die Struktur des XML Dokuments kann sich nachteilig auf die Performance auswirken, da die Daten nicht binär gespeichert sind. Zudem ist der Einsatz eines Parsers notwen dig. In [Dostal, et al., 2005] wurde die Performance von SOAP ausführlich untersucht

und mit CORBA und RMI verglichen. Insbesondere der Einsatz von SOAP über HTTP mit großen Nachrichten erzeugte dabei schlechte Resultate.

3.2.6 Sicherheit

Die Spezifikation bietet aus Gründen der Einfachheit keine Lösungen an, um die An wendung zu sichern. Die Daten können im Klartext mitgelesen werden, da sie im XML Format übertragen werden. Falls es erforderlich ist, können jedoch die Pakete über HTTPS (*Hypertext Transfer Protocol Secure*) übertragen werden. Dies dient zur Ver schlüsselung und Authentifizierung der Kommunikation (über SSL/TLS). Sinnvoll kann SSL mit SOAP jedoch nur bei Punkt zu Punkt Verbindungen eingesetzt werden. Ist le diglich die Verschlüsselung der Datei gewünscht, kann dies auch mit anderen Frame works erzielt werden. *XMLEncrypt* (siehe [XmlEnc]) ermöglicht die Verschlüsselung von XML Bäumen und Dateien. Die Realisierung von Richtlinien kann durch das *WS Policy* Framework erfolgen (siehe [WSPOL]). Weitere Frameworks sind *WS Trust*, *WS Secure Conversation*, *WS Privacy*, *WS Federation* oder *WS Authorization* (vgl. [Dostal, et al., 2005]*)*. Diese Spezifikationen sind zwar vorhanden, bilden aber nur einen Teil der ver fügbaren Sicherheitsrichtlinien. Die individuelle Einbindung in die Anwendung ist je doch erforderlich. Auch eine standardisierte Bibliothek ist derzeit nicht verfügbar. Da durch erhöht sich der Implementierungsaufwand.

3.3 RMI

Ein alternativer Ansatz zur Entwicklung von verteilten Anwendungen ist der Aufruf ent fernter Methoden (engl. *Remote Method Invocation*). RMI ist eng an die Program miersprache Java geknüpft und erlaubt eine Verteilung von Anwendungen auf ver schiedene virtuelle Maschinen (siehe Abschnitt 3.5.1). In der Standardedition von Java (*Java SE*) ist das RMI Framework bereits enthalten. Grundlegend ist es nur mit Java möglich RMI zu nutzen. In der Praxis existieren jedoch Schnittstellen zur Nutzung von RMI über andere Spezifikationen bzw. Architekturen (z.B. über CORBA). Dem Entwick ler werden sämtliche Kommunikationsdetails verdeckt, weshalb der Fokus bei der ei gentlichen Umsetzung der Anwendung liegt, somit ist RMI eine sehr effektive Lösung der verteilten Anwendungsprogrammierung. Äquivalent zu CORBA setzt RMI auf *Ob*

jekttransparenz. Der Client kann nicht zwischen einer lokalen oder entfernten Metho de unterscheiden (vgl. [Heinzl, et al., 2005]).

Eine lokale Methode kann sämtliche serialisierbare Objekte untereinander austau schen. Durch den objektorientierten Ansatz ist es möglich, eigene Klassen zu definie ren und zwischen den verteilten Anwendungen auszutauschen. RMI nutzt die in Java vorhandenen Sicherheitsmechanismen. Dazu gehört der *Security Manager (siehe Ab schnitt 3.5.3)* und das „Sandkasten" Prinzip (siehe Abbildung 3.5.2). Die Entwicklungs zeiten von RMI Anwendungen sind relativ gering, da sich der Entwickler nur um die ei gentliche Anwendung beschäftigen muss. Außerdem ist es möglich Altsysteme (engl. *legacy systems*) zu nutzen. Java bietet dazu das Java Native Interface (*JNI*). Ein vorhan denes Programm, das z.B. in C geschrieben wurde, kann dadurch genutzt und Kunden wünsche flexibel realisiert werden. Dies ist insbesondere notwendig, wenn Altsysteme (engl. *legacy systems*) bestehen bleiben müssen. Das Konzept der virtuellen Maschine ermöglicht die Portierung sämtlicher RMI Anwendungen. Es sind keine großen Ände rungen nötig, um eine Portierung von Windows auf Linux umzusetzen. (vgl. [SunRMIb])

3.3.1 Aufbau und Kommunikation

RMI besteht aus den folgenden Komponenten:

- RMI Server und entfernte Objekte,
- RMI Client,
- RMI Registry,
- Stub(Objekte) und
- Skeleton(Objekte).

RMI-Server

Der Server implementiert die entfernten Methoden und stellt diese dem RMI Client bereit. Dazu werden sowohl ein Remote Objekt bei der Registry als auch ein eindeu tiger Name festgelegt. Beispiel: *rmi://192.168.0.1/meineRmiAnwendung.*

RMI-Client

In der verteilten Anwendung können beliebig viele Clients auf den Server zugreifen.

Die Kommunikation wird über die entsprechenden Stellvertreter realisiert. Dazu muss sich der RMI Client lediglich eine Referenz des entfernten Objekts über die RMI Registry holen.

RMI-Registry

Die Registry stellt einen Namensdienst bereit, von welchem der Client erst nach Anfra ge die entfernte Methode aufrufen kann (vgl. [Heinzl, et al., 2005]).

Skeleton

Der Skeleton ist der serverseitige Stellvertreter.

Stub

Der clientseitige Stellvertreter des entfernten Objektes beim RMI Server wird Stub Objekt genannt. Der Client ruft das Stub Objekt auf, worauf hin die Anfrage mit den entsprechenden Parametern an die entfernte Methode weitergeleitet wird (siehe Ab bildung 3 5). Ab Java SE 5.0 werden Stubs zur Laufzeit dynamisch generiert. In früheren Versionen musste der Stub mit dem Tool *rmic* manuell erstellt werden.

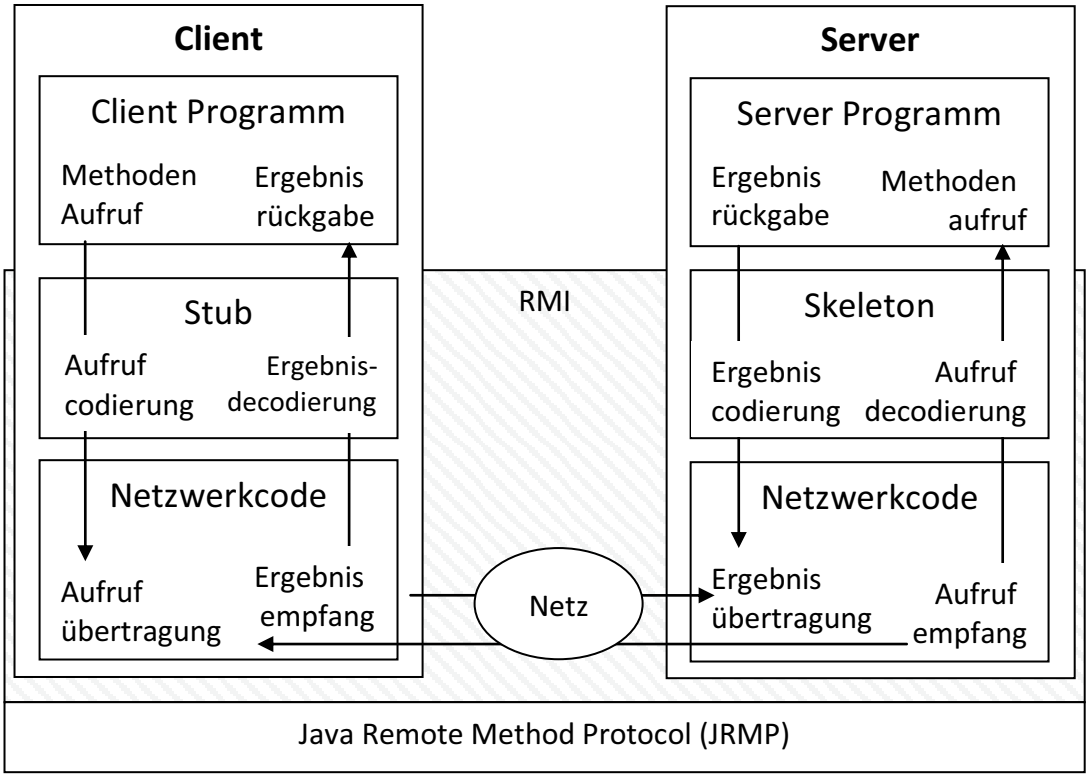


Abbildung 3-5 Aufruf einer entfernten Methode (nach [Abts, 2007])

Zuvor müssen die Daten für die Serialisierung in eine übertragbare Form gebracht werden (marshalling). Serialisierung bedeutet, dass der Status des Objekts in ein Byte Array zerlegt wird, damit eine Verarbeitung als Stream möglich ist. Wo die Daten wie der deserialisiert (unmarshalling) und weiterverarbeitet werden (siehe Abbildung 3 5). Rückgabewerte werden wieder zurückgewandelt. Ein Objekt ist allerdings nur seriali sierbar, wenn die Klasse das Interface *java.io.Serializable* implementiert (siehe [Heinzl, et al., 2005 S. 112]). Sämtliche Verbindungen werden über TCP/IP verbindungsorien tiert realisiert, wozu das Java Remote Method Protocol (JRMP) genutzt wird.

Remote Interface

Das Remote Interface ermöglicht dem Client eine Sicht auf die Methoden des Servers. Als Namenskonvention hat sich hier die Erweiterung *Impl* durchgesetzt. Dieses Inter face muss vom Interface *Remote* abgeleitet sein, wodurch die Methoden der einzelnen Interfaces als entfernte Aufrufe gekennzeichnet sind. Tabelle 3 10 zeigt eine Metho dendefinition mit dem Namen *requestKeys*. Bei Kommunikationsproblemen durch die Netzwerkverbindungen werden die Ausnahmen von *RemoteException* behandelt.

```
public class AbrufImpl extends Remote implements Abruf {
        public String[] requestKeys() throws RemoteException;
}
```

Tabelle 3-10 Remote Interface in RMI

3.3.2 Einsatz hinter Firewalls

Im Gegensatz zu SOAP nutzen RMI Anwendungen Socketverbindungen, über welche das Protokoll übertragen wird. Mit diesem Konzept lassen sich Anwendungen realisie ren, die hohe Performance erfordern. Allerdings besteht die Gefahr, dass Firewalls die Verbindung blockieren. Im ungünstigsten Fall sind sowohl der Client als auch der Ser ver durch die Firewall geschützt. Die Aufgabe der Firewall ist Anwendungen daran zu hindern über bestimmte Ports miteinander zu kommunizieren. Die Sicherheit kann noch erhöht werden, indem zusätzlich nur noch bestimmte Protokolle die Firewall durchlaufen. Da es jedoch keinen Sinn macht alle Verbindungen zu blockieren, muss zumindest das HTTP Protokoll zugelassen werden, damit Anwender das World Wide Web nutzen können. Die Entwickler von Java haben RMI so konzipiert, dass auch ein

Einsatz hinter Firewalls möglich ist. Eine Verbindung wird über die folgenden Schritte aufgebaut.

Verbindungsverfahren von RMI

1. Es wird versucht eine *direkte Verbindung* über die reservierten Ports von RMI herzustellen (Port 1099 Namensdienst. Port 1098 für den Activator).
2. Aufbau einer *direkten Verbindung*. Diesmal wird jedoch die Anfrage über das *HTTP Protokoll* gestellt. Dies funktioniert allerdings nur, wenn die Firewall di rekte Verbindungen zu geschützten Sockets zulässt und nur HTTP Anfragen ak zeptiert.
3. Schlägt auch der zweite Verbindungsaufbau fehl, so versucht RMI nun über den *Proxy Server* eine Verbindung herzustellen. Der Proxy soll die Anfrage an den RMI Server weiterleiten. Voraussetzung ist, dass die Firewall den HTTP Transfer über jeden beliebigen Port zulässt.
4. Da zumindest *Port 80* geöffnet sein muss, versucht RMI über diesen wieder *di rekt* zu verbinden. Ein Proxy Server wird in die Anfrage nicht miteinbezogen. Al lerdings ist jetzt ein Webserver notwendig, der die gleiche Aufgabe hat. Der RMI Server nimmt nun keine Verbindungen mehr an, sondern der Webserver lädt den Stub herunter. Damit er die Anfragen weiterleitet, muss ein CGI Programm oder ein Servlet auf dem Webserver aktiv sein. Sowohl das CGI Programm als auch das Servlet sind in Java SE enthalten.
5. Die letzte Möglichkeit ist eine Kombination von Punkt drei und vier (siehe Ab bildung 3 6). Die HTTP Anfrage wird jetzt über den Proxy Server gestellt wo raufhin ein Webserver die Anfrage entgegennimmt. Dieser leitet alle Anfragen an den RMI Server weiter (vgl. [Neal Harman]).

RMI versucht immer den einfachsten Weg der Verbindung herzustellen, erst dann wird eine komplexere Verbindung aufgebaut.

Mit dem gezeigten Verfahren sind zwar RMI Anwendungen in fast jeder Umgebung einsetzbar, nicht aber ohne erhebliche Aufwände. Das Öffnen der betroffenen Ports

durch den Administrator ist zweifellos die einfachste Lösung aus sich der RMI Anwendung. Allerdings werden so jegliche Sicherheitskonzepte der Firewalls vernach lässigt. Die primäre Aufgabe der Firewall ist es, das Intranet vor dem Internet zu schüt zen. Dagegen soll der jedoch Webverkehr zugelassen werden. Weiterhin wird die RMI Anwendung nur von einer definierten Anzahl von Nutzern verwendet. Es ist nicht sinn voll, die Sicherheit der anderen Anwender durch das Öffnen der Ports zu gefährden.

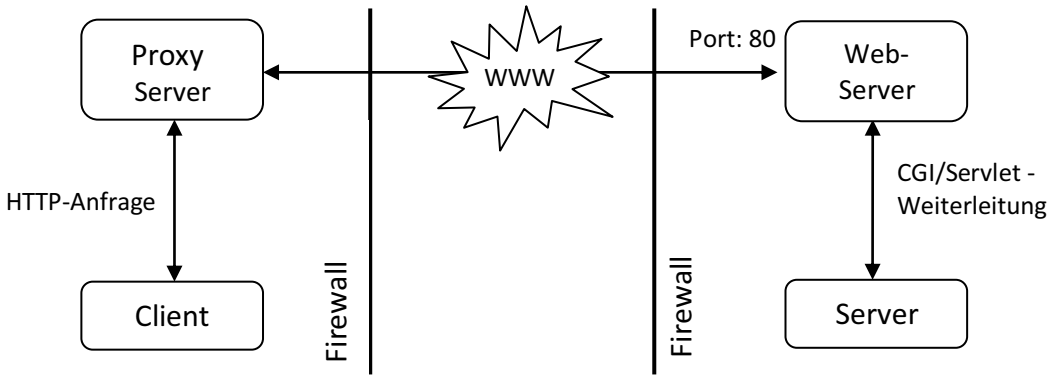


Abbildung 3-6 Tunnelung der RMI-Verbindung (nach [SunRMIa])

RMI ist besonders performant. Durch das Tunneln der Firewall werden alle Datenpake te zusätzlich in das HTTP Protokoll gekapselt und die Performance sinkt signifikant. Da für sind der erhöhte Overhead des Protokolls und die lose Verbindung zwischen Client und Server verantwortlich. Die Sockets werden also bei jeder Transaktion zerstört und wieder erzeugt. Bei einer Verbindung über JRMP bleiben die Sockets geöffnet. Beson ders nachteilig ist der Einsatz des Webservers. Das dort installierte CGI Programm muss die Daten zusätzlich weiterleiten. Solche Programme sind außerdem nicht be sonders performant. Es gibt zwar die Technologie der schnelleren Servlets, aber die Daten müssen trotz allem weitergeleitet werden. Je mehr Komponenten im Einsatz sind, desto höher ist das Sicherheits und Ausfallrisiko. Zudem steigt der Admi nistrationsaufwand enorm und damit die Kosten. Der Webserver ist auch nicht porta bel (bezogen auf Webserverimplementierung) und muss separat installiert werden. Ein Kompromiss ist die Öffnung der entsprechenden Ports auf dem Server. Der Client wird hingegen durch eine Firewall völlig geschützt gelassen. Dies erhöht das Sicherheitsrisi ko nicht, da der RMI Server vorher (ohne HTTP Tunnelung) auch adressierbar war (al lerdings nur über HTTP).

3.3.3 Ausnahmebehandlung

Ausnahmen werden in RMI sehr umfangreich behandelt (siehe Tabelle 3 11). Der Ent wickler kann Fehlerquellen schnell erkennen, was einen sicheren Betrieb von RMI Anwendungen unterstützt. Die RMI API[7] (siehe [SunRMIAPI]) beschreibt die Bedeutun gen der Ausnahmen explizit.

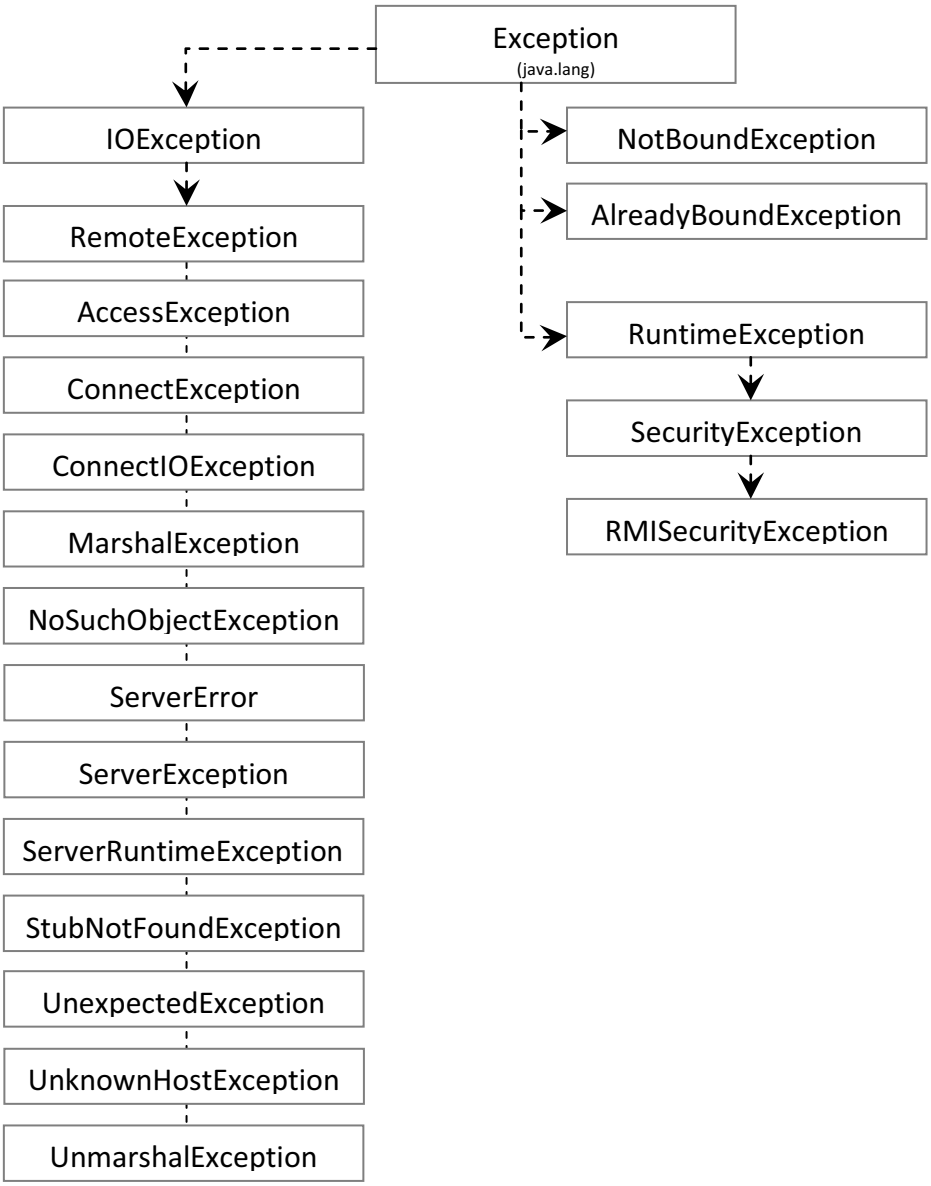


Tabelle 3-11 Ausnahmebehandlungen im Packet java.rmi (nach [SunRMIAPI])

[7] In der Softwareentwicklung bezeichnet API (engl. *Application Programming Interface*) eine Schnittstelle zur Anwendungsprogrammierung.

3.4 Technologieentscheidung

Tabelle 3 12 zeigt die drei Technologien im Vergleich. Dabei ist zu beachten, dass sich der Vergleich explizit auf den systemneutralen automatisierten Abruf bezieht. Ziel ist es nicht eine Middleware für eine komplexere Anwendung zu entwickeln, die als Schnittstelle für viele Anwendungen eingesetzt wird, sondern es soll lediglich die Ver bindung zwischen den Kommunikationspartnern abstrakt überbrückt werden. Die Ein schätzung des Programmieraufwands ist durch die Größe des Projekts relativ zu sehen. Für komplexere Systeme können die Konstellationen variieren.

	RMI	**CORBA**	**SOAP**
Programmiersprachen	Java	C, C++, Java, Smalltalk, Lisp, Cobol etc.	unabhängig
Plattform	Windows, Linux, Mac OS X, Solaris	abhängig von Pro-grammiersprache	unabhängig
Objektmodell	Java	IDL	nicht definiert
Programmieraufwand	++	--	-
Protokoll	JRPM (HTTP-Tunnelung möglich)	beliebig (HTTP-Tunnelung möglich)	beliebig (meist HTTP)
Dienste	standardisiert mit zahllosen Frame-works	standardisiert (CORBA-Services)	nicht standardisiert
Performance	+	+	--
Eignung in heterogenen Systemen	+(IIOP)	++	++
Skalierbarkeit	-	+	++
Herstellerabhängigkeit	ja	nein	Nein
Koppelung Client/Server	fest (TCP)	fest (TCP) / lose (UDP)	fest (Realisiert über Session-ID) / lose
Ausnahmebehandlung	Java Exceptions	IDL Exceptions	SOAP Faults
Kodierung	binär	binär	XML

-- sehr schlecht – schlecht + gut ++ sehr gut

Tabelle 3-12 Vergleich der Technologien

Der systemneutrale automatische Abruf ist ein kleines System in eigenständiger Um gebung. Abgesehen von den Schnittstellen ist eine Kommunikation mit anderen hete rogenen Anwendungen nicht geplant. Die Anwendungen sollten daher nicht kompli zierter umgesetzt werden als erforderlich. Grundsätzlich ist eine Lösung mit jeder Technologie für jede verbreitete Plattform möglich. In Java wird praktisch die Platt

formunabhängigkeit durch das Konzept der virtuellen Maschine (siehe Abschnitt 3.5.1) erreicht. Wesentliche Unterschiede sind zwischen RMI, CORBA und SOAP insbesondere im Abstraktionsniveau zu finden. SOAP bietet lediglich die Spezifizierung des Proto kolls, wohingegen CORBA und RMI eine vollständige Lösung der verteilten Anwendun gen anbieten. RMI ist sogar direkt an Java geknüpft. Frameworks sind dafür bei CORBA und RMI bereits nativ integriert.

CORBA kann seine Vorteile im Bereich der Plattform und Programmiersprachenunab hängigkeit ausspielen, was den Vorteil der exzellenten Skalierbarkeit bietet. Gerade bei großen Projekten in heterogenen Umgebungen ist dies nötig. Allerdings muss durch die Transparenz der Implementierung die eigene Programmsprache IDL erlernt wer den.[8] Für dieses kleine Projekt ist das jedoch als Nachteil anzusehen, da entsprechen des Know How vorhanden sein muss. Zudem handelt es sich um ein homogenes Sys tem. Mit einer Programmiersprache ist daher genügend Flexibilität gegeben. Das gilt sowohl für die Entwicklung der Anwendung als auch für die Wartung und Weiterent wicklung im Software Lebenszyklus. Hier sind höhere Kosten und Entwicklungszeit räume durch IDL zu erwarten. Die Verbindung muss verbindungsorientiert über TCP/IP erfolgen. CORBA bietet weitere Lösungen an, die aber auch in Zukunft nicht für das Projekt benötigt werden. Die CORBA Architektur ist ein sehr mächtiges Instrument in verteilten Systemen. Insgesamt scheint CORBA für den automatisierten Abruf jedoch überdimensioniert.

SOAP zeigt seine Stärken ebenfalls in der Skalierbarkeit der Anwendung sowie bei der Kompatibilität gegenüber Firewalls. Wobei dieses Konzept dennoch kritisch zu sehen ist, da Firewalls über den wahren Inhalt der Nachricht getäuscht werden. Anfänglich wurde dies als Vorteil gewertet. Heute wird dieses Konzept jedoch als Risiko eingestuft (siehe [Dostal, et al., 2005], Seite 40). Außerdem wird der Einsatz von intelligenten Fi rewalls provoziert, die jede Nachricht zusätzlich zur Port und Protokoll Validierung auch inhaltlich analysieren. CORBA und RMI haben hier ebenfalls die Möglichkeit über

[8] CORBA Version 3.0 bietet *reverse mapping*, dadurch kann IDL aus Java-Code erzeugt werden. Der Entwickler benötigt daher keine IDL-Kenntnisse. Zudem sind Java-Objekte über CORBA zugreifbar. RMI und IDL sind allerdings nicht kongruent.

HTTP zu tunneln. Dies ist dagegen als Notlösung zu betrachten, weil das Programm da durch langsamer und fehleranfälliger wird. Da das Abruf Programm hauptsächlich für den Transfer von größeren Daten konzipiert ist, eignet sich das XML Format nicht be sonders für diesen Einsatz. Durch die Speicherung der Daten in einer nicht binären Form wird zusätzlich zum HTTP Protokoll Overhead erzeugt. Die Menschenlesbarkeit des Übertragungsprotokolls ist nicht gefordert. Das Argument der Menschenlesbarkeit ist zudem relativ zu sehen. Eine längere SOAP Nachricht ist sehr kryptisch und daher von Menschen nur sehr schwer und unbequem zu lesen. Das XML Format ist außer dem durch die Tags deutlich länger als Nachrichten von CORBA und RMI. Der Einsatz des Parsers nimmt zusätzliche Ressourcen in Anspruch. Dies geht zu Lasten der Per formance und erhöht Kosten für den Datentransfer. Vergleicht man die Fehlertoleranz gegenüber CORBA oder RMI, so erscheint diese recht rudimentär. Die Spezifikation zu SOAP lässt einige Fragen offen. Für eine praktische Implementierung müssen zusätzli che Schnittstellen zwischen Client und Server beschrieben werden. Es ist außerdem fraglich, ob eine massive Skalierbarkeit notwendig ist, da er Aufwand für die Imple mentierung hier enorm wäre. Selbst Prozeduraufrufe müssten dafür implementiert werden. Prozeduraufrufe werden durch die Spezifikation nicht konkret bestimmt. Le diglich eine beispielhafte Beschreibung ist dabei vorhanden.

Mit RMI steht ein Framework zur Verfügung, dass sich nahtlos in Java einordnet. Si cherheitskonzepte, die in Java bestehen, können ebenfalls für RMI genutzt werden. Zudem besteht eine hervorragende Verzahnung mit externen Bibliotheken. Die Sicher heitskonzepte sind im Gegensatz zu SOAP standardisiert und die RMI Implementierun gen überzeugen durch ihre Einfachheit und Integrität. Methodenaufrufe sind hierbei standardisiert. Zu beachten ist die Bindung an Java als Programmiersprache. Eine Kombination mit anderen Komponenten ist nur mit erhöhtem Aufwand möglich. Aller dings ist eine Interaktion mit CORBA über IIOP denkbar. Das Abruf System wird jedoch als ein abgeschlossenes autonomes System bestehen. Eine Erweiterung zu einer servi ce orientierten Architektur (*SOA*[9]) wird demnach nicht geplant.

[9] Eine serviceorientierte Architektur (*SOA*) ist ein Paradigma zur Organisierung und Nutzung verteilter Funktionalitäten. Jede Funktionalität liegt dabei in der Verantwortung des jeweiligen Nutzers (siehe [OASIS]).

Die Datenübertragung mit RMI ist sehr performant und eignet sich hervorragend für das Übertragen der Daten. Für das beschriebene Einsatzgebiet bietet RMI kongruente Funktionalitäten und wird daher als Middleware implementiert.

3.5 Java

Sun Microsystems hat mit Java eine vollständig objektorientierte Programmiersprache entwickelt, die derzeit wohl am häufigsten in der Anwendungsentwicklung eingesetzt wird. Zudem ist eine plattformunabhängige und umfangreiche Klassen Bibliothek ver fügbar (vgl. [Bell, et al., 2003]).

3.5.1 Java Virtual Machine

Schon beim Entwurf der Programmiersprache Java wurden wesentliche Aspekte der Portabilität berücksichtigt. Die Portierung von Programmen ist im Vergleich zu anderen Programmiersprachen sehr elegant gelöst. Beim Kompilieren in Java wird der Code in eine maschinenunabhängige Form gebracht (*Byte Code*). Die *Java Virtual Machine* (JVM) interpretiert dabei den *Byte Code* zur Laufzeit. Zudem wird nur benötigter Code von der JVM geladen, wobei dieser folglich schnell über ein Netzwerk ladbar ist. Eine große Anpassung des Programms ist demnach nicht mehr nötig. Hervorzuheben ist auch die Unabhängigkeit der Datentypen zu den Betriebssystemen in Java. Die JVM steht für eine große Anzahl von Prozessoren zur Verfügung. Außerdem ist die JVM für viele Betriebssysteme verfügbar (vgl. [Bell, et al., 2003])

3.5.2 Sandbox

Um den Computer vor böswilligem Code zu schützen setzt Java eine spezielle Technik ein. Die *Sandbox* steuert die Anwendung und verhindert das Ausführen verbotener Be fehle. Dies gilt insbesondere beim Zugriff auf Register oder Bereiche im Speicher, die nicht durch die JVM freigegeben sind. Bei verteilten Anwendungen müssen Klassen von einem Server herunterladbar sein. Der Klassenlader (engl. *Class Loader*) legt dabei fest, dass heruntergeladene Klassen nur ihren eigenen Klassenlader verwenden dürfen. Damit sind diese Klassen vertrauenswürdiger. Erst jetzt können Objekte aus der Klasse instanziiert werden. Der Sicherheitsmanager verhindert dabei den Zugriff auf unautori sierte Ressourcen während der Laufzeit (vgl. [Tanenbaum, et al., 2008]).

3.5.3 Sicherheitsmanager

Der *SecurityManager (*dt. Sicherheitsmanager) regelt die Zugriffe auf kritische Metho den in Java Programmen. Standardmäßig ist dieser nicht aktiv. Für RMI Anwendungen ist die Aktivierung jedoch erforderlich, da sich durch den *SecurityManager* allgemein gültige Sicherheitsrichtlinien setzen lassen. Weiterhin können Berechtigungen stark eingeschränkt werden. Jede Methode fragt vor ihrer Ausführung den *SecurityManager*, ob entsprechende Berechtigungen vorhanden sind. Es ist also nicht möglich eine dieser Methoden zur Laufzeit auszuführen, wenn diese nicht entsprechende Berechtigungen aufweist.

Die Rechtevergabe wird durch die Policy Datei geregelt. Vor dem Aufruf des *Security Mangers* muss der Laufzeitumgebung ein Verweis auf die Policy Datei gegeben wer den. Dies kann über einen Eintrag im Java Classpath getan werden oder über den Property Set während das Programm bereits läuft. Für die Zusicherung der Rechte können folgende zusätzliche Angaben gemacht werden:

- Codebase – es werden nur Klassen von einer bestimmten URI zugelassen,
- Signierung – nur signierter Programmcode wird zugelassen und
- Principal – authentifizierte Benutzer (vgl. [Ullenboom, 2008]).

```
grant {
permission java.net.SocketPermission "*", "accept,connect,listen,resolve";
};
```

Tabelle 3-13 Policy-Datei

Tabelle 3 13 zeigt ein Beispiel für die Policy Datei, indem sämtliche Socket Verbindun gen zugelassen werden. Denkbar wäre jedoch auch eine Sperrung von Schreibzugriffen auf dem Server (*java.io.FilePermission*). Versucht eine Methode z.B. eine Datei zu lö schen, so wird eine Ausnahme vom Typ *AccessControlException* ausgelöst. Eine voll ständige Übersicht aller möglichen Berechtigungen und viele weitere Beispiele sind un ter [SunPer] zu finden.

Das Laden der Policy Datei mit dem *SecurityManager* ist in aktuellen Java Versionen für den Einsatz von RMI obligatorisch. Daran ist die Wichtigkeit dieser Funktion zu er kennen.

3.5.4 JDBC

Die *Java Database Connectivity* (JDBC) bildet eine einheitliche Datenbankschnittstelle für Java Anwendungen. JDBC kann genutzt werden, um relationale Datenbanken ver schiedener Hersteller zu nutzen. Für das Abrufverfahren ist diese Datenbankschnitt stelle von grundlegender Bedeutung, um auf Daten zuzugreifen oder eigene speichern zu können. Basisfunktionen von JDBC sind:

- Herstellung der Datenbankverbindung und Zugriff auf Tabellen der Datenbank,
- Weiterleitung von SQL Anfragen und
- Verarbeitung von Ergebnissen.

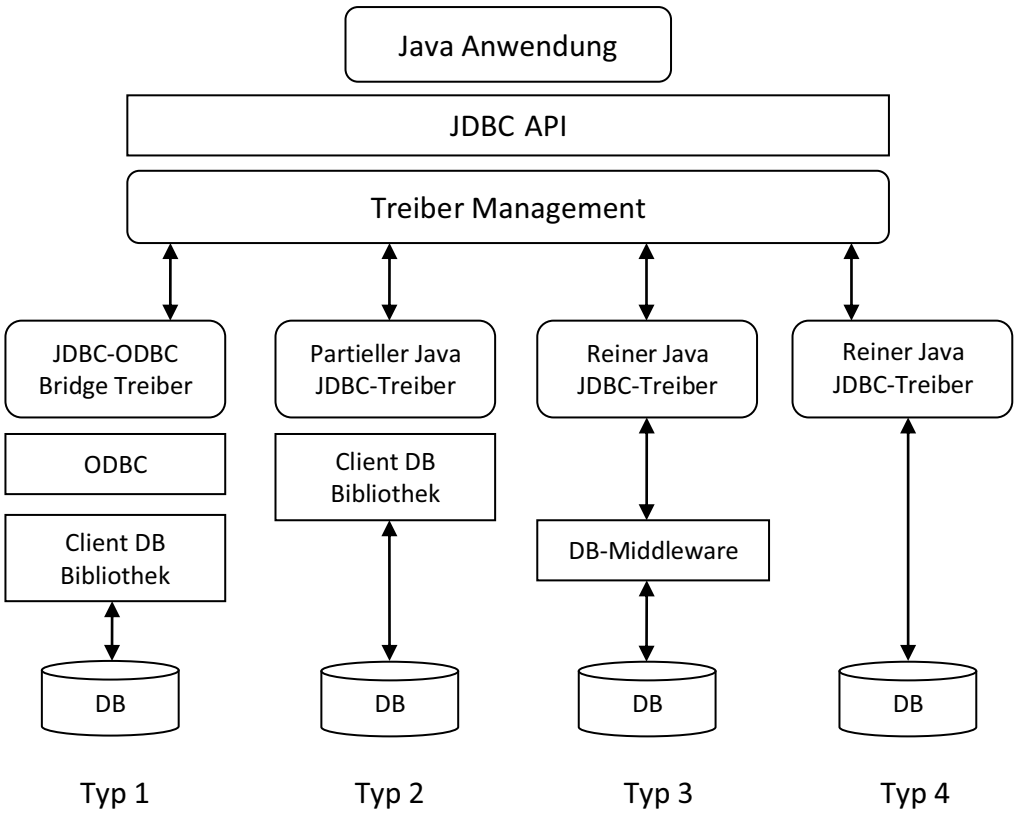


Abbildung 3-7 Treibertypen von JDBC (nach [SunJDBC])

Es existieren vier verschiedene Typen in der JDBC Architektur (siehe Abbildung 3 7). Typ 1 ist durch einen ODBC[10] Aufruf implementiert. Dabei ist ein installierter ODBC Treiber mit korrekter Konfiguration erforderlich. Der eigentliche Aufruf geschieht dabei außerhalb der Java Umgebung. Typ 2 greift auf die native Bibliothek des Datenbank managementsystems zu. In einer Umgebung mit Middleware sendet JDBC Typ 3 über ein unabhängiges Nachrichtenprotokoll Anfragen an die Datenbank Middleware. Ein Server empfängt dabei die Daten über die Middleware und regelt die Zugriffe auf die Datenbank. Sämtliche Antworten werden nun wieder über die Middleware an den Client weitergeleitet. Typ 4 ist die effektivste Methode, um auf eine Datenbank zuzug reifen, da nur Java Code genutzt wird (vgl. [Melton, et al., 2000]).

Mit JDBC steht eine mächtige Schnittstelle zur Verfügung. Der Entwickler kann sich ganz auf die Anwendungsentwicklung konzentrieren. Notwendig sind dabei allerdings die Herstellertreiber, falls diese nicht schon in Java nativ integriert sind (z.B. Access Da tenbanken).

3.6 Secure Sockets Layer

SSL steht für *Secure Socket Layer* und ist seit der Einführung durch Netscape Communi cations stark verbessert worden. Obwohl es sich dabei um einen proprietären[11] Stan dard handelt, entwickelte sich SSL durch die freie Spezifikation zu einem de facto Internet Standard. Die *Internet Engineering Task Force* (IETF) Arbeitsgruppe kam 1996 zusammen, um SSL zu einem Internet Standard weiterzuentwickeln. Die überarbeitete Version von SSL nannte sich TSL (Transport Layer Security). Da sich TSL nur geringfügig von SSL unterschied, wird TSL auch als SSL 3.1 bezeichnet. SSL ist im OSI Modell ober halb der Transportschicht angesiedelt. Es kann somit für jedes Transportprotokoll ver wendet werden, meistens aber für TCP. Die größte Anwendung ist im HTTPS Protokoll zu finden, wobei das HTTP Protokoll in SSL gekapselt wird.

[10] ODBC (*Open Database Connectivity*) ist eine Datenbankschnittstelle von Mircosoft, die ebenfalls SQL als Datenbanksprache verwendet. Für die Verwendung einer Datenbank ist der entsprechende Treiber zu installieren.

[11] Proprietäre Protokolle sind nur schwer in Software zu implementieren, da sie lizenzrechtlich oder durch Patente geschützt sind.

Hervorzuheben ist, dass der Anwender aus einer Reihe von Kryptographieverfahren auswählen kann. Beim „Handshake Dialog“ wird zwischen Client und Server verein bart, welches Verfahren zum Schutz der Verbindung verwendet werden soll. Dabei werden drei Algorithmen festgelegt: der Algorithmus für Authentifizierung, Schlüs selaustausch und für die Integritätsüberprüfung. Diese Festlegung wird *CipherSuite (dt. Chiffrierfolge)* genannt. Jeder *ChipherSuite* hat die Form *SSL_asym_WITH_syn_mac.*

Beispiel für einen CipherSuite: *SSL_RSA_WITH_3DES_EDE_CBC_SHA*

Durch die Zeichenkette sind die verwendeten Verfahren ohne Weiteres ablesbar. *„Asym“* steht für das asymmetrische Verfahren, welches für die Authentifizierung und für den Schlüsselaustausch verwendet wird. Sowohl für die Authentifizierung (RSA oder Diffie Hellmann) als auch für den Schlüsselaustausch (RSA oder DSA) kann zwi schen verschiedenen Verfahren gewählt werden. Für die symmetrische (*„sym“*) Ver schlüsselung können noch weitere Verfahren gewählt werden. Dabei werden Blockver schlüsselungsalgorithmen[12] verwendet. Die Bezeichnung *„mac“* steht für den verwen deten Hashalgorithmus. SHA 1 und MD5 stehen dabei zur Auswahl zur Verfügung. Durch die Flexibilität von SSL ist es möglich die Sicherheitsverfahren für eigene An wendungen individuell zusammenzustellen. Der Entwickler muss sich nicht im Detail mit der Implementierung von Sicherheitsalgorithmen befassen, Priorität hat die Si cherheit. Ein Kompromiss zwischen Performance und Sicherheitslevel ist dabei prakti kabel. Diese Verschlüsselungsverfahren werden in symmetrische und asymmetrische Verfahren gegliedert (vgl. [Lipp, et al., 2000]).

3.6.1 Symmetrische Verfahren

Bei symmetrischen Verfahren werden für die Ver und Entschlüsselung dieselben Schlüssel verwendet. Sender und Empfänger tauschen den Schlüssel über einen siche ren Kanal aus. Der Schlüssel kann jedoch auch von einer dritten Stelle an die Kommu nikationsteilnehmer verteilt werden. In jedem Fall muss der Schüssel geheim gehalten werden. Daher wird das Verfahren auch als *Secret Key Verfahren* bezeichnet. Jedoch muss das Problem des Schlüsselaustauschs gelöst werden. Für Angreifer ist es oft ein

[12] Daten werden in Blöcken fixer Länge verarbeitet (block cipher).

facher den Schlüssel abzufangen als die Schlüssel durchzuprobieren. Ein weiterer Nachteil ist die Anzahl der benötigten Schlüssel. Für jeden Kommunikationsteilnehmer wird ein eigener Schlüssel benötigt (vgl. [Lipp, et al., 2000]).

$$k = n * \frac{n-1}{2}$$

n- Anzahl der Kommunikationsteilnehmer

k- Anzahl der benötigten Schüssel

3.6.2 Asymmetrische Verfahren

Beim asymmetrischen Verfahren sind die Schlüssel für die Ver und Entschlüsselung verschieden. Schon bei der Generierung ebensolcher wird ein Schlüssel für die Ver schlüsselung erzeugt und ein anderer für die Entschlüsselung. Da es nicht möglich ist, von einem Schlüssel auf den weitere zu schließen, kann einer der beiden Schlüssel öf fentlich bekannt gegeben werden. Dieses Verfahren wird dann auch als *Public Key Verfahren* bezeichnet. Hauptproblem bei asymmetrischen Verfahren ist die Perfor mance, da häufig Primzahlen verwendet werden. Diese liegen weit entfernt voneinan der. Daher müssen sehr lange Schlüssel mit 1024 oder 2048 Bit verwendet werden. Nur so kann die Sicherheitsqualität wie bei symmetrischen Verfahren erreicht werden. Allerdings ist das Problem des Schlüsselaustauschs nicht existent(vgl. [Lipp, et al., 2000]).

3.6.3 Digitale Signaturen

Möchte der Absender einer Nachricht dem Empfänger beweisen, dass die Nachricht von ihm ist, so braucht er nur die Nachricht mit dem privaten Schüssel verschlüsseln (siehe Abbildung 3 8). Dabei entsteht die digitale Signatur, die auch als digitale Unter schrift bezeichnet wird. Aus Performancegründen wird nicht die gesamte Nachricht verschlüsselt, sondern nur eine Prüfsumme erstellt. Diese wird mit einer Hashfunk tion[13] zuvor erzeugt. Die Nachricht kann nur mit dem öffentlichen Schlüssel entschlüs selt werden. Da aber nur der Absender Zugang zum privaten Schüssel hat, muss die Nachricht von ihm sein. (vgl. [Lipp, et al., 2000]).

[13] Hashfunktionen erstellen Prüfsummen (kompakter, eindeutiger Repräsentant von Daten beliebiger Länge).

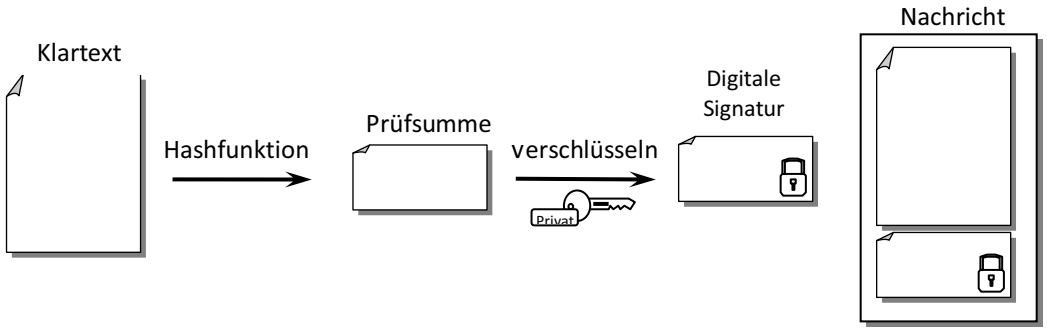


Abbildung 3-8 Erstellung einer digitalen Signatur (nach [Lipp, et al., 2000])

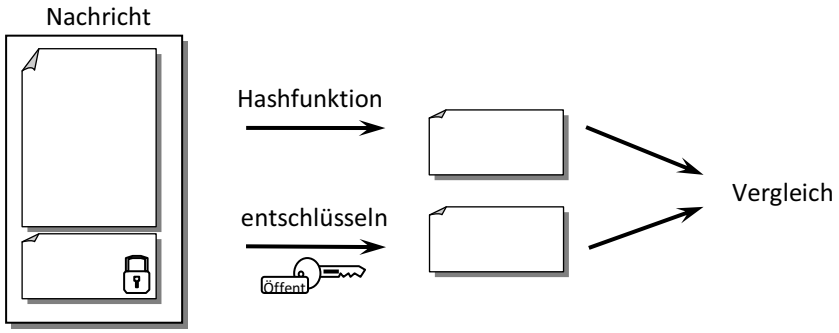


Abbildung 3-9 Prüfen einer digitalen Signatur (nach [Lipp, et al., 2000])

Der Empfänger kann durch das Verfahren auch die Unversehrtheit der Nachricht über prüfen, damit ist eine Manipulation weitgehend ausgeschlossen (siehe Abbildung 3 9).

3.6.4 SSL in Java

Die Java Secure Socket Extension (JSSE) ist eine Erweiterung in Java zur Sicherung der Verbindung in unsicheren Netzwerken, wie z.B. das Internet. Seit der Version 1.4 ist JSSE in Java enthalten.

4 Entwurf

In diesem Abschnitt werden Komponenten logisch eingeteilt, um eine Implementierung zu vereinfachen. Eine weitere Verfeinerung sowie die Beschreibung des technischen Aufbaus jeder Komponente sind notwendig. Einige Module des Systems müssen über Schnittstellen kommunizieren, da sie keinen Einfluss auf die gegenüberliegende Komponente haben.

4.1 Architektur

Da es sich bei der Middleware des automatisierten Abrufs um eine verteilte Anwendung handelt, lässt sich diese in mehrere logische Schichten *(*engl. *tier)* einteilen (siehe Abbildung 4 1). Eine solche Einteilung kann logisch nach der Funktionalität erfolgen. Die Datenschicht ermöglicht die persistente Speicherung der Liegenschaftsdaten. Dies wird meist durch ein Datenbankmanagementsystem realisiert. Darauf baut die Anwendungsschicht auf, die alle anwendungsspezifischen Regeln der Verarbeitung implementiert. Darüber liegt die Präsentationsschicht, die oft durch eine graphische Oberfläche dem Anwender die Daten darstellt und Schnittstellen zur Interaktion liefert.

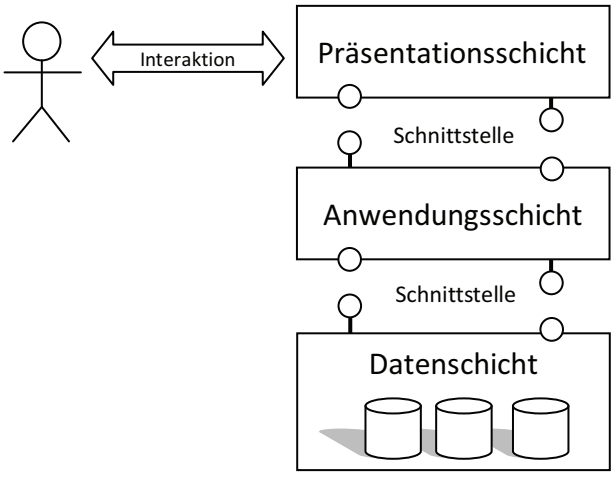


Abbildung 4-1 Zerlegung der Anwendung (nach [Heinzl, et al., 2005])

Es ist möglich, dass jede Schicht auf einem separaten Rechensystem gehalten wird. Dies wird als 3 Schicht Architektur *(*engl. *Three Tier Architecture)* bezeichnet. Es ist je

doch auch möglich die Architektur in weniger oder mehr Schichten zu teilen. In Ab schnitt 4.3 werden die einzelnen Komponenten einer Schicht zugeordnet. Wird durch den Client mehr als nur das Anzeigen von Daten erbracht, heißt dieser *Fat Client*. Übermittelte Daten werden demnach clientseitig weiterverarbeitet (vgl. [Heinzl, et al., 2005]).

4.2 Pakete

Für die Veröffentlichung (engl. *Release*) der Software werden die Komponenten in Pa kete logisch aufgeteilt. Diese können später beliebig viele Klassen oder weitere Pakete enthalten. Die Verteilung (engl. *Deployment*[14]) der beiden Programme kann neben der Wiederverwendbarkeit von Paketen optimiert werden. Die logische Einteilung ist ab hängig von der Zugehörigkeit von Client oder Server. Im Offline Betrieb kann der Ser ver nur seine lokalen Methoden nutzen. Erst beim Aufruf der entfernten Methoden durch den Client kann durch die hergestellte Verbindung der Zugriff auf diese erfolgen. Bei der späteren Veröffentlichung der Software muss jedes Programm die zur Laufzeit benötigten Klassen bereitstellen.

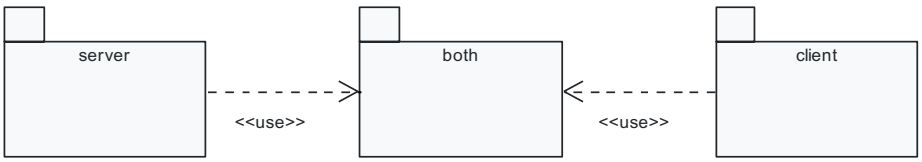


Abbildung 4-2 Paketdiagramm des Abruf-Programms

Einige dieser Klassen bieten Funktionen, die sowohl vom Server als auch vom Client genutzt werden müssen. Daher werden diese in das Paket „both“ abgelegt. Vor dem Transfer der Liegenschaftsdaten werden z.B. alle Dateien vom Server komprimiert und beim Client wieder dekomprimiert. Eine Klasse vom Paket „both“ stellt Methoden be reit, die Dateien dekomprimieren und komprimieren kann. Dadurch werden Redun danzen in der Datenhaltung der Entwicklungsumgebung vermieden. Außerdem ist da

[14] Die Verteilung beschreibt den Prozess zur Installation von Software auf Anwender-PCs (siehe [WikiDepl]).

durch eine bessere Übersicht bei der Entwicklung gegeben. Abbildung 4 2 zeigt das Pa ketdiagramm des Systems.

4.3 Anwendungskomponenten

Abbildung 4 3 illustriert die Komponenten, die für die Implementierung des Servers notwendig sind. Wie in Abschnitt 4.1 erläutert wurde, ist jede Komponente Teil einer bestimmten Schicht. Im Zentrum steht die Anfrageverarbeitung. Ihre Aufgabe ist es, die Anfragen von Clients zu verarbeiten. Gleich zu Beginn wird über die Schnittstelle zur Benutzerverwaltung die Autorisierung des Nutzers geprüft. Die Nutzerdaten könn ten dabei in derselben Datenbank liegen, welche auch für die Datenhaltung bereitges tellt wird. Es ist natürlich auch eine eigene Datenbank für die Benutzerverwaltung denkbar. Zudem muss die Komponente zwischen Datenanfragen und Datenholungs anfragen unterscheiden. Für eine automatische Datenerzeugung wird über eine Schnittstelle die Datenerzeugung anhand der Nutzerdaten eingeleitet. Anwender kön nen auch parallel Zugriffe auf die Server Methoden durchführen. Die Protokollierung registriert sämtliche Anfragen. Einzelne Kriterien zur Protokollierung wurden bereits in Abschnitt 2.5 (§2 (1)) dargestellt. Besonders wichtig ist die Schnittstelle zur Protokollie rungsdatenbank. Dabei kann eine Anbindung an die lokale Datenbank erfolgen. Die Komponente Datenaustausch beschreibt nicht zur den Datentransfer, sondern diese regelt auch die Kommunikation zwischen Client und Server. Eine eigene Komponente sorgt dafür, dass dieser Datenverkehr gesichert wird.

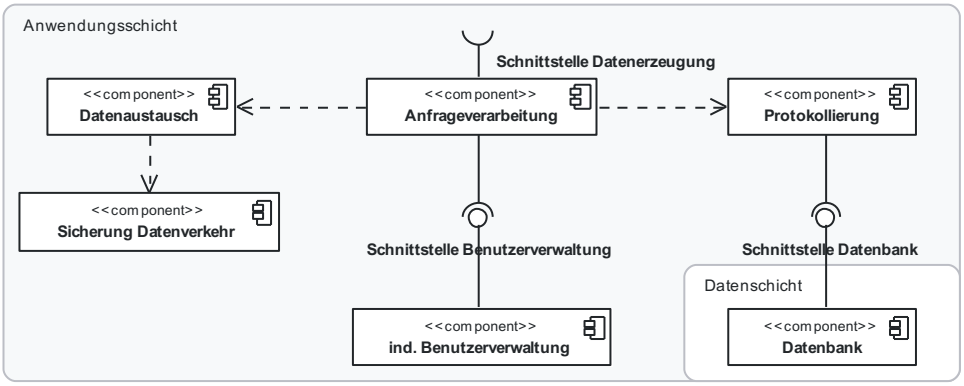


Abbildung 4-3 Serverkomponenten

In Abbildung 4 4 sind die Komponenten des Client dargestellt. Zahlreiche Anwendungskomponenten verdeutlichen den Entwurf als *Fat Client*. Das Kernstück ist dabei die Anfragezusammenstellung. Die Aufgabe der Komponente ist die Anmeldung am Server und die Anfrageorganisation. Deswegen ist diese Komponente abhängig von der Zeitplanung. Handelt es sich also um keine durch den Anwender ausgeführte Anfrage (manuelle Anfrage), so regelt einzig die Zeitplanungskomponente wann Anfragen gestellt werden. Die dort eingestellten Parameter sind für den Zeitpunkt und Zyklus der Anfragestellung verantwortlich. Über die graphische Oberfläche werden Einstellungen konfiguriert. Diese werden mittels Konfigurationsmanagement in eine Konfigurationsdatei gespeichert. Die Parameter für Hostname, Benutzername, Passwort und Ablageort werden dort ebenfalls hinterlegt. Außerdem ist es möglich neue Serververbindungen zu erstellen sowie zu löschen. Ohne Serverparameter ist eine Kontaktaufnahme nicht möglich. Sämtliche Anfragen werden registriert, um die Aktionen des Clients nachprüfen zu können. Hier müssen auch Ausnahmen oder Warnungen protokolliert und angezeigt werden. Diese Aufgabe übernimmt die Logging Komponente. Über die graphische Oberfläche können diese Informationen per „Konsole“ eingesehen werden. Bei dem eigentlichen Abruf der Liegenschaftsdaten werden die Dateien von einem Modul entgegengenommen. Die Kommunikation zum Server ist dabei permanent gesichert. Der Import wird nach Abschluss des Datentransfers über die Schnittstelle zur Liegenschaftsverwaltungssoftware eingeleitet.

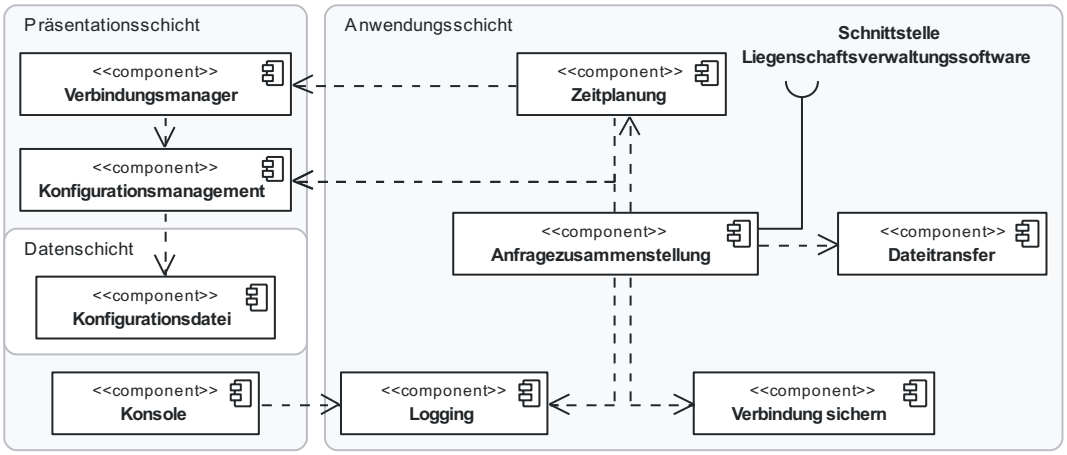


Abbildung 4-4 Clientkomponenten

4.4 Anfragen

Durch die getrennte Implementierung der Datenanfrage und Datenholungsanfrage muss der Server zwischen diesen beiden Anfragen unterscheiden (siehe Abbildung 4 5). Sind bei einer Datenholungsanfrage noch keine Daten vorhanden, so muss dies dem Client mitgeteilt werden. Ein Transfer wird erst dann eingeleitet, wenn die Daten zusammengestellt sind. Abbildung 4 5 zeigt die beiden möglichen Fälle der Datener zeugung (automatische und manuelle Datenerzeugung) sowie die Differenzierung des Server bei der Anfrageverarbeitung.

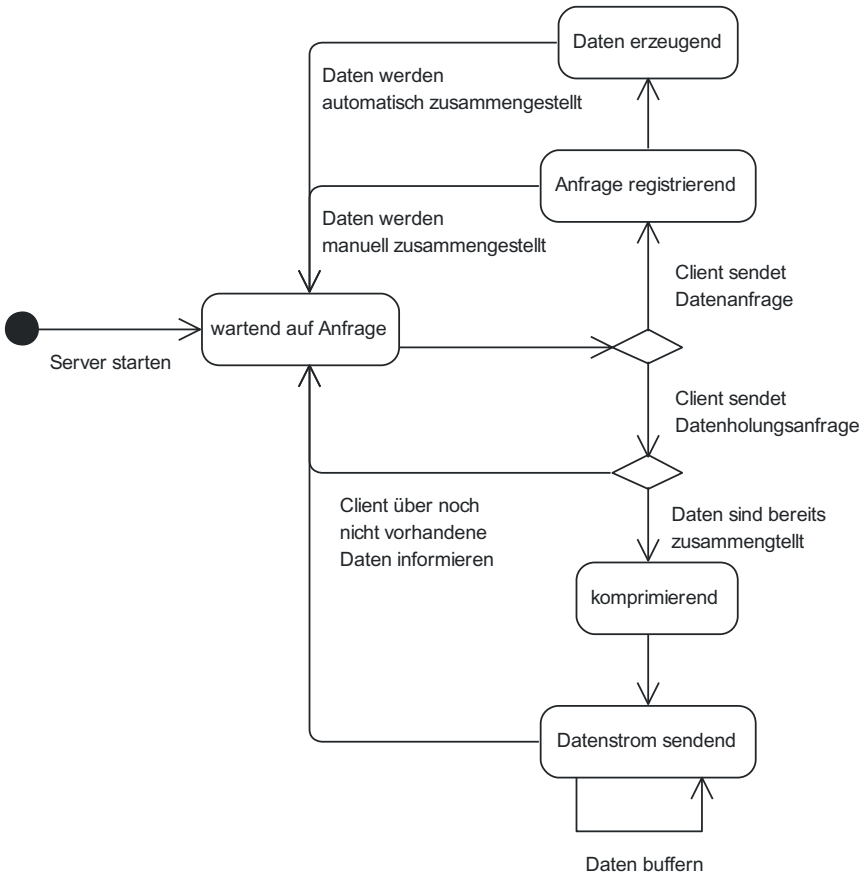


Abbildung 4-5 Zustandsdiagramm der serverseitigen Anfrageverarbeitung

4.4.1 Anfrage-Organisation

In Abschnitt 2.9 wurde bereits auf die Problematik der getrennten Implementierung von Datenanfrage und Datenholung hingewiesen. Der Client stellt dabei eine Daten anfrage an den Server der diese registriert. Nachdem die Daten erzeugt wurden, kann der Client in einer erneuten Anfrage (Datenholungsanfrage) feststellen, ob die Daten

schon zusammengestellt sind. Erst danach kann mit dem Herunterladen begonnen werden. Dazu muss jedoch eine Referenz auf die vorherige Datenanfrage vorliegen, welche vom Server interpretiert werden muss. Der Server als auch der Client kann die Haltung der Referenzen übernehmen. Beide Methoden haben Schwachstellen. Liegen die Referenzen nur auf dem Server, so muss der Client eine Verbindung zum Server aufbauen. Der Client registriert die gesendeten Anfragen, welche anhand des Benut zernamens zugeordnet werden können.

Hält der Client eine Referenz auf die Datenanfrage lokal, so muss eine redundante Re ferenz beim Server vorliegen (siehe Abbildung 4 6). Dies ist notwendig, da der Server auch die zusammengestellten Daten einer Datenanfrage zuordnen muss. Mit anderen Worten ist dies auch als clientseitiges Cachen[15] zu bezeichnen.

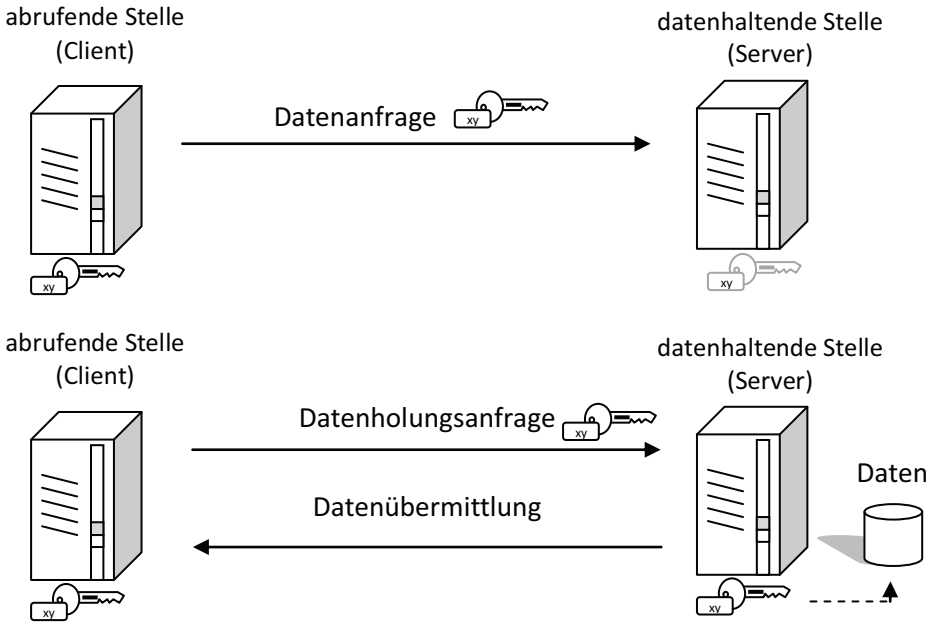


Abbildung 4-6 Referenzhaltung von Anfragen

Als praktikabel erweist sich die redundante Referenzhaltung. Durch eine saubere Im plementierung werden die Probleme der redundanten Referenzhaltung sicher ge handhabt. Besonders wichtig sind hier Ausnahmen, um mögliche Inkonsistenzen abzu fangen. Dies kann z.B. eintreten, wenn die Referenz in der Server Datenbank nicht mehr vorhanden ist und der Client versucht die Datenholungsanfrage zu stellen. Die

[15] Das Cachen bezeichnet eine lokale Speichervorrichtung, die nur vom Client verwaltet wird. Ziel dabei ist es, die Zugriffszeiten auf den Server zu verbessern, bzw. beim systemneutralen Abrufverfahren den Zugriff auf den Server zu vermeiden (vgl. [Tanenbaum, et al., 2008]).

veraltete Anfrage kann gelöscht worden sein. Um Speicherplatz zu sparen und die Übersichtlichkeit zu erhalten können dabei veraltete Anfragen durch einen eigenen Al gorithmus gelöscht werden. Dem Client wird nun mitgeteilt, dass die Anfrage nicht mehr vorhanden ist und eine neue Anfrage gestellt werden muss.

4.4.2 Anfragereferenz-Datenhaltung

Für die Datenhaltung der Anfragereferenzen bietet sich eine dauerhafte (engl. *durable)* Datenhaltung an. Würde eine Speicherung nur zur Laufzeit im Speicher erfol gen, wären alle Referenzen bei einem Systemabsturz verloren. Die datenhaltende Stel le richtet dazu eine Datenbank ein, welche alle Anfragen registriert. Die abrufende Stelle sammelt die ausgegangenen Anfragen in einer Datei. Die Speicherung wird dabei vom Konfigurationsmanager (siehe Abbildung 4 4) übernommen.
Die Referenz für die Anfrage muss einen eindeutigen Repräsentanten haben. Standar disiert sind Identifikatoren durch die Internet Engineering Task Force (IETF) (siehe [RFC4122]). Unter der Spezifikation des Universally Unique Identifier (UUID) existiert ein Standard, um Informationen in verteilten Anwendungen eindeutig kennzeichnen zu können. Dabei handelt es sich um eine feste Länge von 128 Bit, die in fünf Gruppen ge gliedert wird.

Beispiel der Normalform abgebildet als URN[16]:

2973145a 0bf8 47b4 a368 0d273871005e

Es wird allerdings nicht garantiert, dass die Zahl wirklich eindeutig ist. Die Wahrschein lichkeit ist mit 2^{128} ($\cong 3{,}4 \times 10^{38}$) jedoch sehr gering. Tritt dieser Fall ein, so wird der aktuellere Eintrag zurückgegeben. Ein zentralisiertes Organ zum Abgleich der Identifier wird nicht benötigt. In der Java API existiert eine Implementierung zur Erzeugung von UUIDs im Paket java.util.UUID.

[16] Ein URN (Uniform Resource Name) bezeichnet eine dauerhafte, ortsunabhängige Ressource.

4.5 Kommunikationssicherheit

Die Kommunikation für den Abruf der Liegenschaftsdaten wird durch die in SSL ver wendeten Verfahren geschützt. Dabei werden die Vorteile von symmetrischen Verfah ren für den Datentransfer mit den Vorteilen von asymmetrischen Verfahren kombi niert. Die Authentizität, Vertraulichkeit und Integrität der Daten ist somit gesichert. Das für den Prozeduraufruf verwendete Java Remote Method Protocol (JRMP) wird dabei von SSL geschützt. Der Verbindungsaufbau wird verbindungsorientiert (*TCP*) durchgeführt. In Abbildung 4 7 ist die Schichteinteilung der Kommunikation darge stellt.

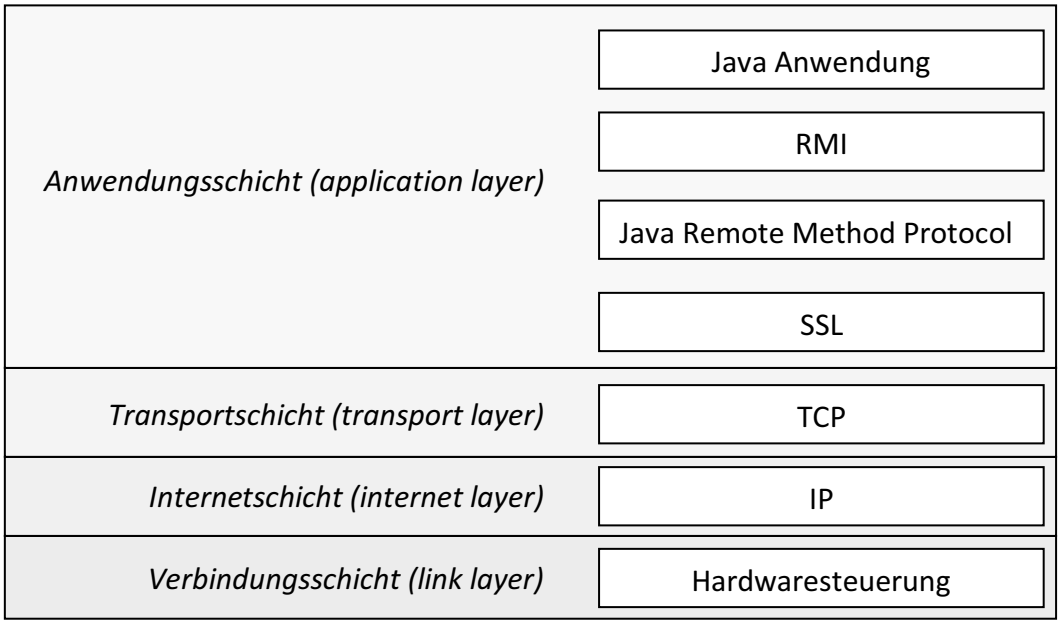


Abbildung 4-7 Schichteinteilung nach RFC1122 (nach [RFC1122])

Um sogenannte Man In The Middle Angriffe[17] zu unterbinden, muss ein Zertifikat ein gesetzt werden. Neben zahlreichen kommerziellen Zertifizierungsstellen existiert auch eine nicht kommerzielle Zertifizierungsstelle (*CAcert*). Akkreditierte Anbieter von quali fizierten Zertifikaten gemäß deutschem Signaturgesetz sind auf der Internetpräsenz der Bundesnetzagentur aufgeführt (siehe [BNetzA]). Dadurch reduziert sich die Aus wahl der Zertifizierungsstellen deutlich.

[17] Bei MITM (Man-In-The-Middle)-Angriffen steht der Angreifer zwischen den Kommunikationspartnern und versucht die komplette Kontrolle über den Datenverkehr zu erlangen. Informationen können dabei eingesehen und manipuliert werden. Die Kommunikationspartner merken von dem Eingriff nichts (vgl. [WikiMITM]).

Es ist zu beachten, dass sowohl Passwörter als auch Zertifikate einer gewissen Abnut zung unterliegen. Je älter diese sind, desto schlechter kann die reale Sicherheit einge schätzt werden. Eine regelmäßige Änderung der Passwörter und Zertifikate ist daher empfehlenswert. Nach einer bestimmten Zeit verlieren Zertifikate allgemein ihre Gül tigkeit und werden widerrufen (engl. *revocation*). Das Zertifikat muss dann rechtzeitig ausgetauscht werden.

4.6 Dateitransfer und Kompression

Sämtliche Daten werden verschlüsselt übertragen (siehe Abschnitt 2.8.5). Um die Per formance des Datentransfers zu erhöhen, ist die Komprimierung der Daten erfor derlich. Dabei stehen zwei verschiedene Verfahren zur Auswahl:

- Komprimierung von vollständigen Dateien und
- Komprimierung einzelner Datenblöcke.

Die blockweise Kompression hat den Vorteil, dass zur Laufzeit die Blöcke dynamisch komprimiert werden. Jedoch ist diese Methode weniger effektiv, als die ganzer Datei en. Zudem muss diese Datei zwangsläufig komplett komprimiert werden. Der größte Nachteil der kompletten Komprimierung ist die lange Wartezeit des Clients zu Beginn des Transfers. Insbesondere bei großen Dateien können hier die Wartezeiten mehrere Sekunden betragen. Trotzdem überwiegen die Vorteile dieses Verfahrens.

4.7 Schnittstellen

Es ist unbedingt notwendig die Kompatibilität zu den Subsystemen Benutzerver waltung, Datenbank und Liegenschaftsverwaltungssoftware zu gewährleisten. Nur dann kann der automatisierte Abruf systemneutral erfolgen, indem es sich in beste hende heterogene Systeme eingliedert.

4.7.1 Benutzerverwaltung

Für die Sicherheit des Systems unverzichtbar ist die Benutzerverwaltung. Hierdurch kann die Autorisierung erfolgen. Da die meisten Systeme den de facto Standard LDAP (*Lightweight Directory Access Protocol*) nutzen, kann eine gute Anpassung vollzogen

werden. Bei LDAP handelt es sich um ein „leichtgewichtiges“[18] Anwendungsprotokoll für die Abfrage und Modifikation von Verzeichnisdiensten (engl. *directory services*), besonders für Authentifizierung, Autorisierung, Benutzer und Adressverzeichnisse. LDAP fungiert dabei selbst als Schnittstelle der Benutzerverwaltung. Vom LDAP Server können sämtliche relevante Nutzerdaten abgefragt werden. Jeder Anwender kann dazu in verschiedenen Rechte Gruppen vertreten sein. Dies ist sinnvoll, wenn die datenhaltende Stelle noch weitere Dienste extern oder intern bereitstellt (vgl. [RFC4510]). Besonders interessant ist die Microsoft Implementierung durch den *Active Directory Service*. Ein schon vorhandener Server mit Windows Server Betriebssystem kann so ohne größeren Aufwand die Benutzerverwaltung bereitstellen. Die Kernkomponenten des *Active Directory Domain Services* organisiert dabei die Benutzerkonten jedes einzelnen Nutzers. Vorteil dieser Implementierung ist die einfache Handhabung, da die Steuerelemente der GUI auf Gewohnheiten von Windows Nutzern basieren. Andernfalls existieren auch viele Open Source Implementierungen für andere Plattformen, wie z.B. OpenLDAP, Fedora Directory Server, OpenDS, Apache Directory Server etc. (vgl. [WikiLDAP]). Bei LDAP ist eine Verwendung von SSL empfehlenswert, auch wenn der Server womöglich zu demselben Intranet gehört. Wie in [RFC4513] empfohlen sollten Passwörter nie direkt gesendet, sondern über eine Hashfunktion verglichen werden.
Für den automatisierten Abruf steht also eine Schnittstelle über das LDAP Protokoll zur Verfügung. Allerdings muss diese noch nutzbar gemacht werden, wozu das JNDI (*Java Naming and Directory Interface*) verwendet werden kann. Die Bibliothek ermöglicht dem Entwickler auf beliebige Namens und Verzeichnisdienste zuzugreifen. Seit Version 1.4 ist JNDI nativ in der Java2 Standard Edition verfügbar.

4.7.2 Protokollierung

Die Daten Anforderungen der Protokollierung wurden in Abschnitt 2.5 analysiert. Benutzerkennung, Datum, Uhrzeit und Ordnungsmerkmale der abgerufenen Datensätze müssen protokolliert werden. Bis auf das Ordnungsmerkmal sind sämtliche Informationen verfügbar und können gespeichert werden. Die Datenerzeugung allein kennt nur die Ordnungsmerkmale. Deswegen sind die Informationen durch die Daten

[18] Im Gegensatz zur strengen Implementierung des Verzeichnisdienstes X.500, hat die LDAP-Implementierung einen kleineren Funktions- und Kontrollumfang. Jedoch ist LDAP deutlich flexibler und hat sich daher auch im Internet durchgesetzt (vgl. [WikiLDAP]).

erzeugung einzutragen (siehe Abbildung 4 8). Die Referenz der Anfrage muss der Datenerzeugung übergeben werden. Es ist sinnvoll die Protokollierung an die jeweilige Anfrage zu knüpfen. Deswegen werden die Protokollierungsdaten in die gleiche Tabelle der Datenbank geschrieben bzw. dem gleichen Datensatz zugeordnet. Zusätzlich wird noch die IP Adresse der abrufenden Benutzer gespeichert. In §2 (2) der LiKatAVO M V (siehe [LiKatAVO]) ist festgelegt, dass alle Protokollierungen nach Ablauf von sechs Monaten zu löschen sind. Ein solcher Algorithmus wird für den Prototyp zunächst nicht implementiert. Eine Möglichkeit wäre die Prüfung auf veraltete Protokolleinträge bei einer Anfrage. Nachhaltiger ist die tägliche Prüfung des Datenbestandes auf alte Anfragen. Dieses Werkzeug muss unabhängig von anderen Dienstprogrammen operieren. Die Festlegungen der Verordnung müssen in jedem Fall eingehalten werden.

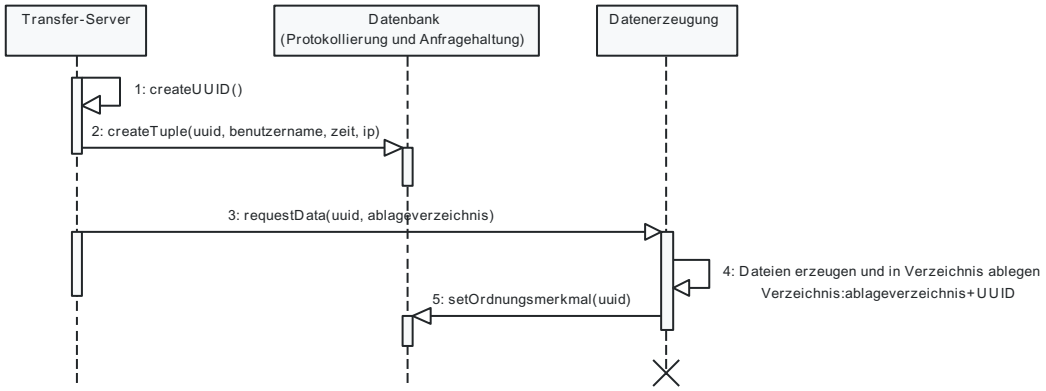


Abbildung 4-8 Datenerzeugung und Protokollierung

4.7.3 Datenerzeugung

Die Datenerzeugung wird über eine Schnittstelle angesprochen, die den Benutzernamen, der zuvor bei der Anmeldung übermittelt wurde, übergibt. Zudem wird die UUID der Anfrage übergeben, damit die Datenerzeugung einen Verweis zur Datenablage hat (Ablageordner+“\“+UUID). Nach der individuellen Datenerzeugung muss das Ordnungsmerkmal in die Datenbank bzw. in den Datensatz eingefügt werden. Wenn nun die abrufende Stelle eine Datenholungsanfrage sendet, können die Daten übertragen werden. Erkennbar ist dies an einem Eintrag in der Datenbank in dem Feld mit

dem Ablageverzeichnis. Liegt dort kein Eintrag vor, so wurden noch keine Daten er zeugt.

4.7.4 Datenbank

Abbildung 4 9 zeigt den Entwurf der graphischen Oberfläche für die Datenbankkompo nente. Ihre Aufgabe ist vorrangig die konsistente Speicherung der Protokollierungs und Anfragereferenzdaten (siehe Abschnitt 4.4.2). Nutzerdaten der LDAP Implementie rung können ggf. auch in der gleichen Datenbank abgelegt werden. Hierfür wird eine relationale Datenbank benötigt, die mindestens den SQL[19] Standard erfüllt. Durch die se Abfragesprache ist keine individuelle Anpassung des Programms nötig, da jede be deutende Datenbank SQL Abfragen nutzt. Durch JDBC sollten die wichtigsten Da tenbanken bedient werden können. Eine vollständige Übersicht aller Treiber ist unter [SunDrv] verfügbar. Die beste Lösung ist dabei die Nutzung von JDBC über Typ 4 (siehe Abschnitt 3.5.4). Sämtliche Treiber sind bei der Veröffentlichung mitzuliefern. Mitunter sollte berücksichtigt werden, dass auch die Datenerzeugung datenbankbasiert arbeiten könnte. Dabei liegen sämtliche Basisdaten in einer Spatial Datenbank[20] (engl. *räumli che Datenbank*) vor und werden durch ein Export Werkzeug in Dateien der einheitli chen Dateiformate konvertiert. Eine andere Bezeichnung hierfür ist auch objekt relationale Datenbank. Natürlich können auch Protokollierungs und Anfragedaten in dieser Datenbank organisiert werden. Die gesamte Datenhaltung würde dann auf eine Datenbank vereinigt werden. Dies würde den Administrationsaufwand geringer halten als bei der Wartung von mehreren Datenbanken. Über die graphische Oberfläche kann der Benutzer die Verbindungseinstellungen zur Datenbank festlegen (siehe Abbildung 4 9). Dazu muss erst der passende Treiber gewählt und folgend der Hostname der Da tenbank sowie Benutzername und Passwort eingetragen werden.

Welche Treiber im Einzelnen mitgeliefert werden sollen, muss mit den Anwendern des Programms abgestimmt werden. Auch wenn der Aufwand den Treiber in die Software zu implementieren relativ gering ist, muss die Software auch in Zukunft weiter gepflegt werden. Dazu gehören ebenso die Aktualisierung und der Test neuer Treiber

[19] SQL (engl. *Structured Query Language*) ist eine Datenbanksprache zur Definition, Abfrage und Manipulation von Daten in relationalen Datenbanken.

[20] Datenbank zur Speicherung von räumlichen Daten.

Versionen. Der Prototyp wird lediglich mit einer Access Datenbank ausgestattet. Die Datenbankauswahl ist daher erst später notwendig.

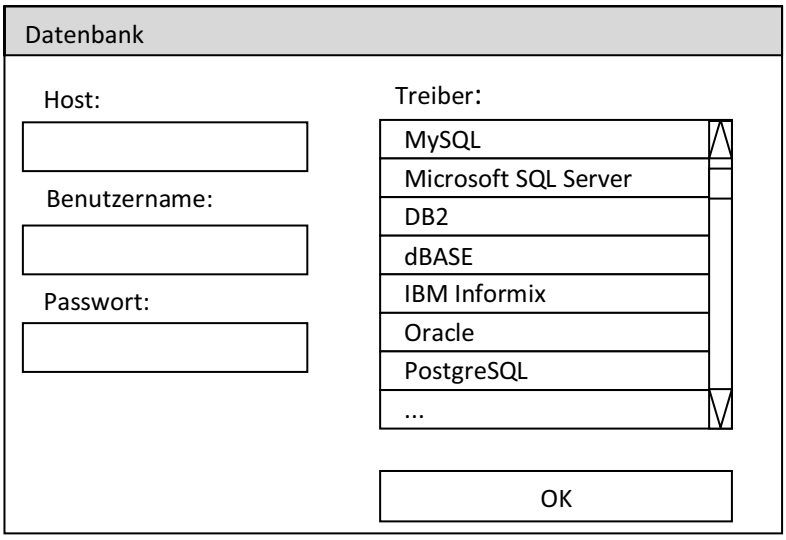


Abbildung 4-9 Entwurf der GUI zur Datenbankauswahl

In der Datenbank müssen die erforderlichen Tabellen mit Spalten und ihren Eigen schaften vor der Inbetriebnahme erstellt werden. Zuvor erstellte SQL Dateien können zum Erstellen der Tabellen importiert werden und den Vorgang erleichtern. Für die Da tenbankzugriffe sollte eine separate Klasse die Zugriffe realisieren, da an verschiede nen Stellen in der Anwendung Daten ausgelesen oder manipuliert werden sollen (z.B. Protokollierung).

4.7.5 Liegenschaftsdatenverwaltungssoftware

Bei der Liegenschaftsverwaltungssoftware handelt es sich um ein Geoinformationssys tem (GIS), welches speziell für die Arbeit im Bereich Liegenschaftskataster angepasst ist. Ein Produkt, das in Mecklenburg Vorpommern eine weite Verbreitung hat, ist die Software

GISAL der Firma BTFietz. Einrichtungen nutzen diese Programme meist für die Kom munalverwaltung. Da geeignete Werkzeuge für die Nutzung der übergebenen Daten formate in der Liegenschaftssoftware vorhanden sind, ist ein Import möglich. Die Softwarehersteller müssen jedoch eine Schnittstelle für den automatischen Import von Dateien bereitstellen. Im Abrufprogramm kann demnach eine ausführbare Datei und für alle Liegenschaftsverwaltungsprodukte systemneutral ausgewählt werden, die für

das Programm als Schnittstelle fungiert. Um der Datei den Verweis der zu importie renden Dateien zu geben, wird der Pfad als Parameter übergeben. Die folgenden Schritte werden durch die Liegenschaftsverwaltungssoftware realisiert. Ein einfacher Entwurf der graphischen Oberfläche zum Konfigurieren des Imports wird in Abbildung 4 10 gezeigt.

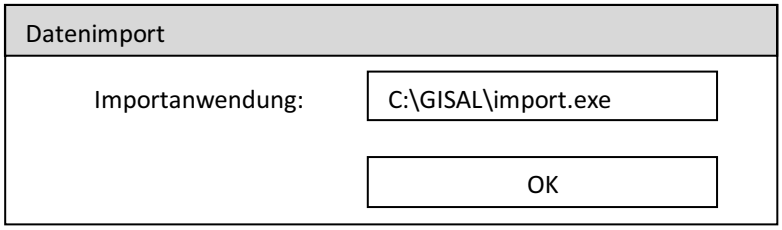


Abbildung 4-10 GUI zum Konfigurieren des Imports

Zur Laufzeit wird nach dem erfolgreichen Herunterladen der Pfad übergeben (siehe Tabelle 4 1). Ob der Datenimport erfolgreich abgeschlossen wurde, liegt in der Zustän digkeit der Liegenschaftsverwaltungssoftware. Wurde die Importanwendung allerdings beim Ausführen des Befehls nicht gefunden, so muss eine Ausnahme ausgelöst wer den.

C:\GISAL\import.exe „C:\Abruf\Eingang\<Referenzschlüssel>“

Tabelle 4-1 Importaufruf

Anzumerken ist, dass die Einstellung für den automatischen Import optional anzuge ben ist. Daher ist es nicht zwingend nötig im Fenster Datenimport (siehe Abbildung 4 10) einen Verweis einzutragen. Der Anwender muss den Import der Dateien dann je doch manuell durchführen. Da *GISAL* diesen Aufruf noch nicht unterstützt, ist vorerst keine Implementierung für den Prototyp notwendig.

4.8 Verbindungsmanager

Abbildung 4 4 zeigt den Verbindungsmanger als Komponente der Präsentationsschicht. Sämtliche verbindungsrelevante Einstellungen werden über die graphische Oberfläche konfiguriert. Dazu gehören auch Zeiteinstellungen. Gestellte Anfragen werden pro tokolliert und ausgewertet. Ein Abruf, der erfolgreich durchgeführt wurde, muss dabei

markiert werden. Einstellungen werden grundsätzlich auf die zugehörige Verbindung bezogen. Jede Verbindung hat also eine eigene Zeiteinstellung, Anfrageprotokollierung sowie Verbindungsparameter (siehe Abbildung 4 11).

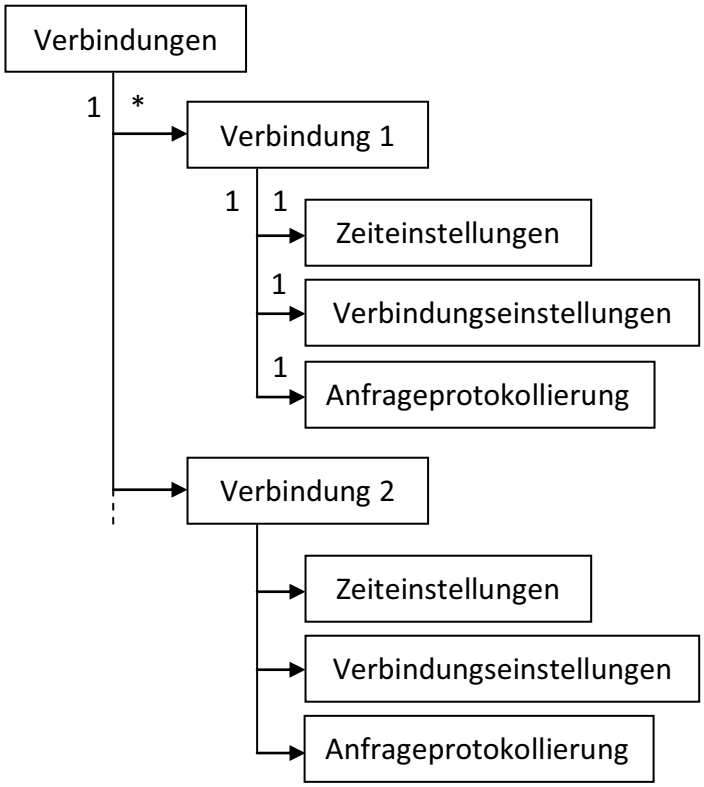


Abbildung 4-11 Baumstruktur der Verbindungen zu datenhaltenden Stellen

4.9 Zeitplanung

Voraussetzung für eine Zeitplanung (engl. *scheduling*) ist der permanente Betrieb eines Prozesses, der ggf. einen Abruf initialisiert. Dazu wird ein Dienst im Betriebssystem re gistriert. Der Unterschied zu einer Autostart Anwendung ist, dass sich ein Benutzer nicht im Betriebssystem anmelden muss. Dadurch wird das Programm ausgeführt, so bald der Rechner hochgefahren ist. Zwischen den Abrufen darf der Dienst nur mini male Ressourcen beanspruchen. Ein permanenter Betrieb ist so softwaretechnisch ge währleistet.

In der datenhaltenden Stelle ist die Laufenthaltung der Daten an ein bestimmtes Inter vall gebunden. Daher sollte die abrufende Stelle auch parallel zu diesem Intervall ar beiten. Unnötige Anfragen werden so vermieden. Der Anwender nutzt die graphische

Oberfläche zum Erstellen des Zeitplans. Dazu sind die wichtigsten Zeiteinstellungen festzulegen:

Zeitpunkt

- Uhrzeit,
- Wochentage,

Intervall

- Tage,
  (z.B. Abruf alle 14 Tage um 22:30 Uhr)
- Wochen und
  (z.B. Abruf alle 2 Wochen um 22:30 Uhr)
  (z.B. Abruf alle 2 Wochen, sonntags um 22:30 Uhr)
- Monate.
  (z.B. Abruf alle 3 Monate, jeden 2. Sonntag im Monat um 22:30 Uhr)

Ein Sonderfall ist der Abruf zu einem bestimmten Stichtag im Monat. Dabei ist es mög lich z.B. am. 1. bzw. 15. jeden Monats den Abruf ausführen zu lassen.

4.10 Konfigurationsdatei

In den vorherigen Abschnitten wird beschrieben welche verbindungsspezifischen Da ten beim Client gespeichert werden sollen. Abbildung 4 11 zeigt eine grobe, aber the matische Einteilung der Baumstruktur. Gewissermaßen handelt es sich um Elemente, die von ihrem Eltern Element erben. Die Wurzel ist dabei *„Verbindungen“.* Sie kann be liebig viele Verbindungen besitzen. Jedoch hat jede Instanz festgelegte Eigenschaften. Dazu gehören die Parameter zu Zeiteinstellungen, Verbindungseinstellungen und Anf rageregistrierungen. Solange noch keine Anfrage versendet wurde, hat die Anfragere gistrierung keine weiteren Kind Elemente.
Um diese Hierarchie abbilden zu können eignet sich eine XML Datei (*Extensible Markup Language*). Die Parameter bekommen so jeweils ein eigenes Element und werden ihren zugehörigen Eltern Elementen zugeordnet. Jedes XML Element kann da bei Attribute und Werte besitzen. Der Konfigurationsmanager stellt Methoden zum

Abfragen und Manipulieren der festgelegten Elemente bereit. Dazu wird ein Parser verwendet der den Zugriff auf die XML Datei organisiert.

4.11 Policy-Datei

Durch die Policy Datei werden zentrale Sicherheitsrichtlinien durchgesetzt. Demzu folge kann eine beschränkte Basissicherheit ausgemacht werden. Implementiert wird diese durch den Java SecurityMananger (siehe Abschnitt 3.5.3). Die Beschränkungen für Server und Client werden in den nächsten beiden Abschnitten festgelegt.

4.11.1 Server

Der Dateizugriff ist in dem Ablageordner für die erzeugten Daten nur lesend möglich. Weiterhin werden nur Socketverbindungen zu den RMI Ports 1089/99 zugelassen. Die se Regelung kann dann gelockert werden, wenn die Verbindung über diese Ports auf grund von Firewalleinschränkungen nicht möglich ist.

4.11.2 Client

Der Client hat schreibenden Zugriff im eigenen Programmordner und die dazugehöri gen Unterverzeichnisse, sowie im Ablageordner für die heruntergeladenen Dateien. Ebenso werden nur Socketverbindungen über die festgelegten Ports zugelassen.

4.12 Verfügbarkeit des Dienstes

Um die Verfügbarkeit des Dienstes zu gewährleisten, müssen zunächst die Hardware anforderungen in Abschnitt 2.10 eingehalten werden. Der Prototyp kann dabei nur die softwarespezifischen Anforderungen erfüllen. Insbesondere ist der permanente Be trieb sowohl für Client, als auch für den Server durch die Registrierung als Dienst er forderlich. Für Wartungsarbeiten am System kann der Server kurzweilig außer Betrieb genommen werden. Anwender müssen jedoch vorab rechtzeitig informiert werden (z.B. E Mail).

4.13 Objektabfragen

In Abschnitt 2.7 wurde die Anforderung für die Abfrage von einzelnen Objekten bereits erwähnt. Der Anwender soll dabei tagaktuelle Daten von einem bestimmten Objekt abfragen können. Hierzu wird in der Liegenschaftsverwaltungssoftware auf das entsprechende Element geklickt und die Aktion ausgeführt. Daraufhin wird der Abruf Client gestartet. Dabei wird eine Anwendung benötigt, die ohne graphische Oberfläche arbeitet. Hierzu wird eine eigene Startdatei für die Objektanfrage erstellt. Der Objektschlüssel wird der Startdatei als Parameter übergeben. Standardmäßig wird jetzt die Verbindung zum Transfer Server hergestellt, welcher die Anfrage an die Datenerzeugung weiterleitet. Es erfolgt die Erstellung der Datei im einheitlichen Datenformat. Nachdem die Datei wieder auf den Client transferiert wurde, kann der automatische Import aufgerufen werden. Jetzt ist es Aufgabe der Liegenschaftsverwaltungssoftware die Datei zu importieren und die Ergebnisse dem Anwender zu präsentieren.

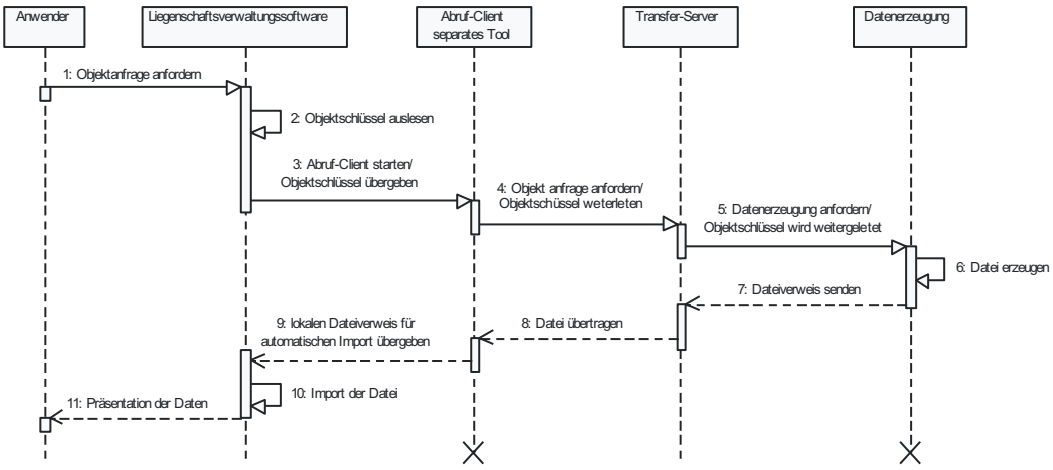


Abbildung 4-12 Abruf von tagaktuellen Objektinformationen

5 Implementierung

Die bisherige Entwicklung basiert auf einem Prototyp. Dabei wurde ein evolutionärer Ansatz (engl. *Evolutionary Prototyping*[21]) gewählt, der bis zur endgültigen Version weiterentwickelt werden soll. Daher sind Qualitätsstandards zu berücksichtigen und eine umfangreiche Dokumentation anzulegen. Probleme, die bei der Analyse und im Entwurf nicht deutlich werden, können hier aufgedeckt werden. Außerdem kann durch den Prototyp der eigentliche Umfang sowie die Leistungsfähigkeit der Implementierung abgeschätzt werden. Mit dem Prototyp kann dem Anwender schon ein erstes Programm praktisch vorgeführt werden. Anforderungen können so besser abgestimmt werden. Zudem kann auf die Bedürfnisse der Anwender besser eingegangen werden. Die Plattform beschränkt sich derzeit noch auf Windows Betriebssysteme. Generell ist eine Portierung aber auf andere Plattformen möglich.

Klassendiagramm

Zusammengehörige Funktionen wurden weitestgehend in separate Klassen gegliedert, welche wiederum in Pakete eingeordnet sind. Die Klassennamen versuchen deren Funktion kurz zu verdeutlichen. Die Klasse *MultiStartBlocker* bietet beispielsweise Funktionen, um das mehrfache Starten von Anwendungen zu verhindern. Für den RMI Aufruf sind besonders die beiden Schnittstellen *RemoteLogin* (Anmeldung am Server) und *Abruf* (zugriffsgeschütze Funktionen des Servers) wichtig. Abbildung 5 1 zeigt sämtliche Klassen. Abhängigkeiten, Methoden und Variablen wurden zur Wahrung der Übersichtlichkeit nicht dargestellt.

[21] Im Gegensatz zum „Throw Away“ Prototyping (dt. Wegwerf-Prototyp), wird der Prototyp bei „Evolutionary Prototyping“ über mehrere Versionen bis hin zum endgültigen System weiterentwickelt. Weiterführende Informationen siehe [Sommerville, 2001], Seite 181 ff..

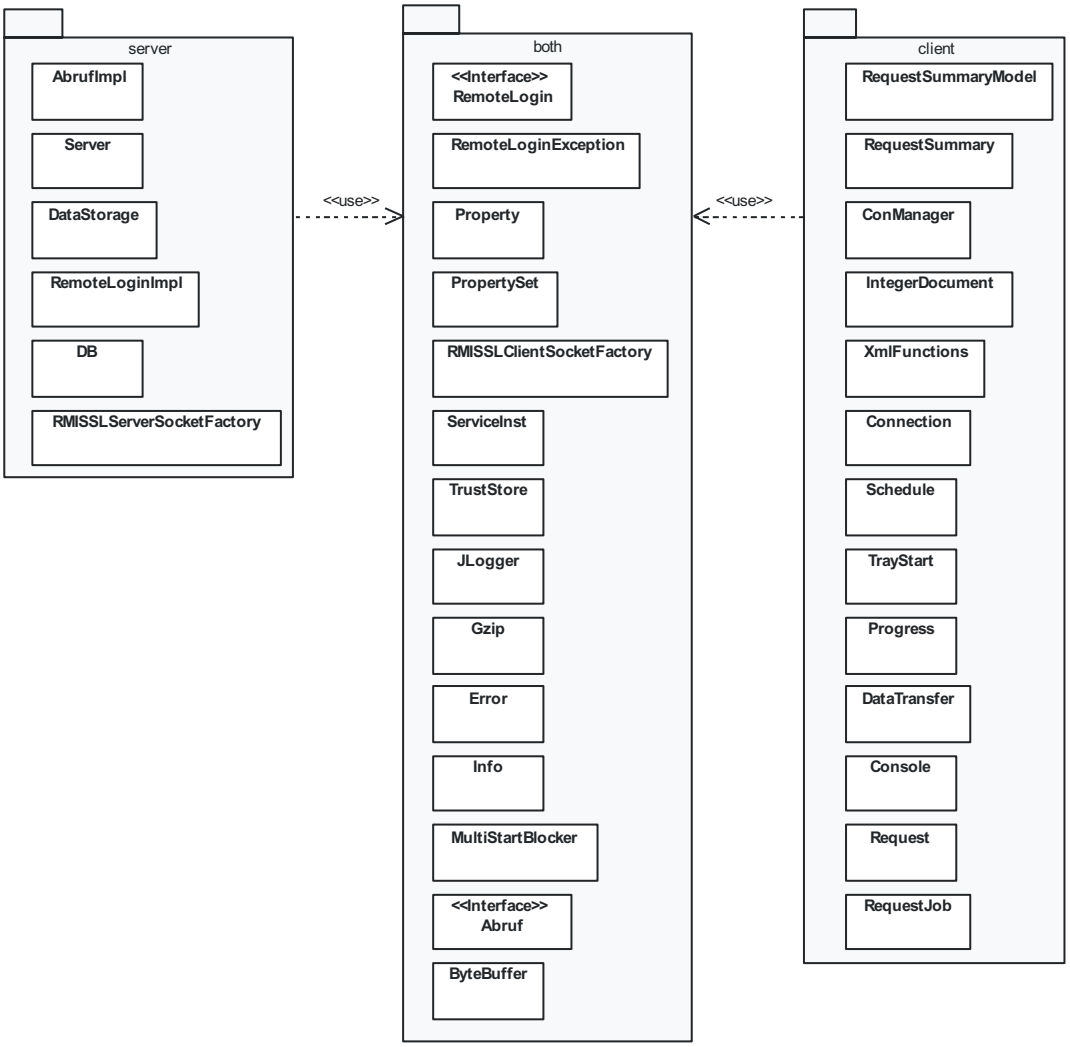


Abbildung 5-1 Klassendiagramm der Abruf-Software (vereinfacht)

5.1 Sicherung der Kommunikation

Die Sicherung der Kommunikation erfolgt in mehreren Schritten. Zunächst muss ein geeignetes Protokoll für die Sicherung der Kommunikation gewählt werden. SSL ist ein hybrides Verfahren und vereinigt die Vorteile von symmetrischen Verfahren bei der Datenübertragung und asymmetrischen Verfahren beim Austausch des Schlüssels. Für den Prototyp werden daher Zertifikate benötigt. Ein Anspruch auf höchste Vertraulich keit des ausgestellten Zertifikats durch die CA (engl. *Certification Authorities*) ist dafür zunächst nicht erforderlich, da es lediglich für Testzwecke zum Einsatz kommt.

5.1.1 Kryptografieverfahren

Die verfügbaren Kryptografieverfahren sind durch das Zertifikat festgelegt. Der Client wählt in einem Dialog mit dem Server das bevorzugte Verfahren automatisch aus. (hier: SSL_RSA_WITH_RC4_128_MD5).

5.1.2 Zertifikate

Im Prototyp wurde ein Beispielzertifikat (nach X.509v3[22] Standard) der Firma Veri Sign[23] verwendet. Dies muss für den öffentlichen Einsatz des Programms ausgetauscht werden. Das Problem des Schlüsselaustausches besteht nicht, da der Schlüssel schon installiert wird. In Java wird zudem ein Passwort für das Zertifikat verlangt. Wiederum ist im Prototyp nur ein unzureichendes Passwort verwendet worden. Dies ist vor Veröf fentlichung der Software zu ändern, um die Sicherheit des Systems zu erhöhen.

5.1.3 Datenhaltung von sicherheitsrelevanten Dateien

Leider sind sowohl sämtliche Zertifikate als auch die Policy Datei durch jede Person, die Zugang zum Dateisystem des Rechners hat, modifizierbar. Jedes digitale Zertifikat kann nur dann als sicher eingestuft werden, wenn es vom Zugriff Unbefugter geschützt wird. Eine Patentlösung für dieses Problem existiert derzeit noch nicht. Allerdings ist es möglich diese sicherheitsrelevanten Dateien in die Jar Datei zu importieren. Diese Da ten sind aber für den Anwender nicht direkt sichtbar. Über ein Archivierungsprogramm ist der Zugriff auf diese Dateien natürlich weiterhin möglich.

Die Klasse *TrustStore* hat die Aufgabe, zur Laufzeit die sicherheitsrelevanten Dateien aus dem Jar Archiv zu entpacken und als temporäre Dateien verfügbar zu machen. Nach Beendigung des Programms werden Zertifikate und die Policy Datei wieder ge löscht. Eine weitere Klasse *PropertySet* übernimmt das Laden sämtlicher Systemda teien zur Laufzeit. Dadurch kann die Sicherheit zusätzlich erhöht werden.

[22] X.509 ist eine plattformunabhängige Notation, welche die *Abstract Syntax Notation One* (ASN.1) verwendet. Aktuelle liegt X.509 in der Version 3 vor.

[23] Bei VeriSign handelt es sich um ein Unternehmen, das u.a. als Zertifizierungsstelle für digitale Zertifikate tätig ist.

5.1.4 Secure Sockets Layer

Bevor es zur Anwendung von SSL kommen kann, müssen die Dateien für den Trust store geladen werden. Die Klasse *PropertySet* setzt dazu die Umgebungsvariablen in Java. Alternativ können diese auch beim Start der Java Anwendung über einen Para meter übergeben werden. Neben dem Truststore Passwort wird auch die Policy Datei verknüpft. Die Speicherung des Passworts im Quellcode birgt dabei das Risiko, dass Unbefugte das Passwort auslesen (auch nach der Kompilierung). Es wird dabei voraus gesetzt, dass nur autorisierte Personen Zugang zum Rechner haben. Alternativ könnte das Passwort auch zur Laufzeit eingegeben werden. Ein automatischer Abruf wäre dann ohne präsenten Anwender nicht mehr möglich.

Für die Erzeugung der RMI Registry werden benutzerdefinierte Socket Factories ge nutzt (siehe Tabelle 5 1). Der Server nutzt die Klassen *RMISSLClientSocketFactory* und *RMISSLServerSocketFactory* zum Schutz der Kommunikation. Die vorher festgelegten Umgebungsvariablen für den Truststore werden dabei benötigt, um die Zertifikate ein zulesen. Die Authentisierung der Kommunikationspartner wird durch den Java Key manager realisiert. Dieser ist ebenfalls in der Klasse *RMISSLServerSocketFactory* imple mentiert. Für die Übermittlung der Daten werden die RMI Standardport genutzt.

```
// Security Manager installieren
if (System.getSecurityManager() == null)
                System.setSecurityManager(new RMISecurityManager());

System.out.println("Erzeuge RMI Registry");
Registry registry = LocateRegistry.createRegistry(Registry.REGISTRY_PORT,
        new RMISSLClientSocketFactory(), new RMISSLServerSocketFactory());
// LoginModul
RemoteLogin remote = new RemoteLoginImpl();
serverThread = new Thread();
serverThread.start();

System.out.println("OK");
registry.bind("Abruf", remote);
System.out.println("Server gestartet ...");
```

Tabelle 5-1 Server-Implementierung mit SSL

Für die Durchsetzung der Policy Richtlinien wurde der Verweis für die entsprechende Datei bereits gesetzt. Beim Start des SecurityManagers wird diese automatisch initiali

siert (siehe Tabelle 5 1). Erst jetzt wird die Remote Schnittstelle bereitgestellt, damit sich der Client am Server sicher anmelden (*RemoteLogin* Klasse) kann.

5.2 Serververbindung

Wie in Abschnitt 3.3.1 beschrieben ist, registriert der Server ein Remote Objekt bei der RMI Registry. Über den Namensdienst kann der Client eine Verbindung aufbauen und das Objekt nutzen. Damit der Anmeldevorgang realisiert wird, stellt der Server nur ei ne Methode *„login“* über die Schnittstelle *RemoteLogin* bereit. Dies ist der einzige Weg über den der Client Kontakt zum Server aufnehmen kann. Als Übergabewerte werden hier Benutzername und Passwort gesendet. Serverseitig wird nun die Autorisierung geprüft. Im Prototyp ist die Schnittstelle zur Benutzerverwaltung noch nicht implemen tiert. Daher wurden zwei Testnutzer angelegt, die unterschiedliche Identitäten reprä sentieren und die die Benutzerverwaltung simulieren. Jeder Anwender hat demzufolge ein separates Kennwort für den Zugang zum Server. Bei einer fehlerhaften Eingabe von Benutzername oder Passwort wird eine eigens implementierte Ausnahme vom Typ *RemoteLoginException* ausgelöst. Wurde die Autorisierung bestätigt, so wird das Ob jekt zurückgeben, welches die zugriffsgeschützten Methoden des Servers enthält. Die se sind durch die Schnittstelle *Abruf* implementiert. Abbildung 5 2 beschreibt den Ab lauf bei einem erfolgreichen Anmeldevorgang am Server.

Ausreichend Sicherheit kann nur erzielt werden, wenn eine Passwortvergabe gewis senhaft erfolgt. Dabei sollten grundlegende Aspekte berücksichtigt werden.

- Lange Passwörter nutzen,
- Passwort darf nicht dem Benutzernamen entsprechen,
- Zufällige Zeichen nutzen (d.h. keine gewohnten Wörter),
- Zeichenketten nicht wiederholen und
- Sonderzeichen und Groß /Kleinschreibung nutzen.

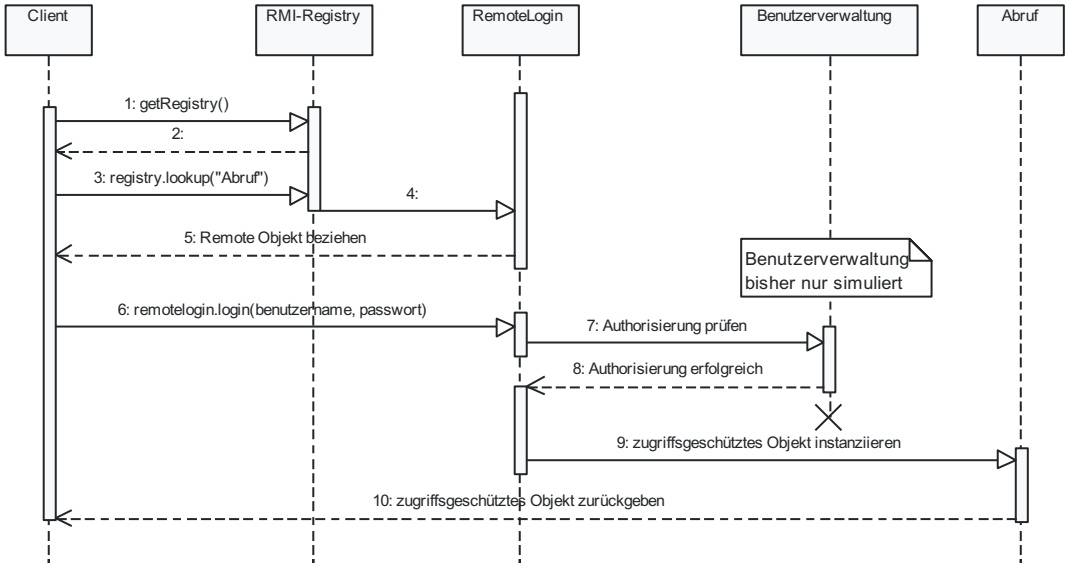


Abbildung 5-2 Sequenzdiagramm für einen erfolgreichen Anmeldevorgang

Unter [Cnlab] ist es möglich sein eigenes Passwort überprüfen zu lassen. Es gibt drei Kategorien (weak, usable, strong) in welche das Passwort eingeordnet werden kann. Außerdem wird die ungefähre Rechenzeit angegeben die ein Angreifer benötigt, um das Passwort zu errechnen. Zusätzlich werden dem Anwender Funktionalitäten bereit gestellt, die eine syntaktische Analyse des Passworts ermöglichen.

5.3 Anfragen und Protokollierung

In Abschnitt 4.4 wurde bereits das Problem der Anfrageorganisation angesprochen. Dabei speichert der Server eine Referenz für jede Anfrage. Der Client legt eine redun dante Referenz in eine XML Datei ab. Dadurch braucht der Client nicht permanent eine Verbindung zum Server aufbauen.

Der Prototyp legt die Anfragen in eine Access Datenbank ab. Vorteil ist dabei die Por tabilität, da es sich um eine Desktop Datenbankanwendung handelt. Sämtliche Daten liegen dabei in einer einzigen Datei vor. Denkbar ist natürlich auch eine Umsetzung mit mächtigeren Implementierungen, die jedoch mehr Know How im Administrationsbe reich erfordern. Der Installationsaufwand ist zudem relativ hoch. Eine Nutzung des mdb Formats auf Linux Betriebssystemen ist möglich.

Die Klasse *DB* organisiert dabei sämtliche Zugriffe auf die Datenbank. Um später die Datenbank ersetzen zu können, muss in dieser Klasse die Datenbankauswahl (siehe

Abschnitt 4.7.4) ausgelesen werden. Außerhalb der Klasse ist nicht sichtbar mit welcher Datenbank eigentlich gearbeitet wird bzw. wo diese liegt (siehe Abbildung 5 3).

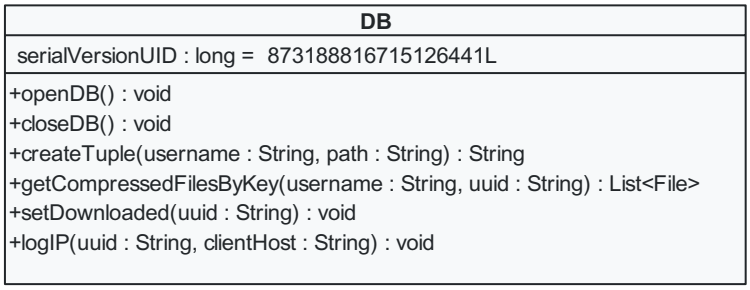


Abbildung 5-3 Klassendiagramm der Datenbankzugriffverwaltung

5.4 Graphische Oberfläche

Für die Interaktion mit den weiteren Komponenten ist die graphische Oberfläche in der Symbolleiste des Betriebssystems verankert (Klasse *TrayStart*). Neben dem Beenden des Programms stehen zwei weitere Funktionen zur Verfügung:

- Verbindungsmanager und
- Konsole.

5.4.1 Verbindungsmanager

Der Verbindungsmanager (Klasse *ConManager*) ist die zentrale Komponente der graphischen Oberfläche. Der Anwender kann dort sämtliche Konfigurationen vornehmen. Abbildung 5 4 zeigt auf der linken Seite einen Baum, der mehrere Verbindungen enthält. Es können beliebig viele Blattelemente an den Baum angefügt werden. Der Baum muss jedoch mindestens eine Verbindung enthalten. Demzufolge ist es nicht möglich das letzte Element zu löschen. Mit einem Klick der rechten Maustaste auf den Baum öffnet sich ein Popup Menü, welches außerdem die Möglichkeit bietet Verbindungen zu löschen oder den Namen der Verbindung zu ändern. Allerdings darf jeder Name nur einmalig vorkommen. Eine doppelte Namensvergabe wird durch das Programm blockiert.

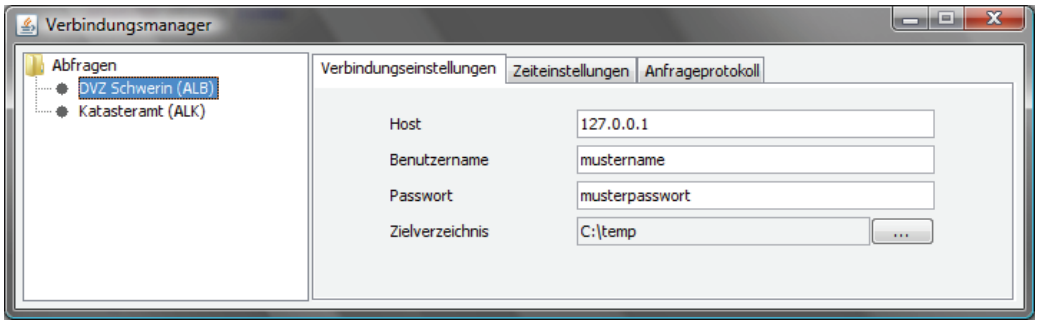


Abbildung 5-4 Graphische Oberfläche des Verbindungsmanagers

Verbindungseinstellungen

Wie in Abschnitt 4.8 beschrieben wird, gibt es zu jeder Verbindung drei Kategorien: verbindungsspezifische Einstellungen, Zeiteinstellungen und das Anfrageprotokoll. Die Verbindungseinstellungen sind die Basisdaten jeder Verbindung. Hier werden Hostna me zum Transfer Server, Benutzername, Passwort und das Zielverzeichnis festgelegt. Dieses kann mit Hilfe eines *„file choosers“*[24] gewählt werden. Fehlerhafte Eingaben bei der Verzeichniswahl sollen dadurch vermieden werden. Außerdem kann der Anwender neue Verzeichnisse erstellen oder bestehende Verzeichnisse manipulieren.

Zeiteinstellungen

Der Anwender kann zwischen drei Intervallen wählen – Tage, Wochen, Monate. Neben der Uhrzeit wird dann die Länge des Intervalls eingestellt. Dabei können lediglich ganze Zahlen mit sinnvollen Werten eingetragen werden (Klasse *IntegerDocument*). Das Intervall für Tage kann dabei zwischen einem Tag bis hin zu 99 Tagen gewählt werden. Bei der wochenweisen Abfrage muss der Anwender noch zusätzlich den Wochentag wählen (siehe Abbildung 5 5).

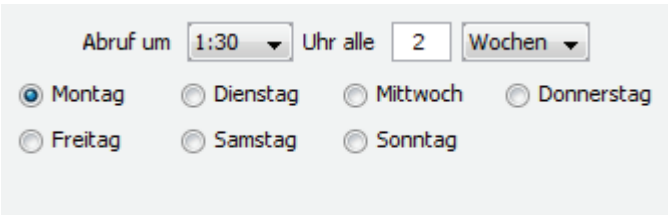


Abbildung 5-5 Zeiteinstellungen im Wochenintervall

[24] Ein „file chooser“ ermöglicht das Navigieren im Dateisystem mittels graphischer Oberfläche. Dabei können Dateien oder Verzeichnisse gewählt werden, die jedoch von Parametern abhängig sind.

Bei einem Intervall nach Monaten können ebenfalls wochentags spezifische Einstellun gen erfolgen. Optional werden die weiteren Parameter aktiviert. Da jeder Monat meh rere Wochen besitzt kann dort auch eine Festlegung erfolgen. Dadurch ist es bei spielsweise möglich jeden 1. Sonntag im Monat Abrufe auszuführen (siehe Abbildung 5 6). Die interne Umsetzung wird in Abschnitt 5.9 erläutert. Der Startzeitpunkt des Ab rufs ist immer der nächstmögliche Termin.

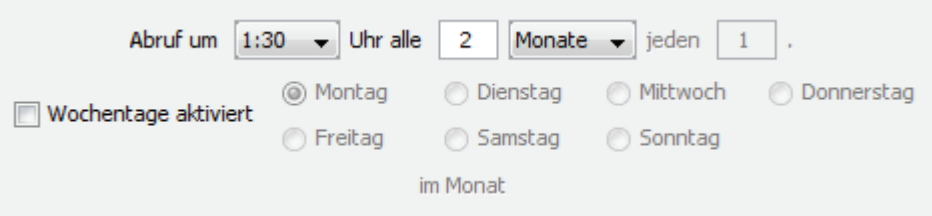


Abbildung 5-6 Zeiteinstellungen im Monatsintervall

5.4.2 Anfrageprotokoll

Im Reiter Anfrageprotokoll werden sämtliche erfolgreich gestellte Anfragen clientseitig gespeichert. Die Datenhaltung wird durch die Konfigurationsdatei (siehe Abschnitt 4.10) geregelt. Außerdem werden manuelle Anfragen von dort aus gestartet. Dazu bie tet ein Popup Menü verschiedene Funktionen. Bevor die Daten heruntergeladen wer den können, muss der datenhaltenden Stelle eine Datenanfrage gesendet werden (*„neue Daten anfragen“*). Diese Anfrage wird beim Server registriert. Erst danach kann eine Datenholungsanfrage gesendet werden *(„herunterladen“*).

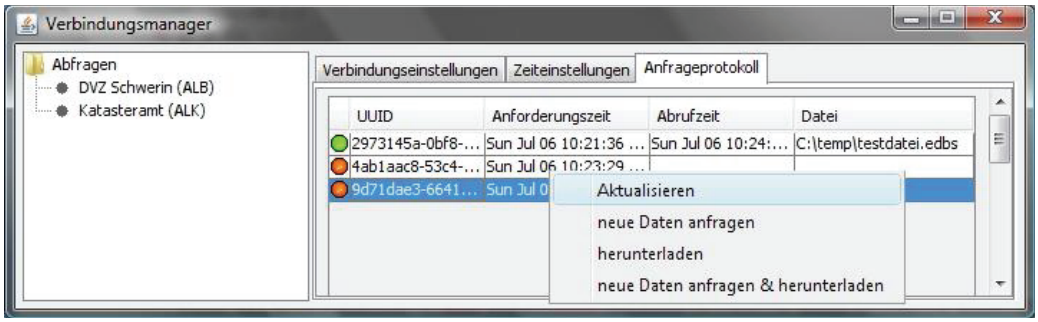


Abbildung 5-7 Protokollierung und Status der Anfragen

Da im Prototyp die Daten automatisch nach der Datenanfrage bereitgestellt werden, existiert der Menüpunkt „neue Daten anfragen & herunterladen“. Daten und Datenho lungsanfrage werden dabei sequentiell ausgeführt. Für die Veröffentlichung der Soft

ware sind daher unter Umständen Anpassungen notwendig, da der Prozess der Daten erzeugung noch ungeklärt ist (siehe Abschnitt 2.9, *Datenanfrage und Daten holungsanfrage)*.

Abbildung 5 7 zeigt die graphische Oberfläche des Prototyps. Bereits heruntergeladene Liegenschaftsdaten werden grün gekennzeichnet. Noch ausstehende Abfragen haben hingegen eine rote Kennzeichnung. Zur Orientierung und Einordnung der Anfragen wird die Anforderungszeit genutzt. Für den Prototyp besteht außerdem die Anzeige der UUID. Dies sollte dem Anwender später verborgen bleiben. Abgerufene Daten sind neben der farblichen Markierung auch durch einen Eintrag in der Spalte *„Abrufzeit“* und *„Datei“* gekennzeichnet. Dadurch kann der Anwender die Anfrage zumindest zeit lich und dem Status nach (angefragt oder schon heruntergeladen) einordnen.
Momentan ist es noch nicht möglich veraltete Anfragen aus dem Protokoll zu entfer nen. Eine solche Funktion sollte daher noch implementiert werden. Offen ist zudem die Identifikation des Abrufs im Dateisystem. Dies kann durch die UUID erfolgen. Aller dings ist die Zeichenkette sehr kryptisch und schwer memorierbar. Selbst eine fortlau fende Nummer als Ordnername bringt nicht die gewünschte Übersicht, obwohl da durch zumindest der erste und letzte Abruf identifiziert wäre. Auch wenn sehr viele Abrufe erfolgt sind und heruntergeladene Dateien im Ablageordner liegen, sollte der Anwender komfortabel zu den Dateien navigieren können. Durch einen Kommandozei lenbefehl könnte eine Funktion bereitgestellt werden, die es erlaubt per Mausklick den Windows Explorer mit dem gewünschten Verzeichnis zu öffnen. Dem Anwender würde die Bedienung so erleichtert.

Nach dem Ausführen einer Anfrage wird automatisch die Konsole geöffnet. Informa tionen über den Status des Abrufs werden dem Anwender mitgeteilt. Dem Entwickler werden zusätzlich wichtige Informationen für das Debugging ausgegeben.

5.4.3 Konsole

Die Konsole (Klasse *Console*) zeigt dem Anwender Informationen über die Abrufe an. Dabei wird jede Meldung über den Verbindungsnamen zugeordnet. Der Anwender

kann somit jede Meldung einer Verbindung zuordnen. Datengrundlage der angezeig ten Daten ist die Log Datei, welche über das Logging Modul erzeugt wurde. Bevor Meldungen angezeigt werden, durchlaufen sie eine Aufbereitung. Zweck ist die Ver besserung der Benutzerfreundlichkeit. Fehler werden durch die Farbe Rot signalisiert. Daneben existiert eine grüne Markierung für erfolgreich abgeschlossene Abrufe. Wei terhin kommen dort drei Knöpfe vor, um den Zeitplanungsdienst zu starten, beenden oder nur neu zu starten. Zur Orientierung wird zu jedem Eintrag das Datum und die Uhrzeit angezeigt (siehe Abbildung 5 9). Ein Feld mit dem Namen *„Status“* informiert den Anwender, ob der Dienst momentan gestartet ist (grüne Markierung). Ansonsten wird das Feld rot markiert. Implementiert wurde diese Funktion über einfache Kom mandozeilenbefehle. Mit dem Befehl *„net start“* können sämtliche in Windows gestar tete Dienste angezeigt werden. In Kombination mit dem Befehl *„find“ über* den Pipe Operator[25](*„|“*) kann nach dem Namen des Dienstes für den automatisierten Abruf ge sucht werden. Der Prototyp trägt den Dienstnamen „Abruf Liegenschaftsdaten“. Wird diese Zeichenkette gefunden, so wird ein Wert größer 0 zurückgegeben (siehe Abbil dung 5 8).

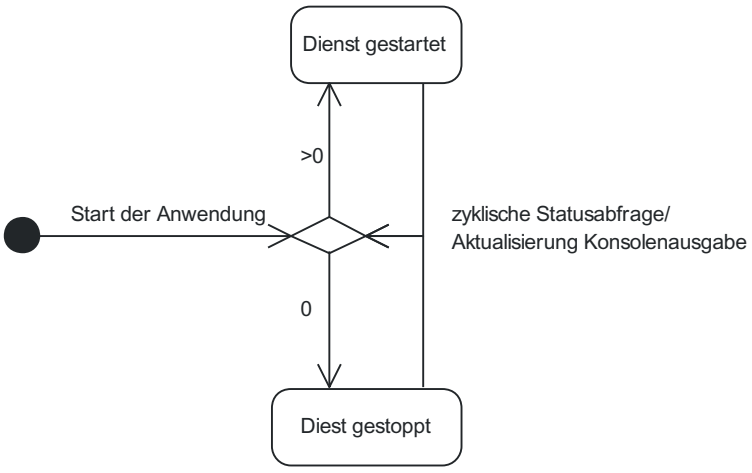


Abbildung 5-8 Zustandsdiagramm des Dienst-Status

Eine Implementierung für Linux ist derzeit nicht enthalten. Hierfür muss lediglich der Kommandozeilenbefehl angepasst werden.

[25] Der Pipe-Operator (*dt. Röhre*) erzeugt einen durch zwei Prozesse genutzten Speicherbereich. Der Pipe-Server erzeugt dabei die Pipe und der Client verbindet sich. Dabei schreibt ein Prozess Daten in den gemeinsam genutzten Speicherbereich und der andere Prozess liest diese Daten (vgl. [PIPE]).

Da neue Meldungen jederzeit hinzukommen können, muss eine automatische Aktualisierung des Konsolenfensters erfolgen. Zudem kann sich der Status des Dienstes ändern. In Java gibt es die Timer Funktion in der Bibliothek *java.util.Timer*. Es wird dazu ein neues Objekt vom Typ Timer instanziiert, das die Aufgabe (*TimerTask*) zugeordnet bekommt. Zusätzlich müssen noch der Startzeitpunkt und das Intervall der Wiederholungen festgelegt werden. In Tabelle 5 2 wird ein *TimerTask* erzeugt. Als Startzeitpunkt ist eine Millisekunde angegeben, was praktisch einem sofortigen Beginn entspricht. Das Wiederholungsintervall wurde auf vier Sekunden gesetzt. Ein eigener Thread organisiert die Aktualisierung.

```
Timer timer = new Timer();
timer.schedule( new Task(), 1, 4000 );
```

Tabelle 5-2 Timer-Funktion in der Java-Bibliothek

Das Intervall ist ein Kompromiss aus Ressourcenschonung und Benutzerfreundlichkeit. Änderungen müssen dem Anwender relativ zügig in der Konsole angezeigt werden. Trotzdem sollen andere Prozesse nicht signifikant eingeschränkt werden.

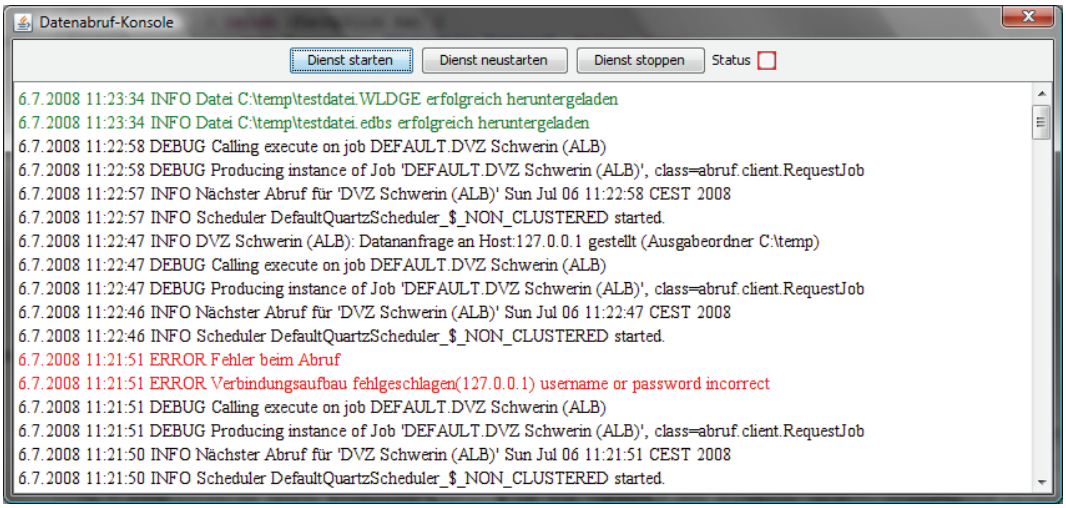


Abbildung 5-9 Konsole der graphischen Oberfläche

5.5 Einbindung von Diensten

Für die Einbindung der Dienste in das Betriebssystem wurde das *JavaService Framework*[26] (siehe [Service]) genutzt. Dabei ist es unerheblich, ob es sich um ein Windows oder Linux Betriebssystem handelt. Lediglich die passende Ausführungsdatei muss zur Plattform passen. Unterschiedliche Versionen stehen auf der *JavaService* Homepage zum Download bereit. Die weitere Verarbeitung übernimmt dabei *JavaService*. Bevor das Programm genutzt werden kann, muss die Klasse *Schedule (abruf.client)* als Jar Archiv exportiert werden. Die Programmdateien von *JavaService* werden bei der Veröffentlichung mitgeliefert und können so über einen Parameteraufruf die Jar Datei als Dienst registrieren (Klasse *ServiceInst*). Ebenso ist es möglich den Dienst zu stoppen oder zu starten. Ein Neustart wird intern durch die Kombination von stoppen und starten erzielt. Bei jeder Änderung der Zeiteinstellung muss der Dienst automatisch neugestartet werden. Dabei werden sämtliche Zeiteinstellungen aus der Konfigurationsdatei neu ausgelesen (siehe Abbildung 5 12). Bei der Entwicklung wurde dieser Algorithmus noch nicht implementiert, da keine effiziente Entwicklung möglich ist, wenn für Tests eine Jar Datei erstellt werden muss. Dies sollte erst durchgeführt werden, wenn keine Änderungen am Quellcode mehr erfolgen. Der Algorithmus der Änderungen an den Zeiteinstellungen ist jedoch schon rudimentär vorhanden. Eine Weiterentwicklung ist daher in kürzester Zeit möglich.

Der Funktionsumfang von *JavaService* kann durch den Erwerb der *Professional* oder *Standard Edition* erweitert werden. Die Implementierung des Prototyps nutzt die *Community Edition*, welche jedoch bei weitem nicht alle verfügbaren Funktionen benötigt.

5.6 Logging

Datenbasis für die Konsole ist die Log Datei, die durch das *„Logging Modul“* verwaltet wird (Klasse *JLogger*). Sämtliche vom Programm erzeugten Meldungen werden durch dieses Modul verwaltet. Neben der nativen Java Logging Bibliothek existiert das log4j Framework. Hierbei handelt es sich um einen de facto Standard der Apache Software

[26] Java Service steht unter GNU-Lizenz. Damit fallen keine Lizenzgebühren an und eine kommerzielle Nutzung ist legitim. Veröffentlichte Software, welche Java Service nutzt ist jedoch ebenfalls unter der GNU-Lizenz zu veröffentlichen. Der Quellcode muss ebenfalls öffentlich gemacht werden.

Foundation. Das Framework steht damit unter Apache Lizenz[27]. Meldungen werden primär nicht an die Standardausgabe, sondern zum Loggingsystem weitergeleitet. Dort können sogenannte Appender hinzugefügt werden, an die sämtliche Meldungen weitergeleitet werden. Bei der Implementierung wurden ein *Console* und ein *Fileappender* verwendet. Dadurch wird jede Meldung sowohl in der Konsole angezeigt, als auch in eine Datei geschrieben. Der Fileappender ist dabei so konfiguriert, dass er ab einer Größe von 80 Kilobyte eine neue Datei anlegt, während die alte Datei bestehen bleibt. Das Format der Ausgabe kann sehr einfach konfiguriert werden. Eine Besonderheit bei log4j ist die Einteilung der Nachrichten in Kategorien. So kann abhängig von der Wichtigkeit bzw. Kategorie eine Filterung bei der Erzeugung erfolgen. Der Entwickler muss die Nachrichten natürlich in die richtige Kategorie implementieren. Hierbei können sechs Kategorien ausgewählt werden: *„TRACE“, „DEBUG“, „INFO“, „WARN“, „ERROR“, „FATAL“* (vgl. [log4j]).

Für log4j existieren mittlerweile Implementierungen für alle bedeutenden Programmiersprachen (log4c, log4cplus, log4perl, log4net etc.).

5.7 Meldungen

Wie in Abschnitt 5.6 beschrieben wurde, werden sämtliche Meldungen an das Logging Modul weitergegeben. Diese werden jedoch nur in die Kategorien „INFO“ und „ERROR“ eingeteilt. Ein fataler Fehler wird dem dem Entwickler im Kontext der Fehlermeldung angezeigt und muss daher nicht besonders gekennzeichnet sein. Dadurch ist eine feinere Auflösung der Meldungen nicht nötig. Die Klassen *Error* und *Info* im Paket abruf.both realisieren dabei die Konfigurierung der Meldung. Neben der eigentlichen Meldung wird festgelegt, ob überhaupt eine Speicherung in die Log Datei notwendig ist. Die Ausgabe kann so variabel konfiguriert werden. Ein Beispiel hierfür wäre z.B. ein Fehler beim Start des Servers, wenn bereits ein anderes Programm die Ports blockiert oder dasselbe Programm bereits läuft. Die Ausnahme wird dabei übergeben und über die Klasse ausgegeben (siehe Tabelle 5 3). Dies minimiert Quellcode und spart Zeit, da der Entwickler nicht bei der Fehlermeldung den Quellcode neu schreiben

[27] Bei der Apache Lizenz handelt es sich um eine Freie-Software-Lizenz.

muss. Darüberhinaus wird ein Grund des Fehlers übergeben, der z.B. als Überschrift des Fensters präsentiert wird. Als letztes wird festgelegt, ob ein Fenster angezeigt bzw. ein Logging erfolgen soll. Es macht keinen Sinn den fehlerhaften Start des Programms zu protokollieren. Jedoch muss dem Anwender die Fehlermeldung angezeigt werden. Eine fehlgeschlagene Anfrage ist dagegen zu protokollieren.

```
public class Error {
        public Error(Exception ex, String reason,
                                        boolean showDialog, boolean logError){
        //...
        }
}
```

Tabelle 5-3 Verwaltung von Ausnahmen durch die Klasse *Error*

```
{
//Programmstart
//...
} catch (Exception ex) {
        new Error(ex, "Server konnte nicht gestartet werden", true, false);
}
```

Tabelle 5-4 Beispiel für die Verwendung der *Error*-Klasse

5.8 Konfigurationsdatei

Die Baumstruktur von Abbildung 4 11 wurde durch die Konfigurationsdatei umgesetzt (siehe Abbildung 5 10). Neben den verbindungsspezifischen Elementen wie Hostname, Benutzername, Passwort und Ablageverzeichnis werden ebenfalls die Zeiteinstellungen für jede Verbindung gespeichert. Dabei ist der objektorientierte Ansatz von XML Dateien sehr vorteilhaft. Zumal davon auszugehen ist, dass der Zugriff über einen Par ser leistungsfähiger sein sollte als der Zugriff auf herkömmliche Textdateien.

Die Komponente des Konfigurationsmanagers wurde durch die Klasse *XmlFunctions* umgesetzt. Zahlreiche Methoden bieten so Abfrage und Modifikationsfunktionen der XML Datei. Jede Änderung an der Konfigurationsdatei sollte über die bereitgestellten Funktionen abgehandelt werden. Eine manuelle Änderung kann zu Fehlern in der Da

tenstruktur führen. Trotz ausgiebiger Ausnahmebehandlung kann das Programm in solchen Fällen nicht mehr ordnungsgemäß funktionieren.

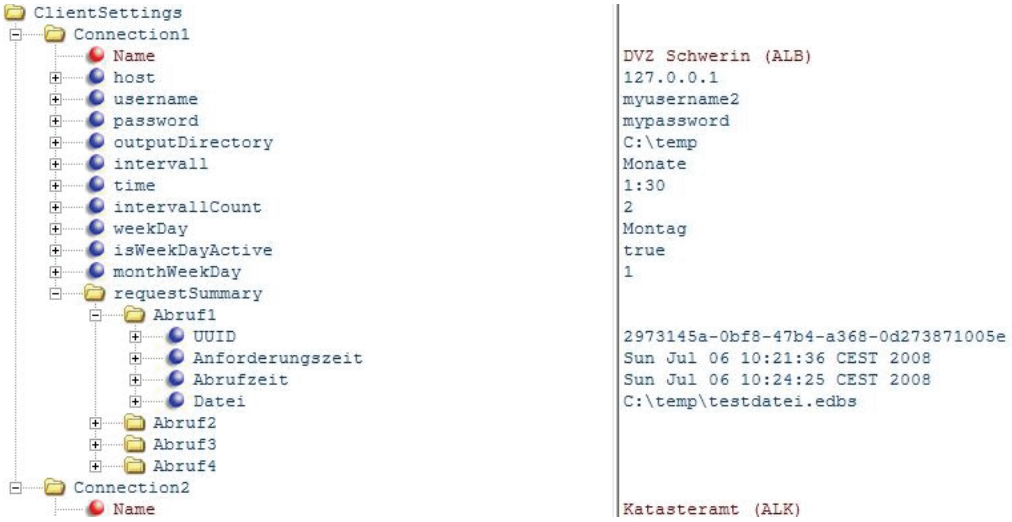


Abbildung 5-10 Beispielinhalt der Konfigurationsdatei

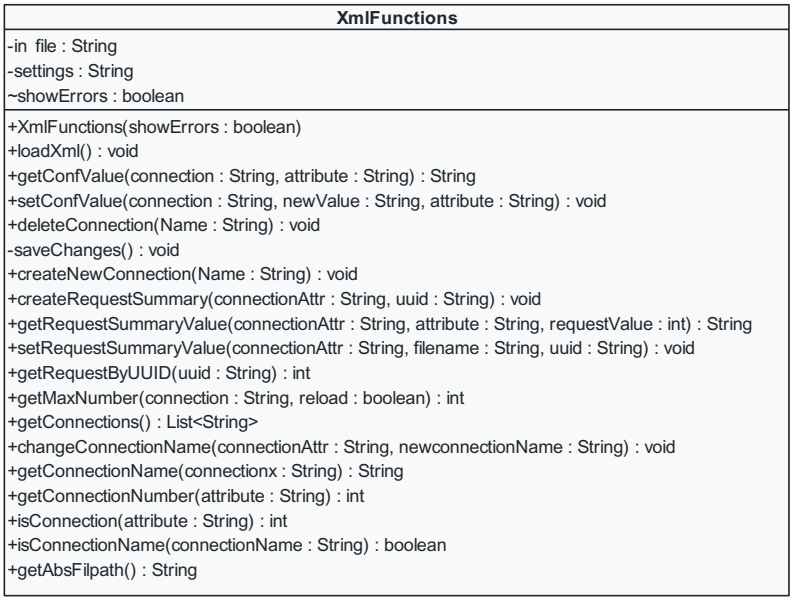


Abbildung 5-11 Klassendiagramm *XmlFunctions* (abruf.client)

5.9 Zeitplanung

Für die Zeitplanung wurde das Scheduling Framework *Quartz* zur Implementierung verwendet. Genau wie log4j (siehe Abschnitt 5.6) steht Quartz unter Apache Lizenz. Ei ne freie Nutzung ist somit gewährleistet. Meldungen von Quartz werden ebenso an log4j weitergeleitet.

Zeitgesteuerte Ereignisse können durch *Quartz* sehr effizient umgesetzt werden. Dazu muss zunächst der Scheduler gestartet werden. Mit der *SchedulerFactory* kann *Quartz dann* gestartet werden. Sogenannte *Jobs* können nun dem *Scheduler* hinzugefügt wer den. Dabei wird dieser so konfiguriert, dass er die entsprechende Klasse zum gewähl ten Zeitpunkt instanziiert. Zusätzlich wird ein eindeutiger Name zugeordnet (siehe Ta belle 5 5).

```
JobDetail jobDetail = new JobDetail("myJob", null, DumbJob.class);
```

Tabelle 5-5 Erstellung eines Jobs in Quartz

Der Auslöser (engl. *Trigger*) kann dazu sehr flexibel konfiguriert werden. Quartz bietet zwei verschiedene Implementierungen des Triggers an. Der *SimpleTrigger* dient dazu, einen Job innerhalb eines bestimmten Intervalls auszuführen, welches separat konfigu riert wird. Für die Abrufsoftware wird jedoch ein kalenderspezifischer Auslöser benö tigt, was alleine mit dem *SimpleTrigger* nicht realisiert werden kann. Der zweite Auslö ser, der *CronTrigger*, bietet dazu hinreichende Kalenderfunktionen. Dabei wird dieser über einen eigenen String Ausdruck angesprochen (*Cron Expressions*). Ein einziger Ausdruck ist in bis zu sieben Unterausdrücke gegliedert.

1. Sekunden,
2. Minuten,
3. Stunden,
4. Tag des Monats,
5. Monat,
6. Wochentag und
7. Jahr (optional) (vgl. [Quartz]).

Tabelle 5 6 verdeutlicht die Flexibilität, die mit den *Cron Expressions* erreicht werden kann. Zudem soll die Arbeitsweise der Ausdrücke aufgezeigt werden. Jede der sieben Spalten wird durch ein Leerzeichen getrennt. Verschiedene Sonderausdrücke sind für jede Spalte möglich.

Ausdruck	**Bedeutung**
"0 0/5 * * * ?"	Auslöser wird alle fünf Sekunden ausgelöst
"10 0/5 * * * ?"	Auslöser wird alle fünf Sekunden ausgelöst, wenn es zehn Sekunden nach der vollen Minute ist
"0 30 10-13 ? * WED,FRI"	Auslöser wird jeden Mittwoch und Freitag um 10:30, 11:30, 12:30 und 13:30 ausgelöst
"0 0/30 8-9 5,20 * ?"	Auslöser wird jede halbe Stunde zwischen 8 Uhr und 10 Uhr ausgelöst, wenn das Monatsdatum zwischen dem 5 und 20 liegt

Tabelle 5-6 Cron-Expressions –Beispiele (vgl. [Quartz])

Sämtliche Auslöser erhalten ebenfalls einen eindeutigen Namen. Für den Start des Jobs wird der zugehörige Auslöser übergeben.

In der Abruf Software sind neben den zeitgesteuerten Abrufen auch manuelle Anfragen möglich. Um die Komplexität der Software nicht unnötig zu erhöhen sind diese ebenfalls durch *Jobs* realisiert. Dafür wird ein *SimpleTrigger* verwendet, der den Job unmittelbar ausführt.

Die Klasse *Schedule* wird für die Zeitsteuerung über einen Parameteraufruf gestartet. Bevor der *Job* erzeugt werden kann muss bekannt sein, um welche Art von Anfrage (zyklisch, manuelle Datenanfrage, manuelle Datenholungsanfrage) es sich handelt. Zu dem muss bei einer manuellen Anfrage festgelegt werden, welche Verbindung angefragt werden soll. Wiederum wurde als Identifikation für Datenholungsanfragen die UUID genutzt. Dies ist notwendig, da sich die Datenholungsanfrage auf eine vorherige Datenanfrage bezieht. Bei einer zeitgesteuerten Anfrage werden immer alle Verbindungen initialisiert. Ein Java Programm, welches als Dienst registriert ist, startet den Abruf (siehe Abschnitt 5.5). Die weitere Verwaltung wird dann vom Quartz Framework übernommen.

Die Klasse *Request* im Paket *abruf.client* untersucht die übergebenen Parameter. Au ßerdem werden Abfragen bereitgestellt, die Aufschluss über die Art der Anfrage bein halten. Im Verbindungmanager ist zu jeder Verbindung die Zeiteinstellung konfiguriert worden. Der Konfigurationsmanager hat die Daten in die Konfigurationsdatei abgelegt. Zur Laufzeit werden die Parameter für die benötigten Verbindungen ausgelesen. Ab bildung 5 12 stellt die Erzeugung des Jobs in einem Aktivitätsdiagramm dar. *Quartz* nutzt dabei als Logging Komponente ebenfalls das log4j Framework. Die verwendeten Funktionen sind nur ein kleiner Teil der Funktionen, die das *Quartz Framework* be reitstellt. Beispielsweise können Jobs auch in einer Datenbank gehalten werden.

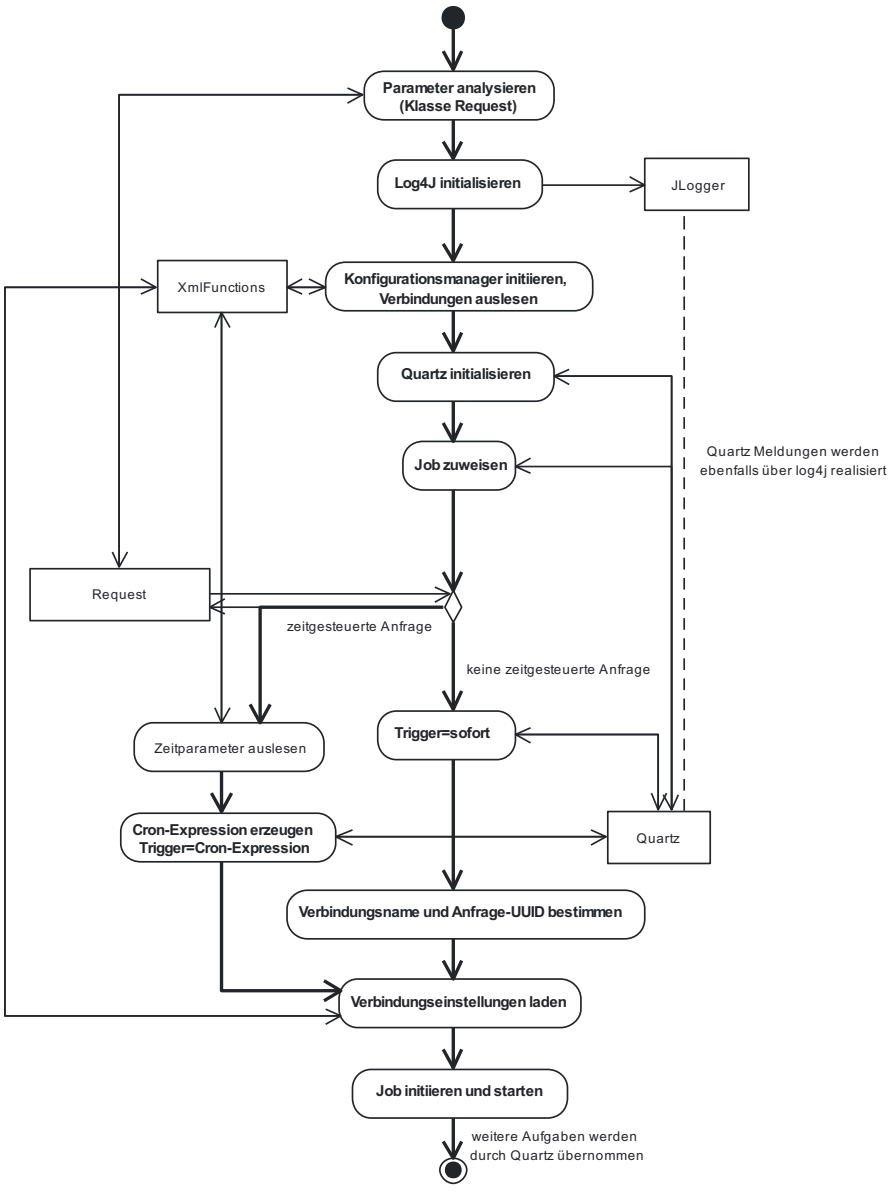


Abbildung 5-12 Aktivitätsdiagramm Klasse *Schedule*

5.10 Anfragevorgänge

Das Auslesen der Verbindungseinstellungen wird u.a. durch die Klasse *RequestJob* er zielt. Verbindungsspezifische Abfragen werden über die Klasse *Request* realisiert. Da *Request* von der Klasse *RequestJob* erbt, ist keine Zwischenspeicherung dieser Parame ter notwendig. Der Verbindungsaufbau zum Server wird dabei durch die Klasse *Con nection* aufgerufen. Darin ist z.B. auch der Anmeldevorgang oder das Laden des Securi ty Managers enthalten. Sämtliche Verbindungsparameter werden übergeben. Eine ei gene Klasse *Property* hält diese Einstellungen, welche bei Bedarf zur Laufzeit abgefragt werden können. Wird z.B. der Verbindungsname benötigt, so kann dieser mit einer Ge tter Methode[28] abgefragt werden. Dies ist ebenfalls für alle anderen Verbindungsein stellungen notwendig.

Ist eine Datenanfrage gestellt worden, so wird diese in der Access Datenbank regist riert. Tabelle 5 7 zeigt einen Beispieleintrag. Der Primärschlüssel ist dort nicht zu se hen, da er nur intern relevant ist. Gesucht wird nach dem Benutzernamen und der UUID. Wie in Abschnitt 4.7.2 beschrieben wurde, fehlt derzeit noch ein Feld für die Ordnungsmerkmale der einzelnen Datensätze. Der Pfad gibt den Verweis zu den er zeugten Datensätzen an, bzw. zum übergeordneten Verzeichnis. Dieses Verzeichnis hat den gleichen Namen wie die UUID.

UUID	Benutzer	Anforderungs-zeit	Abrufzeit	AbrufIP	Pfad
f3c8b0b1-673f-4478-9e8a-157b5d5ac1f5	username	Sun Jul 06 10:06:51 CEST 2008	Sun Jul 06 10:09:31 CEST 2008	169.254.155.76	C:\...\ServerDir\f3c8b0b1-673f-4478-9e8a-157b5d5ac1f5

Tabelle 5-7 Beispiel für einen Datenbankeintrag

Der Datenbankeintrag zeigt einen bereits heruntergeladenen Anfragetupel an (Abruf IP/Abrufzeit ist vorhanden). Ist das Feld *Abrufzeit* und *AbrufIP* leer, so wurden die Da ten noch nicht abgeholt. Nach Abschluss der Datenerzeugung durch ein externes Pro gramm wird abschließend der Pfad in den Datensatz eingefügt.

[28] Mit Getter-Methoden können private Attribute von Objekten ausgelesen werden. Auf der anderen Seite können mit Setter-Methoden diese Attribute festgelegt werden.

5.11 Datentransfer

Wird eine erfolgreiche Datenholungsanfrage gestellt, so ist letztendlich ein Daten transfer nötig. Die Klasse *DataTransfer* organisiert dabei den Datenaustausch. Die Da tenerzeugung hat vorher schon die angeforderten Dateien erzeugt und in den ausge handelten Ordner im Dateisystem verschoben. Nachdem sämtliche Dateien aus dem Anfrageordner in einem Array gespeichert sind, können diese nacheinander angefor dert werden. Der Server nutzt dabei eine weitere Klasse *Gzip*, um die Dateien zu komp rimieren. Für die Komprimierung wird dabei das GZIP Format (Dateiendung: *.gz*) ge nutzt. Die Spezifikation des Formats ist in [RFC 1952] beschrieben. Im Gegensatz zum ZIP Format ist beim GZIP Format nur die Komprimierung einzelner Dateien möglich. Für den Datentransfer ist dies ausreichend und außerdem ist der Implementierungs aufwand geringer. Java bietet dazu ebenfalls native Methoden. Der *GZIPInputStream* ist dabei für die Komprimierung, der *GZIPOutputStream* für die Dekomprimierung zu ständig. Die komprimierte Datei wird automatisch erstellt. Die Ausgangsdatei wird da nach gelöscht (vgl. [Middendorf, et al., 1999]).

Damit der Datenstrom zwischen Client und Server funktioniert, ist eine Wrapper Klasse (auch Mantelsklasse genannt) notwendig. In Java gibt es zu den primitiven Datentypen jeweils eine zugehörige Wrapper Klasse. Für *int* existiert die Wrapper Klasse *Integer,* für *double* die Wrapper Klasse *Double*. Die Übertragung von Dateien in Java kann über einen *Filestream* realisiert werden. Da der Client aber keinen direkten Zugriff auf das Serverdateisystem hat, kann er den Dateistrom nicht nutzen. Die Wrapper Klasse übernimmt die Datenübertragung byteweise und löst das Problem, indem es über das Remote Interface ausgeführt wird. Die Implementierung dieser Klasse wurde als *Byte Buffer (abruf.both)* bezeichnet.

Die Klasse *DataStorage* liefert die Verweise zu den zusammengestellten Dateien. Be ginnt der Datentransfer, so wird automatisch eine Fortschrittsanzeige eingeblendet. Sind mehrere Dateien angefordert worden, so werden diese sequentiell übertragen. Hierzu schließt sich das Fenster nach einigen Sekunden automatisch. Vorher wird na türlich dem Nutzer noch angezeigt, dass der Transfer erfolgreich verlaufen ist. In der

Titelleiste des Fensters werden der Verbindungsname und die gerade zu übertragene Datei angezeigt. Außerdem wird die Downloadrate eingeblendet (siehe Abbildung 5 13). Die Implementierung ist in der Klasse *Progress* realisiert worden.

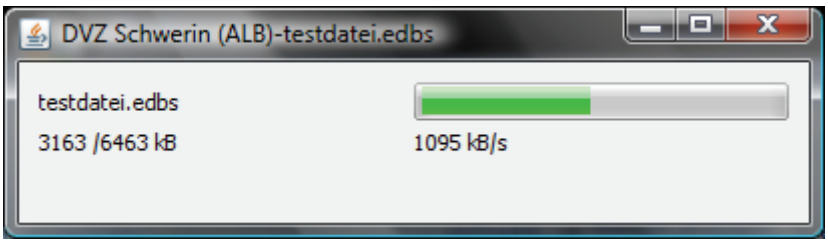


Abbildung 5-13 Fortschrittsanzeige während des Datentransfers

In die Konfigurationsdatei wird zusätzlich zur serverseitigen Referenzhaltung ein Ele ment eingefügt. Damit wird die Referenz clientseitig gespeichert und zur jeweiligen Verbindung zugeordnet. Abbildung 5 10 stellt den Inhalt dieser Datei beispielhaft dar. Die Kindelemente von *requestSummary* sind die bereits gestellten Anfragen. Im Ver bindungsmanager können die Daten über die graphische Oberfläche angezeigt wer den. Zum Abschluss werden die Daten über die Klasse *Gzip* entpackt. Über das Logging wird der erfolgreiche Datentransfer gemeldet. Derzeit löscht der Server die herunter geladenen Daten vom Server abschließend.

5.12 Dokumentation

Für die Implementierung wurde eine automatische Dokumentation angelegt. Hierfür wurde *Javadoc* aus dem JDK (*Java Development Kits*) verwendet. Die durch Sun Micro systems erzeugten APIs (engl. *Application Programming Interface*) wurden ebenfalls mit diesem Werkzeug erstellt. Mit der Entwicklungsumgebung Eclipse kann dadurch eine effektive Erstellung oder Aktualisierung der Dokumentation erfolgen. Aus spe ziellen Doclet Tags des Quellcodes erzeugt *Javadoc* HTML Seiten die Interfaces, Klas sen, Methoden und Felder beschreiben. Der Vorteil dieses Verfahrens ist die einheitli che Haltung von Dokumentation und Quellcode in einer Datei. Bei Änderungen muss dabei ebenfalls der entsprechende *Doclet Tag* angepasst werden. Wird beispielsweise eine neue Methode erzeugt, so muss diese beschrieben werden. Mit einem weiteren Werkzeug *JAutodoc* kann dies jedoch weitgehend automatisiert werden. Auf der ande ren Seite wirkt sich die Größe des Quellcodes etwas negativ auf die Übersichtlichkeit

aus. Tabelle 5 8 zeigt ein Extrembeispiel für die Beschreibung der Remote Schnitt stelle. Dabei werden sämtliche Parameter beschrieben. Zudem können direkt Verweise auf andere Objekte gesetzt werden (*„@see“*). Die Verwendung von *Javadoc* ist daher für diese Größe des Projekts nicht unbedingt notwendig. Eine herkömmliche Doku mentation kann ausreichen. Da jedoch eine Weiterentwicklung des Prototyps an gestrebt wird, ist das Ausmaß der Erweiterungen ungewiss.

```
/**
 * LoginMethode am Server. <ul>
 * <li>Login Modul für Client Login erfolgreich
        => liefern der privaten Methoden</li>
 * <li> Login fehlgeschlagen => RemoteLoginException</li>
 * </ul>
 * @param username
 *         Benutzername
 * @param password
 *         Passwort
 *
 * @return gibt Abruf Objekt zurück (zugriffsgeschützte Mehoden)
 *       @see abruf.server.AbrufImpl
 *
 * @throws RemoteException
 *          Ausnahme beim Aufruf der entfernten Methode
 * @throws RemoteLoginException
 *          Ausnahme bei der Serveranmeldung
 * @throws Exception
 *          allgemeine Ausnahme
 */
Abruf login(String username, String password) throws RemoteException,
         RemoteLoginException, Exception;
```

Tabelle 5-8 Beschreibung der Remote-Schnittstelle mit Doclet-Tags

Schnittstellendokumentation

Möchten die Unternehmen die systemneutrale Software in der entsprechenden Lie genschaftsverwaltungssoftware nutzen, so ist eine exakte Beschreibung der Schnitt stellen nötig. Dabei muss im Einzelnen standardisiert sein, wie ein Aufruf von beiden Seiten umgesetzt wird. Praktisch sieht die Realisierung sehr einfach aus, da lediglich Parameter übergeben werden müssen.

6 Tests

Das Zusammenspiel der einzelnen Komponenten kann in der Praxis immer wieder zu Problemen führen. Selbst der beste Entwurf kann dabei Tests unter realen Bedingungen nicht ersetzen. Eine besondere Herausforderung stellt dabei die Interaktion mit dem Betriebssystem dar. Die Entwicklung der verteilten Anwendung erfolgte in einer lokalen Umgebung. Dadurch sind die Pakete *„client“* und *„server“* nicht örtlich getrennt. Greift die Client Anwendung beispielsweise auf das Paket „server“ zu, so würde dies ohne das Wissen des Entwicklers geschehen. Bei einer Veröffentlichung hätte jedoch der Client dieses Paket gar nicht verfügbar und ein Laufzeitfehler würde die Anwendung zum Absturz bringen. Implementierungsfehler, die nicht vom Compiler abgelehnt werden, können so behoben werden. Netzwerkspezifische Probleme konnten durch die lokale Umsetzung ebenfalls nicht hinreichend simuliert werden. Damit die Leistungsfähigkeit des Programms optimiert werden kann, muss letztlich eine Analyse der Puffer Einstellungen erfolgen.

6.1 Netzwerkbetrieb

Wichtigster Schritt ist dabei zunächst die Trennung der nicht zusammengehörigen Pakete. Dazu werden zwei separate Anwendungen kompiliert. Dies sind einerseits der Server, welcher die datenhaltende Stelle (Pakete „both“ und „server“) repräsentieren soll, sowie der Client (Pakete „both“ und „client“). Jede Anwendung wurde auf einen eigenen Rechner gestartet, welche über ein Netzwerk miteinander verbunden sind. Ziel war es hierbei, die Funktionsfähigkeit und Unabhängigkeit der beiden einzelnen Anwendungen außerhalb der lokalen Umgebung unter Beweis zu stellen. Nachdem einige Laufzeitfehler behoben wurden, konnten Anfragen erfolgreich an den Server gestellt werden. Die Leistungsfähigkeit im Netzwerkbetrieb ist in Abschnitt 6.3 analysiert worden.

6.2 Internetbetrieb

Besonders kritisch ist der Betrieb im Internet, weil für den Server dabei eine Adressumsetzung im Router konfiguriert sein muss. Firewalleinstellungen müssen die Nut

zung der RMI Ports zulassen. Der Client (Internetverbindung über GPRS) versuchte bei diesem Test Anfragen an den Server zu stellen, wobei der Client eine Softwarefirewall nutzte. Die Client Einstellungen erlaubten nur die Nutzung des HTTP Protokolls. Die Vorgehensweise von RMI ist dabei in Abschnitt 3.3.2 beschrieben. Sämtliche Daten konnten daher erfolgreich bezogen werden. Das Anzeigen der Downloadrate war auf grund der langen Transferzeiten (bedingt durch GPRS) sehr hilfreich. Außerdem arbei tet die Fortschrittsanzeige intervallbasiert. Im nachfolgenden Abschnitt wird die Leis tung im lokalen Betrieb sowie im Intranet und Internetbetrieb analysiert.

6.3 Leistungsbewertung

Bei den nachfolgenden Tests soll die Leistungsfähigkeit des Systems in Abhängigkeit der Puffereinstellung und Netzwerkanbindung analysiert werden. Die gelieferten Re sultate sollen auch eine optimale Konfiguration der Software zum Ziel haben. Für die potentiell performanteste Lösung wurde ein lokaler Betrieb für in der Testumgebung gewählt. Unterstützt wurden die Daten durch eine 100 Mbit Netzwerkverbindung und eine langsame GPRS Verbindung (engl. *General Packet Radio Service*). Dabei bestand die Testumgebung aus zwei Rechnern.

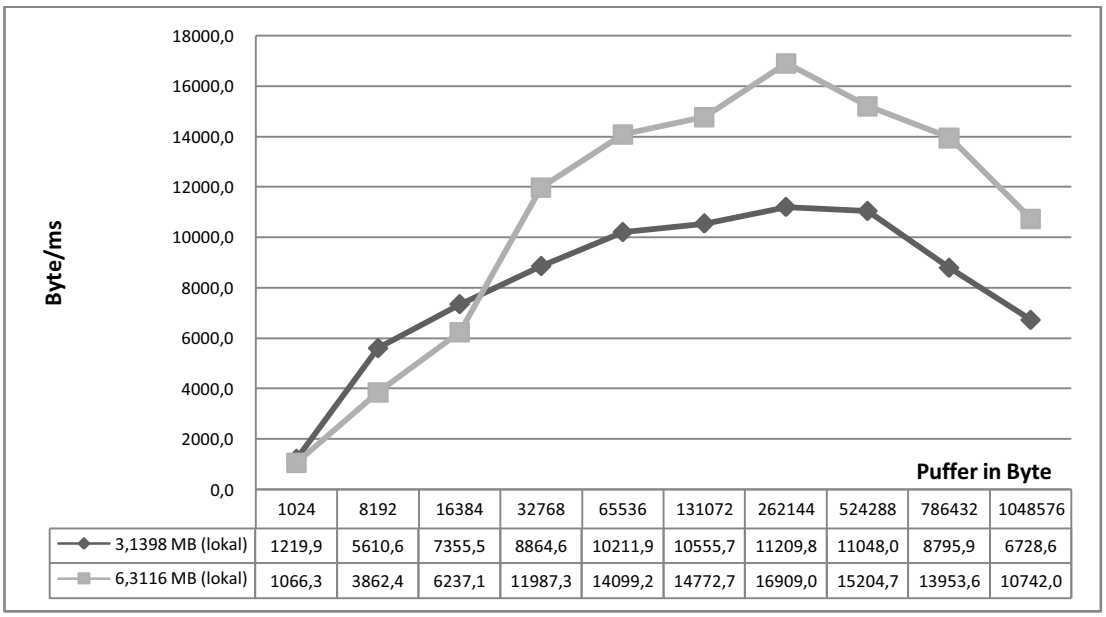
	Rechner 1 (Server)	Rechner 2 (Client)
Modell:	Workstation	Acer Aspire 5920G (Notebook)
Prozessor:	AMD Athlon™ 64 3700+ 2.21 GHz	Intel®Core™2 Duo CPU T7300 @2GHz
Arbeitsspeicher:	2048 MB DDR2	2048 MB DDR2
Betriebssystem:	Windows Vista Business 64 Bit	Windows Vista Home Premium 32Bit
Festplatte:	Hitachi/IBM IC35L090AVV207-0 Ultra-ATA/100 7200 U/min 2 MB Cache 8,5 ms Zugriffszeit	Seagate ST9160821AS SATA2 300MB/s 5400 U/min 8MB Cache 12,5 ms Zugriffszeit

Tabelle 6-1 Leistungsdaten der eingesetzten Test-Rechner

Damit die Testbedingungen realitätsnah sind, wurden EDBS Datensätze der Größe 61,4 MB und 32,2 MB verwendet. Da es sich um ASCII Dateien handelt, konnte etwa eine Kompressionsrate von 1:10 erreicht werden. Die zu übertragenden Dateien waren

demnach nur 6,3 MB und 3,1 MB klein. Die Komprimierung der Dateien erwies sich dabei als eine sehr sinnvolle Maßnahme.

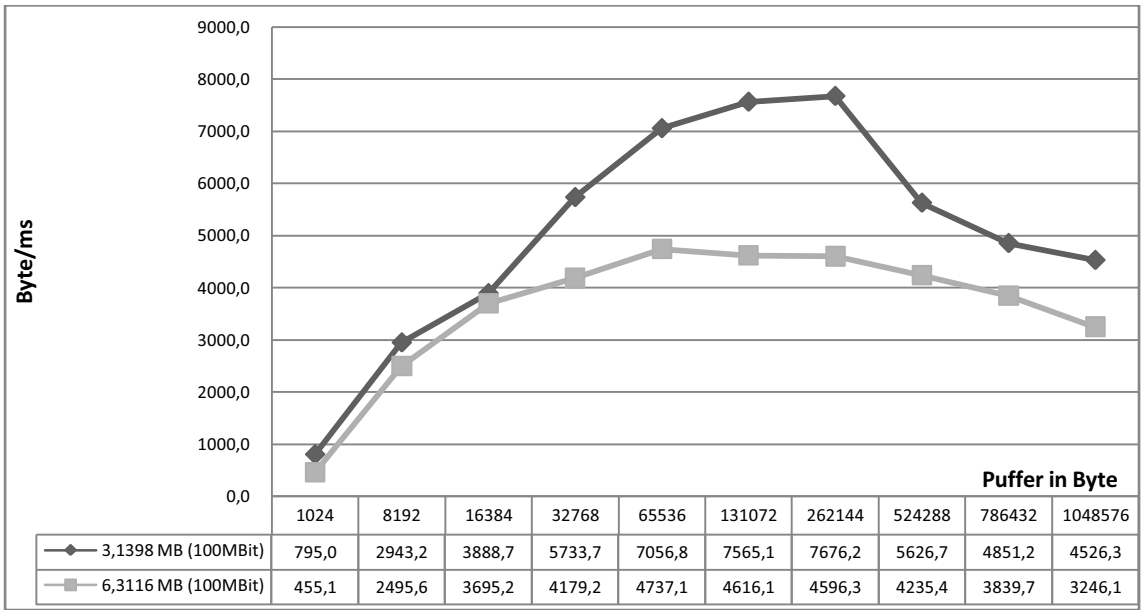
Betrachtet wurde nicht der gesamte Anfragevorgang, sondern lediglich die Zeit während der Datenübertragung. Die Komprimierungszeit wurde nicht miteinbezogen. Für die Puffergrößen wurden nach einigen Testläufen die Werte 1 Kb, 8 Kb, 16 Kb, 32 Kb, 64 Kb, 128 Kb, 256 Kb, 512 Kb und 1024 Kb gewählt. Kleine Puffergrößen zeigten bei den Anfangstests sehr mäßige Resultate. Zudem waren noch Stützwerte zwischen 512 Kb und 1024 Kb notwendig, um Trends besser untersuchen zu können (bei lokal und Intranet). Gemessen wurde jede Puffereinstellung mit den beiden Dateigrößen in Kombination mit den drei verschiedenen Netzwerkanbindungen. Jede Messung wurde 10 mal durchgeführt. Danach wurde der Mittelwert gebildet. Aufgrund der langen Übertragungszeiten bei der GPRS Verbindung wurden hier nur Einzelmessungen vorgenommen.



	1024	8192	16384	32768	65536	131072	262144	524288	786432	1048576
3,1398 MB (lokal)	1219,9	5610,6	7355,5	8864,6	10211,9	10555,7	11209,8	11048,0	8795,9	6728,6
6,3116 MB (lokal)	1066,3	3862,4	6237,1	11987,3	14099,2	14772,7	16909,0	15204,7	13953,6	10742,0

Abbildung 6-1 Leistungstest lokal

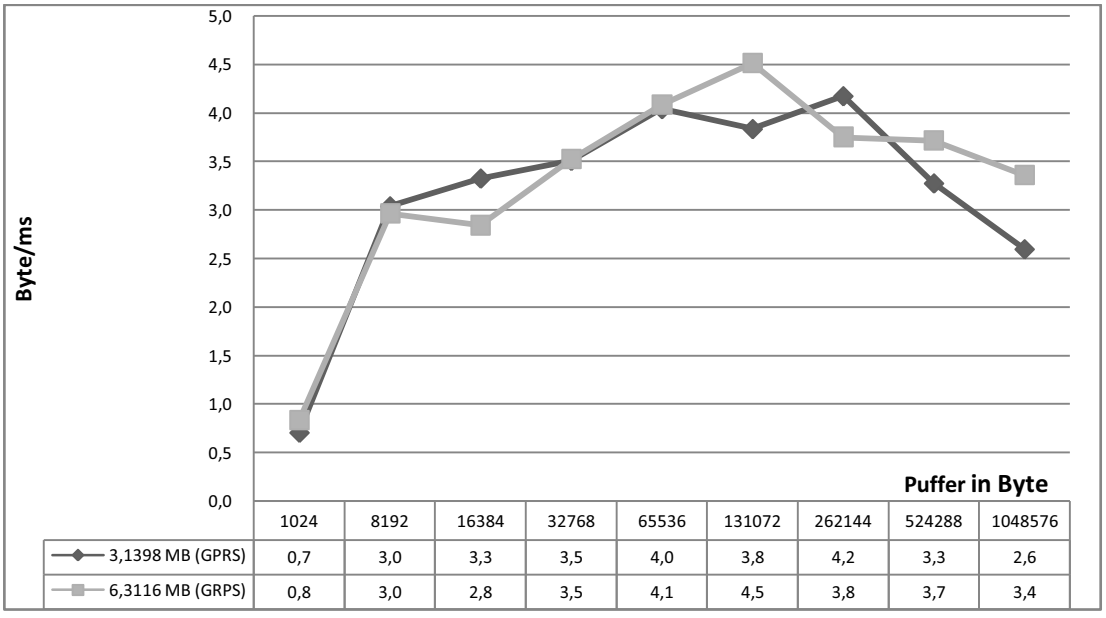
Abbildung 6 1 zeigt, dass die Diskrepanzen zwischen den Puffereinstellungen enorm sind (minimaler Wert: 1066,3 Byte/ms; maximaler Wert: 16909,0 Byte/ms). Der beste Datendurchsatz wurde von beiden Dateigrößen mit der Puffereinstellung 256 Kb erzielt. Größere Puffereinstellungen ließen den Datendurchsatz wieder sinken.



	1024	8192	16384	32768	65536	131072	262144	524288	786432	1048576
3,1398 MB (100MBit)	795,0	2943,2	3888,7	5733,7	7056,8	7565,1	7676,2	5626,7	4851,2	4526,3
6,3116 MB (100MBit)	455,1	2495,6	3695,2	4179,2	4737,1	4616,1	4596,3	4235,4	3839,7	3246,1

Abbildung 6-2 Leistungstest 100MBit

Abbildung 6 2 zeigt den Datendurchsatz im 100 MBit Netzwerk. Auch hier sind sehr ähnliche Trends zu beobachten. Bis zum 256 Kb Puffer steigt der Durchsatz. Danach ist ein Abfall zu beobachten. Bei den unterschiedlichen Dateigrößen erzielt die Dateigröße von 3,1 MB die besten Werte.



	1024	8192	16384	32768	65536	131072	262144	524288	1048576
3,1398 MB (GPRS)	0,7	3,0	3,3	3,5	4,0	3,8	4,2	3,3	2,6
6,3116 MB (GRPS)	0,8	3,0	2,8	3,5	4,1	4,5	3,8	3,7	3,4

Abbildung 6-3 Leistungstest GPRS

Die GPRS Verbindung ist mit 55,6 KBit/s die langsamste Verbindung. Durch den schwa chen Datendurchsatz sind keine Unterschiede zwischen den beiden Dateigrößen fest zustellen. Ebenfalls sind die besten Maximalwerte im Bereich zwischen 64 Kb und 256

Kb. Abbildung 6 4 liefert die Basisdaten für die optimale Pufferkonfiguration. Ausge hend vom Maximalwert jeder Netzanbindung wurden die zugehörigen Werte prozen tual bestimmt. Dabei ist schon deutlich zu erkennen, dass die beste Leistung im Be reich zwischen 64 Kb und 256 Kb zu finden ist. Abschließend müssen jedoch noch die Resultate der einzelnen Netzverbindungen gemittelt werden.

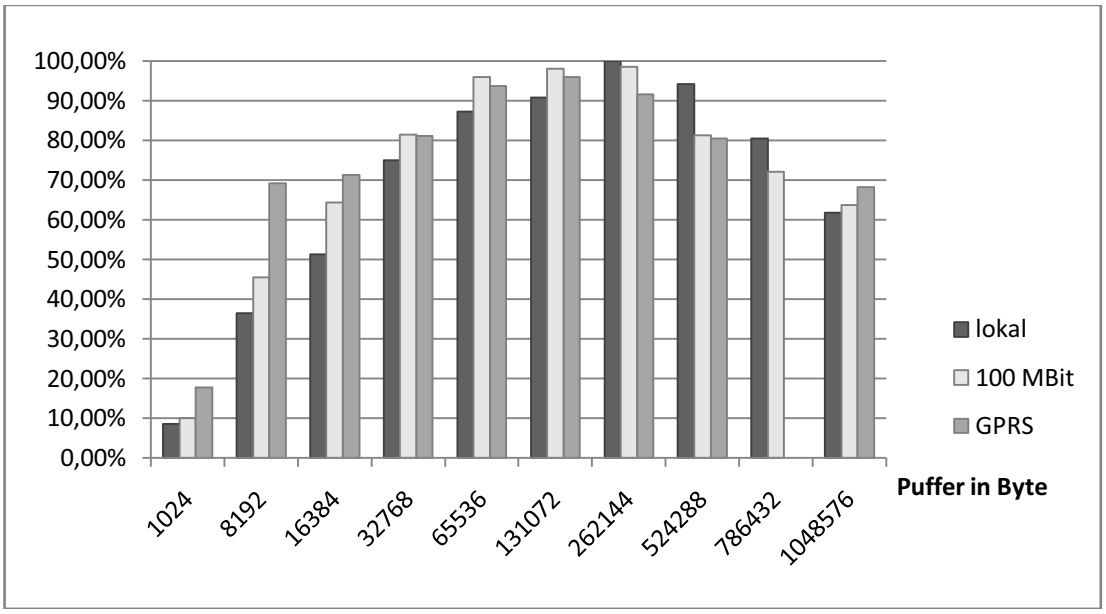


Abbildung 6-4 Datendurchsatz prozentual

Abbildung 6 5 zeigt den maximalen Mittelwert aller Netzanbindungen bei 256 Kb.

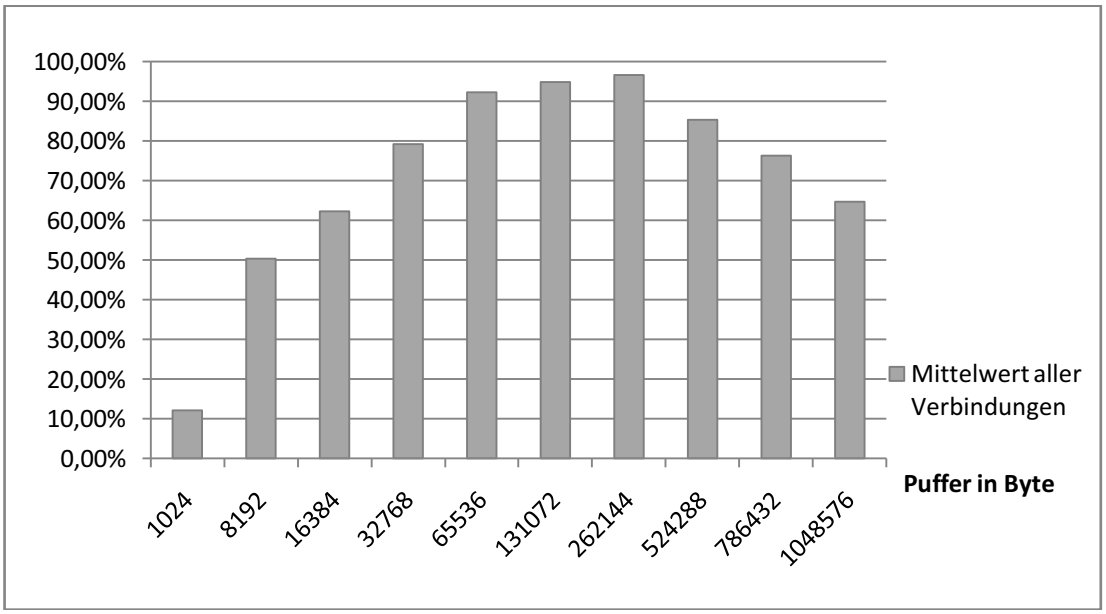


Abbildung 6-5 Mittlerer prozentualer Datendurchsatz

Um die volle Performance des Systems zu erhalten, muss der Puffer daher standard mäßig korrekt konfiguriert sein. Bei einer suboptimalen Konfiguration würde die Per formance des Systems signifikant sinken. In Extremfällen läge die Leistung mit einer Puffereinstellung von 1 kb bei nur ca. 10 % der Maximalleistung. Die Tests haben je doch auch gezeigt, dass es unterschiedliche Resultate mit verschiedenen Dateigrößen geben kann. Bei langsamen Verbindungen sind diese Trends jedoch nicht zu bestäti gen. Durch die Leistungsbewertung ist der leistungsfähige und kostengünstige Betrieb des Systems erzielbar.

6.4 Konkurrierende Zugriffe

Da bei den Tests nur zwei Rechner verfügbar waren, wurden konkurrierende Zugriffe softwaretechnisch realisiert. Hierzu wurde die Clientanwendung mehrfach gestartet, wobei mehrfaches Starten sonst nicht möglich ist. Zeitnah wurden nun mehrere Anfra gen an den Server gesendet. Dieser konnte sämtliche Anfragen nebeneinander abwi ckeln. Durch die suboptimalen Testbedingungen kann jedoch keine Garantie für die Kollisionsfreiheit von Zugriffen gegeben werden.

Es ist zu beachten, dass für die Datenbankzugriffe das ACID Prinzip nicht angewendet wurde. Zum einen stellt der reine Lesezugriff kein Risiko dar. Zum anderen kann eine Manipulation an einem Datensatz logischerweise nicht von zwei Clients angefordert werden. Denn jeder Client besitzt seine eigenen Datensätze, die auch nur dem Client zugeordnet sind. Theoretisch könnte es vorkommen, dass eine UUID doppelt generiert wurde und dadurch ein konkurrierender Zugriff eintritt. In diesem Fall wird eine Aus nahme ausgelöst und eine erneute Anfrage muss gesendet werden. Die einzige Inkon sistenz im Datenschema kann bei der Erzeugung der Primärschlüssel auftreten. Da durch, dass die Threads nebenläufig auf dem Server arbeiten, könnte zweimal derselbe Primärschlüssel für einen neuen Datensatz erzeugt werden. Der Primärschlüssel darf jedoch nur einmal vorkommen. Das Verhalten des Datenbankmanagementsystems ist für diesen Fall ungetestet. Eigentlich sollte wiederum eine Ausnahme ausgelöst wer den. Selbst wenn der doppelte Primärschlüssel eingefügt würde, hätte dies keine Aus wirkung auf die Anwendung, da hier nur nach der UUID in Kombination mit dem Be

nutzernamen gesucht wird. Der Primärschlüssel hat in diesem Zusammenhang keine direkte Relevanz.

6.5 Secure Sockets Layer

Um die korrekte Funktionsweise von SSL prüfen zu können, werden sämtliche SSL Meldungen an die Konsole weitergeleitet. Hierzu ist der Eintrag in „javax.net.debug“ im Propertyset auf „all“ zu setzen. Wichtige Informationen über den Inhalt des Zertifikats werden dabei ausgelesen. Zudem werden die angewandten Sicherheitsverfahren angezeigt. Entsprechen die Einstellungen nicht den Vorgaben, so wird eine Ausnahme ausgelöst. Diese ist auch ohne spezielle Debugging Einstellung sichtbar. Die ersten Implementierungen des Prototyps wurden ohne SSL durchgeführt. Nachdem die Basisfunktionen (wie z.B. Datentransfer) funktionsfähig waren, wurde die Verbindung gesichert. Über das Programm Wireshark[29] wurde der Netzwerkverkehr zwischen Client und Server abgefangen und in speziellen Dump Dateien gespeichert. Insbesondere der Anmeldevorgang wurde hier untersucht. Dabei konnten sämtliche Anmeldeinformationen wie z.B. Benutzername und Passwort abgefangen werden. Dies war relativ einfach, da keine Verschlüsselung erfolgt war. Derselbe Test wurde nach der SSL Implementierung durchgeführt. Danach gestaltete sich allerdings die Suche nach den Anmeldepaketen als schwierig. Alle Daten waren zudem wie gewünscht verschlüsselt. Weitere Tests, in denen falsche Anmelde oder Zertifikatpasswörter verwendet wurden, lehnte der Server wunschgemäß ab. Die Sicherheitsanforderungen des Systems sind im Rahmen der Testumgebung ausreichend. Das verwendete Beispielzertifikat und die unzureichende Passwortwahl muss trotzdem beachtet werden (siehe Abschnitt 5.1).

6.6 Installer

Die Veröffentlichung der Anwendung wurde testweise mit einem Installer durchgeführt. Dabei sollte insbesondere die Portabilität der Anwendung nachgewiesen werden. Zudem diente der Test dazu, Erfahrungen im Umgang mit dem Skripten der Installer Software zu sammeln.

[29] Bei Wireshark handelt es sich um ein Programm zur Analyse der Netzwerkkommunikation.

6.7 Registrierung von Diensten mit JavaService

Die Funktionalitäten von *JavaService* (siehe Abschnitt 6.7) sind sehr vielversprechend. In der Praxis treten jedoch einige Probleme mit Zugriffsrechten bei der Windows UAC (engl. *User Account Control*) auf. Windows Vista besitzt einen Benutzerkontoschutz der bei kritischen Zugriffen die Zustimmung des Administrators verlangt. Der Anwender bestätigt die Aktion per Mausklick. Konkret heißt das, ein Dienst kann nicht ohne Autorisierung durch *JavaService* registriert werden. Das Stoppen oder Neustarten des Dienstes ist demnach auch nicht direkt möglich. Die Ursache hierfür liegt in Java, da die Ausführungsmethode die Autorisierung nicht automatisch anfordert. Es ist aber zwingend notwendig, bei jeder Änderung der Konfigurationsdatei (Zeiteinstellung, Verbindungseinstellungen) den Dienst neu zu starten. Der Konsolenbefehl muss daher unter Administrationsrechten ausgeführt werden. Mit dem „runas“[30] Befehl können Befehle unter anderen Nutzernamen durchgeführt werden. Allerdings führt der Weg nicht an der Eingabe des Passworts über die Eingabeaufforderung vorbei (mit Datenströmen kann dies auch über andere Eingabemasken realisiert werden). Bedenkt man, dass der Dienst bei jeder Änderung der Verbindungsparameter neugestartet werden muss, so wird schnell deutlich, dass dies inakzeptabel ist. Das Abschalten der UAC ist zwar prinzipiell möglich, kann aber nur als Zumutung für den Anwender betrachtet werden. Vielmehr muss eine native Ausführung des Befehls erzwungen werden. Es ist unklar, weshalb Java dies im Gegensatz zu der Programmiersprache C# nicht tut. Dort wird der Benutzer automatisch zum Bestätigen der Aktion aufgefordert. Der Prototyp nutzt daher ein kleines Programm (*exec.exe*), welches über C# implementiert wurde. Übergebene Parameter werden weitergeleitet und ausgeführt. Windows Vista hat die .NET Laufzeitumgebung für das C# Programm standardmäßig installiert. Ältere Windows Versionen sollen das Zwischenprogramm nicht nutzen, sondern den Befehl nur über Java ausführen. Auf der einen Seite kann die .NET Laufzeitumgebung nicht zusätzlich zur Java Laufzeitumgebung gefordert werden, auf der anderen Seite ist die UAC bei älteren Windows Versionen aber nicht vorhanden.

[30] Da aktuelle Betriebssysteme Mehrbenutzersysteme sind, ist es manchmal erforderlich Befehle mit höheren Nutzerrechten auszuführen. Unter Windows ist dies mit dem „runas“-Befehl möglich.

In mehreren Tests konnte die Anwendung für den permanenten Betrieb registriert werden. Dabei sollten Abrufe zu bestimmten Zeiten automatisch im Hintergrund erfol gen. Sämtliche Tests verliefen erfolgreich.

7 Installation und Konfiguration

Für die Installation sollten folgende Mindestanforderungen für den ordnungsgemäßen Betrieb erfüllt sein:

	Server	Client
Prozessor:	>1,5 GHz (Mehrkernprozessor empfohlen)	>1 GHz
Arbeitsspeicher:	1024 MB	512 MB
Betriebssystem:	Derzeit nur Windows Betriebssysteme[31]	Derzeit nur Windows Betriebssysteme
Festplatte:	(RAID-Spiegelung empfohlen)	/
Internetanbindung:	DSL 2000 mit fester IP-Adresse oder DNS (DSL6000 empfohlen)	ISDN (DSL empfohlen)
Sonstiges	Softwarefirewall (Port 1089/1099 bzw. Serveranwendung zulassen) Ist die Datenerzeugung ebenfalls auf dem Server installiert, so ist ein Backup-System erforderlich. JRE[32]-Installation erforderlich	Lokale Installation der individuellen Liegenschaftssoftware für Import notwendig JRE-Installation erforderlich

Ein Ziel der Anwendung sollte die Benutzerfreundlichkeit darstellen. Dazu gehört je doch auch die Dokumentation der Anwendungsfunktionen, welche in einem (Online) Handbuch zusammengefasst werden sollten. Die Dokumentation kann durch eine PDF Datei dem Installationspaket beigefügt werden. Bestandteil dieses Handbuchs muss auch ein Kapitel sein, welches sich mit Installationshinweisen auseinandersetzt. Exemplarisch könnte dabei die Konfiguration des Servers mit Router oder Fire walleinstellungen beschrieben werden. Zusätzliche Konfigurationen bei der Clientan wendung sind nicht erforderlich. Daher muss die Software durch einen Installer kom fortabel zu installieren sein. Das Einbinden der Anwendung in den Installer erfordert minimalen Aufwand, da das Anwendungsverzeichnis keinerlei Registry Einträge benö tigt. Testweise wurde dies bereits durchgeführt.

[31] Die Umsetzung für andere Plattformen erfordert nur geringe Aufwände.
[32] JRE steht für die Java Laufzeitumgebung (engl. *Java Runtime Environment*) die zum Ausführen von Java-Programmen notwendig ist.

Bei der Konfiguration der Anwendung müssen die Verbindungseinstellungen mit Benutzername, Passwort, Ablageordner und Serveradresse konfiguriert werden. Auf Seiten der datenhaltenden Stelle ist die Datenerzeugungsschnittstelle standardisiert. Wichtige Parameter für die Schnittstelle zur Benutzerverwaltung sind lediglich die Datenbankparameter (Benutzername, Passwort, Hostname). Damit ist die Anwendung systemneutral und kann mit bestehenden Systemen verknüpft werden.

ISDN

Verfügt die abrufende Stelle lediglich über eine ISDN Anbindung, so muss diese noch konfiguriert werden. Damit beim Abruf ein automatischer Wählvorgang einsetzt, ist dies in der DFÜ Verbindung vorzunehmen. Eine sinnvolle Option ist ebenfalls das Trennen der Verbindung nach Inaktivität (z.B. 2 Minuten). So muss die Verbindung nicht permanent aufrecht gehalten werden und es entstehen keine unnötigen Kosten.

8 Zusammenfassung und Ausblick

Durch den Entwurf und die Implementierung der Anwendung konnte bewiesen wer den, dass eine vollständige Lösung des systemneutralen automatisierten Abrufs von Daten aus dem Liegenschaftskataster möglich ist. Hierzu sind jedoch die gesetzlichen Beschränkungen bezüglich der Systemsicherheit einzuhalten. Den potentiellen An wendern kann eine Software bereitgestellt werden, die sämtliche Funktionen beinhal tet und modular aufgebaut ist. Weitere Anpassungen sollten daher einfach durchzu führen sein. Da noch einige organisatorische Fragen offen sind, konnten noch nicht alle Funktionen implementiert werden. Damit eine Vollautomatisierung des geplanten Ver fahrens erreicht werden kann, sollte die Datenerzeugung ebenfalls autonom erfolgen. Sämtliche Funktionen sind dazu bereits einsatzbereit. Besonders wichtig ist jedoch, dass nicht von den Basiseigenschaften *Systemneutralität* und *Automatisierung* abgewi chen wird. Hierzu gehört ebenso die *Plattformunabhängigkeit*. Der Prototyp ist zwar temporär für Windows Betriebssysteme ausgelegt, jedoch sind die Änderungen für die Lauffähigkeit auf anderen Plattformen gering. Probleme, wie z.B. die Rechte zum Re gistrieren eines Dienstes, müssen weitsichtig gelöst werden. So ist die Imple mentierung unabhängig von der Plattform anwendbar und ein Maximum an Portabili tät wird gewährleistet. Die exakte Beschreibung der Schnittstellen ist dabei Grundvor aussetzung, damit Hersteller von Liegenschaftsverwaltungssoftware die Abruf Anwendung nutzen können. Wünschenswert wäre eine hohe Flexibilität. Dies bedeutet ebenfalls, dass wenige Anpassungen beim Anwender notwendig wären. Die endgülti gen Implementierungen für die Schnittstellen zur Benutzerverwaltung und Datener zeugung sind mit geringem Anpassungsaufwand realisierbar. Methoden sind hierzu be reits partiell erstellt.

Die Anwendung der Java Technologien hat gezeigt, dass durch die Vielzahl von Frame works eine qualitativ hochwertige, zuverlässige und effiziente Entwicklung fast jeder Problemstellung möglich ist. Besonders gut bewährte sich die einheitliche Nutzung der Frameworks (*Java Secure Socket Extension, Quartz, log4j*) im Zusammenspiel mit RMI. Die flächendeckenden Sicherheitskonzepte von Java erleichterten die Entwicklung enorm. Nicht zuletzt deshalb konnte sich Java in der Technologieentscheidung durch

setzen. Für die Weiterentwicklung des Prototyps sollten daher weiterhin Java Technologien verwendet werden.

Die Vision, dass die Software nur durch die Installation bereits vollständig mit den wei teren Modulen funktioniert, ist nicht nur theoretisch möglich. Gewiss ist dafür ein ho her Implementierungsaufwand nötig. Allerdings führt jeder andere Weg zu Einschrän kungen in Benutzerfreundlichkeit, Flexibilität, Systemneutralität und Automatisierung.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Abts, Dietmar 2007] *Client/Server Programmierung mit Java.* Wiesbaden : Friedr. Vieweg & Sohn Verlag/ GWV Fachverlag GmbH, 2007. ISBN 978 3 8348 0322 1.

[BDSG] Bundesdatenschutzgesetz (BDSG). *juris GmbH.* [Online] [Zitat vom: 29. August 2008.] http://bundesrecht.juris.de/bundesrecht/bdsg_1990/gesamt.pdf.

[Bell, Douglas und Parr, Mike 2003] *Java für Studenten.* München : Person Studium, 2003. ISBN 3 8273 7045 0.

[BNetzA] Bundesnetzagentur. *Zertifizierungsdiensteanbieter.* [Online] [Zitat vom: 1. August 2008.] http://www.bundesnetzagentur.de/enid/1f39643c254eb4a5d1a22d3531f9c644,0/ph.html.

[Cnlab] cnlab AG. *Secret Code Checker.* [Online] [Zitat vom: 5. Juli 2008.] https://www.cnlab.ch/codecheck/check.php.

[Dostal, Wolfgang, et al. 2005] *Service orientierte Architekturen mit Web Services.* Heidelberg : Spektrum Akademischer Verlag, 2005. ISBN 3 8274 1457 1.

[DVZ M-V] DVZ Schwerin GmbH. *ALB/ALB Online.* [Online] [Zitat vom: 26. Mai 2008.] http://www.dvz mv.de/produkte/dienstl_lsg_alb.htm.

[EgoMV] *Zweckverband „Elektronische Verwaltung Mecklenburg Vorpommern“.* [Online] [Zitat vom: 26. August 2008.] http://www.ego mv.de/.

[Fietz, Christian 2008] *Systemneutraler automatisierter Abruf von Daten aus dem Liegenschaftskataster (Projekt Skizze).* Rostock : BTFietz GmbH, 2008.

[Heinzl, Steffen und Markus, Mathes 2005] *Middleware in Java.* Wiesbaden : Friedr. Vieweg & Sohn Verlag/GWV Fachverlag GmbH, 2005. ISBN 3 528 05912 5.

[Hofmann, Johann, Jobst, Fritz und Schabenberger, Roland 2001] *Programmierung mit COM und CORBA.* Wien : Carl Hanser Verlag München Wien, 2001. ISBN 3 446 21479 8.

[LiKatAVO] juris GmbH. *Liegenschaftskataster Abrufverordnung (LiKatAVO M V).* [Online] [Zitat vom: 13. Juni 2008.] http://mv.juris.de/mv/gesamt/LiKatAV_MV_2007.htm.

[Lipp, Peter, et al. 2000] *Sicherheit und Kryptographie in Java.* München : Addison Wesley Verlag, 2000. ISBN 3 8273 1567 0.

[log4j] Apache Logging Services Project. *log4j.* [Online] [Zitat vom: 8. Juli 2008.] http://logging.apache.org/log4j/.

[Melton, Jim und Eisenberg, Andrew 2000] *Understanding SQL and Java Together.* San Diego : Academic Press, 2000. ISBN 1 55860 562 2.

[Middendorf, Stefan und Singer, Reiner 1999] *Programmierhandbuch und Referenz für die Java 2 Plattform.* Heidelberg : dpunkt, 1999. ISBN 3 920993 82 9.

[Neal Harman] Internet Computing. *RMI Tunneling.* [Online] [Zitat vom: 4. Juni 2008.] http://www.cs.swan.ac.uk/~csneal/InternetComputing/Tunnelling.html.

[OASIS] OASIS Standard. *Reference Model for Service Oriented Architecture 1.0.* [Online] [Zitat vom: 4. Juli 2008.] http://docs.oasis open.org/soa rm/v1.0/soa rm.pdf.

[OMGCon] OMG. *Concurrency Service Specification.* [Online] [Zitat vom: 29. Juli 2008.] http://www.omg.org/docs/formal/00 06 14.pdf.

[OMG] OMG. *CORBA.* [Online] [Zitat vom: 30. Mai 2008.] http://www.omg.org/docs/ptc/06 05 01.pdf.

[PIPE] Microsoft. *Microsoft Developer Network.* [Online] [Zitat vom: 30. Juli 2008.] http://msdn.microsoft.com/en us/library/aa365780.aspx.

[Quartz] OpenSymphony. *Quartz Enterprise Job Scheduler.* [Online] [Zitat vom: 14. Juli 2008.] http://www.opensymphony.com/quartz/.

[RFC 1952] IETF. *GZIP file format specification version 4.3.* [Online] [Zitat vom: 29. Juli 2008.] http://www.ietf.org/rfc/rfc1952.txt.

[RFC1122] IETF. *Requirements for Internet Hosts.* [Online] [Zitat vom: 1. August 2008.] http://tools.ietf.org/html/rfc1122.

[RFC4122] IETF. *Universally Unique Identifier (UUID).* [Online] [Zitat vom: 6. Juli 2008.] http://tools.ietf.org/html/rfc4122.

[RFC4510] IETF. *Technical Specification Road Map.* [Online] [Zitat vom: 1. Juli 2008.] http://tools.ietf.org/html/rfc4511.

[RFC4513] IETF. *Authentication Methods and Security Mechanisms.* [Online] [Zitat vom: 1. Juli 2008.] http://tools.ietf.org/html/rfc4513.

[Service] Tanuki Software. *Java Service Wrapper.* [Online] [Zitat vom: 30. Juli 2008.] http://wrapper.tanukisoftware.org.

[SOAPData] W3C. *SOAP.* [Online] [Zitat vom: 16. Juni 2008.] http://www.w3.org/TR/xmlschema 2/#built in primitive datatypes.

[SOAPP0] W3C. *SOAP.* [Online] [Zitat vom: 3. Juni 2008.] http://www.w3.org/TR/2007/REC soap12 part0 20070427.

[SOAPP1] W3C. *Role.* [Online] [Zitat vom: 3. Juni 2008.] http://www.w3.org/TR/2007/REC soap12 part1 20070427.

[Sommerville, Ian 2001] *Softwareengineering.* München : Pearson Studium, 2001. ISBN 3 8273 7001 9.

[SunDrv] SUN Microsystems. *JDBC Data Access API.* [Online] [Zitat vom: 29. Juli 2008.] http://developers.sun.com/product/jdbc/drivers.

[SunJDBC] Sun. *Java Database Connectivity API.* [Online] [Zitat vom: 29. Juli 2008.] http://java.sun.com/products/jdbc/overview.html.

[SunPer] SUN. *Permissions in Java.* [Online] [Zitat vom: 18. Juni 2008.] http://java.sun.com/javase/6/docs/technotes/guides/security/permissions.html.

[SunRMIAPI] Sun. *Remote Methode InvocationAPI.* [Online] [Zitat vom: 17. Juni 2008.] http://java.sun.com/j2se/1.4.2/docs/api/java/rmi/package summary.html.

[SunRMIa] Sun Microsystems. *Remote Methode Invocation.* [Online]
[Zitat vom: 05. Juni 2008.]
http://java.sun.com/developer/onlineTraining/rmi/RMI.html.

[SunRMIb] Sun Microsystems. *Remote Methode Invocation.* [Online] [Zitat vom: 5. Juni 2008.] http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp.

[Tanenbaum, Andrew S. und Steen, van Maarten 2008] *Verteilte Systeme.* München : Pearson Studium, 2008. ISBN 978 3 8273 7293 2.

[Ullenboom, Christian 2008] *Java ist auch eine Insel.* Bonn : Galileo Press GmbH, 2008. ISBN 978 3 8362 1146 8.

[WikiDepl] Wikipedia. *Softwareverteilung.* [Online] [Zitat vom: 24. Juni 2008.]
http://de.wikipedia.org/wiki/Softwareverteilung.

[WikiLDAP] Wikipedia. *Lightweight Directory Access Protocol.* [Online] [Zitat vom: 1. Juli 2008.] http://de.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol.

[WikiMITM] Wikipedia. *Man In The Middle Angriff.* [Online] Wikipedia.
[Zitat vom: 28. Mai 2008.] http://de.wikipedia.org/wiki/Man In The Middle Angriff.

[WikiSOAP] Wikipedia. *SOAP.* [Online] [Zitat vom: 03. Juni 2008.]
http://de.wikipedia.org/wiki/SOAP.

[WikiW3C] Wikipedia. *World Wide Web Consortium.* [Online] [Zitat vom: 29. Juli 2008.]
http://de.wikipedia.org/wiki/World_Wide_Web_Consortium.

[WSPOL] W3C. *Web Services Policy 1.2.* [Online] [Zitat vom: 29. Juli 2008.]
http://www.w3.org/Submission/WS Policy/.

[XmlEnc] W3C. *XML Encryption Syntax and Processing.* [Online]
[Zitat vom: 29. Juli 2008.] http://www.w3.org/TR/xmlenc core/.

Abkürzungsverzeichnis

ACID	Atomicity, Consistency, Isolation, Durability
ALB	Automatisiertes Liegenschaftsbuch
ALK	Automatisierte Liegenschaftskarte
ALKIS	Amtliches Liegenschaftskataster Informationssystem
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Noation One
BNetzA	Bundesnetzagentur
CA	Certification Authorities
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
DNS	Domain Name System
DVZ M-V GmbH	DVZ Datenverarbeitungszentrum Mecklenburg Vorpommern GmbH
EDBS	Einheitliche Datenbankschnittstelle
GIOP	General Inter ORB Protocol
GISAL	Graphikintegriertes Informations System der Automatisierten Lie genschaften
GPRS	General Packet Radio Service
GUI	Graphical User Interface
GZIP	GNU zip
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IIOP	Internet Inter ORB Protocol
IP	Internet Protocol
JDBC	Java Database Connectivity

JDK	Java Development Kits
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JRE	Java Runtime Environment
JRMP	Java Remote Method Protocol
JSSE	Java Secure Socket Extension
JVM	Java Virtual Machine
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LiKatAVO	Liegenschaftskataster Abrufverordnung
MD5	Message Digest algorithm 5
MDB	Microsoft Access Database
M-V	Mecklenburg Vorpommern
NAS	Normbasierte Austauschschnittstelle
NAT	Network Address Translation
ODBC	Open Database Connectivity
OMG	Object Management Group
ORB	Object Request Broker
PDF	Portable Document Format
RAID	Redundant Array of Independent Disks
RMI	Remote Method Invocation
RootPOA	Root Program Object Adapter
RPC	Remote Procedure Call
RSS	Really Simple Syndication (seit Version 2.0)
SHA-1	Secure Hash Algorithm One
SOA	Service Oriented Architecture
SOAP	ursprünglich: Simple Object Access Protocol (wird jedoch offiziell seit Version 1.2 nicht mehr als Akronym gebraucht)
SQL	Structured Query Language

SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UAC	User Account Control
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URN	Uniform Resource Name
UUID	Universally Unique Identifier
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WCAG	Web Content Accessibility Guidelines
WLDG(E)	Workdatei Liegenschaftsbuch Daten-Gewinnung Entschlüsselt
XML	Extensible Markup Language