Ludwig
Maximilians
Universität
München

LMU

DOKUMENTATION

Implementierung einer WWW-Oberfläche für das digitale Zeichnungsarchiv der BMW AG

Fortgeschrittenenpraktikum

November 1996 - Februar 1997

Bearbeitet von:

Florian Pahl, Sven Meßfeldt

Aufgabensteller:

Prof. Dr. Heinz-Gerd Hegering

Betreuer:

Alexander Keller

Kurzbeschreibung

Der hohe Grad an Plattformunabhängigkeit des World Wide Web bietet gewichtige Vorteile bei der unternehmensweiten Bereitstellung von Dokumenten und Grafiken.

Damit verbunden ist die Fragestellung, wie sehr große Grafikdateien auf effiziente Weise über das WWW zugänglich gemacht werden können. Ein erster Lösungsansatz hierzu sind Skripten, die über das Common Gateway Interface (CGI) auf die gewünschten Dateien zugreifen.

Im Digitalen Zeichnungsarchiv (DZA) der BMW AG sind Konstruktionszeichnungen in einer Datenbank auf Dateisystemebene abgelegt und können momentan nur über eine RPC-Schnittstelle abgerufen werden. Aufgrund der großen Zahl an vorhandenen Zeichnungen ist es wünschenswert, diese mit Hilfe eines werksinternen WWW-Servers einem eingeschränkten Benutzerkreis zugänglich zu machen. Es sind daher verschiedene Alternativen zur Bereitstellung der Zeichnungen über das WWW zu bewerten. Anschließend ist die zweckmäßigste Alternative prototypisch zu implementieren.

Zur Implementierung soll die Programmiersprache Java verwendet werden.

Inhaltsverzeichnis

KAPITEL 1 Einführung

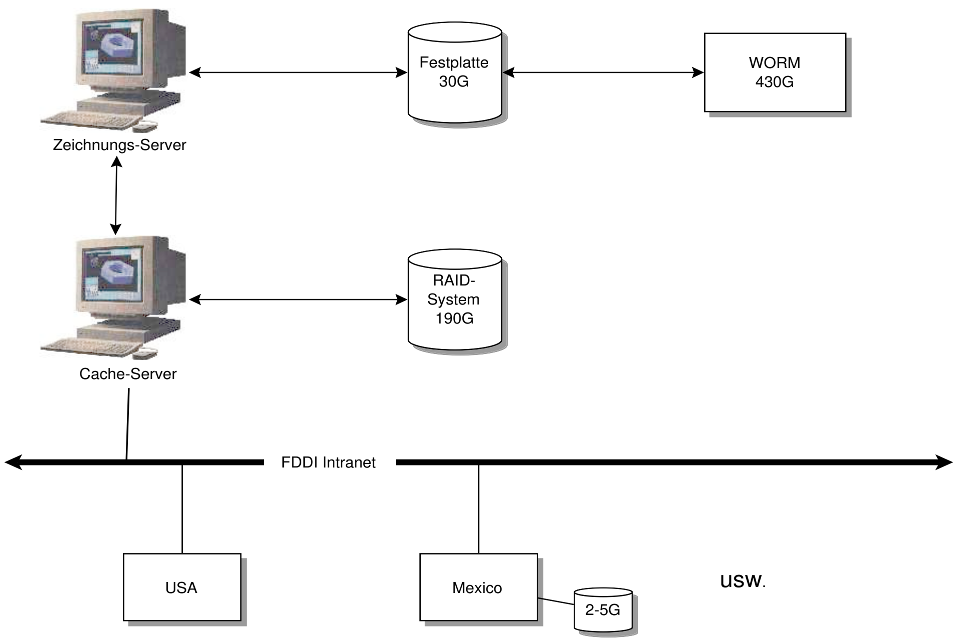
1.1 Überblick

In ihrem Digitalen Zeichnungsarchiv (DZA) speichert BMW alle Arten von Konstruktionszeichnungen. Hierzu gehören sowohl solche für gefertigte Produkte (z.B Aufbau der Fahrertür des 3ers), als auch Konstruktionszeichnungen für alle Fertigungsanlagen. Früher wurden alle diese Informationen mit Hilfe von Microfiche-Karten archiviert. Hierbei war der Hauptsortierschlüssel das Archivierungsdatum. Als von diesem Archivierungssystem auf ein elektronisches umgestellt werden sollte, begann man diese Microfiche-Karten einzuscannen. Bei der Wahl der Speicherungsstruktur für die so erhaltenen Dateien entschied man sich nicht für den Einsatz einer relationalen Datenbank, sondern dafür, die Dateien in einem Filesystem zu archivieren. Ein Teil dieser Dateien kann in einem Cache gehalten werden, der Großteil der Dateien wird aber mit Hilfe eines WORM-Systems archiviert.

Alle Zeichnungsdateien sind in einem speziellen Grafikformat (tiled tiff) gespeichert. Um diese anzeigen zu können, existierte bereits zur damaligen Zeit ein Viewer für MS Windows und UNIX Betriebssysteme, den die in München ansässige Firma MOSS für BMW programmiert hatte.

1.2 Speicherung und Zugriff auf die Zeichnungsdateien

Für die Speicherung und den Zugriff auf die Zeichnungsdateien kommt bei BMW folgende Rechnerstruktur zum Einsatz:



Als zentraler Cache-Server für die Zeichnungen steht eine Sparc-Station 1000 zur Verfügung. Sie verfügt über ein 190 Gigabyte großes RAID-Plattensystem, von dem 44 Gigabyte als Cache für die Zeichnungen genutzt werden können. Die zentrale Speicherung der Zeichnungen erfolgt auf einem 430 Gigabyte großem WORM-System, auf das über eine Sparc-Station 10 zugegriffen werden kann. Zusätzlich zu diesem zentralen Cache-System existieren noch weitere 13 Cache-Server in Werken wie z.B. in der USA oder in Mexiko. Diese verfügen allerdings über viel kleinere Kapazitäten. Sie sind alle mittels des BMW Intranets verbunden.

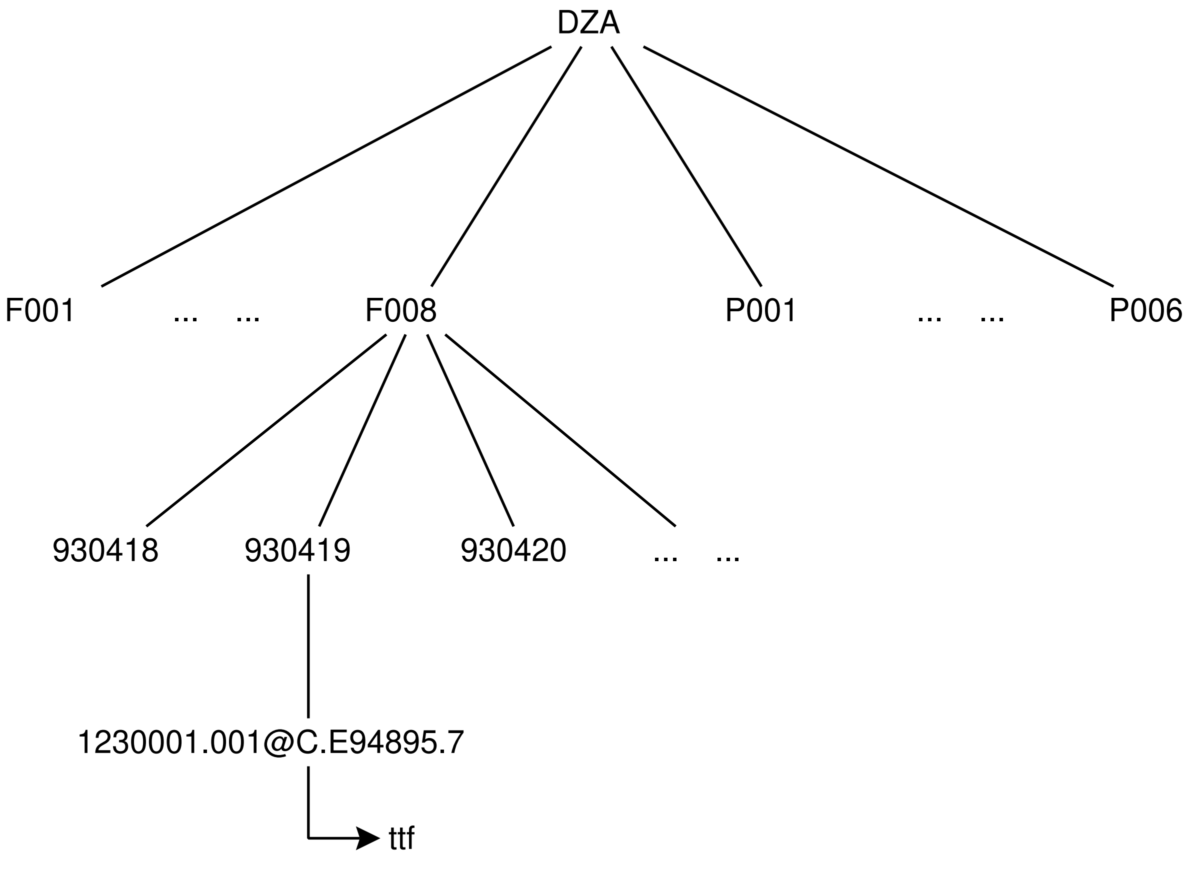
Wird nun eine Zeichnungsdatei angefordert, die nicht im Cache existiert, so wird diese mit einem RPC ähnlichen Aufruf (OFW) aus dem WORM-Archiv ausgelesen und in den Cache geladen. Dieser Mechanismus erfolgt für den Benutzer automatisch und mußte somit auch bei unserer Arbeit nicht berücksichtigt werden. Auf dem Cache-Server läuft ein Dämonenprozeß, der bei Platzmangel die Zeichnungen löscht, die am längsten nicht benutzt wurden.

Auf den zentralen Cache-Server erfolgen zur Zeit 2500-3000 Zugriffe am Tag.

1.3 Aufbau des Filesystems

Das Filesystem enthielt im August 1996 ungefähr 1,1 Millionen Zeichnungsdateien, die ca. 400 Gigabyte Speicherplatz benötigen. Da immer noch alte Microfiche-Karten eingescannt wurden, erweiterte sich die Zahl der zu verwaltetenden Zeichnungsdateien zu dieser Zeit täglich um weitere 150-200 Stück. Diese Zeichnungen sind im Durchschnitt 280 kB groß, können aber bis zu 3MB groß werden. Viele dieser Zeichnungen sind ausgedruckt im A0 oder A1 Format.

Das Filesystem hat folgenden Aufbau:



Das „DZA“-Hauptverzeichnis ist in Verzeichnisse für Produkte (P001 bis P006) und solche für Fertigungsmittel (F001 bis F008) unterteilt. Diese Directories sind dann wieder in Verzeichnisse unterteilt, deren Namen das jeweilige Einspieldatum wiederspiegelt (z.B 951110). Hier wurden nun Verzeichnisse erstellt, deren Namen eindeutig einem Zeichnungsobjekt zugeordnet werden kann. Dieser besteht aus einer Zeichnungsnummer, gefolgt von einer Blattnummer, einem Indexwert und weiteren Angaben über den Aufbau der Zeichnung. In diesen Verzeichnissen liegt nun die entsprechende Zeichnung mit dem Namen „ttf“ und eventuell noch ein weiteres File „ttt.inf“, das Informationen über die Zeichnung enthält. Leider ist dieses File nicht sehr oft vorhanden und enthält keine Angaben über Titel oder Inhalt der Zeichnung. Es beinhaltet nur Angaben über das Zeichnungsformat (z.B. Höhe und Breite), die auf anderem Weg auch leicht abgefragt werden können.

Um effizient auf die Dateien zugreifen zu können, gibt es zwei C-Programme. Das Programm „dza_rechserver“ erstellt eine Indexdatei der vorhandenen Zeichnungen. Mit dem Programm „dza_rechclient“ kann für eine als Parameter angegebene Nummer der entsprechende Pfad der gesuchten Datei aus dem Indexfile herausgesucht werden. Ist die Nummer unvollständig, so werden alle Zeichnungen ausgegeben, deren Nummer mit dem angegebenen Wert beginnt.

1.4 Anforderungen

Zugriff über WWW

Um die Konstruktionszeichnungen leichter verfügbar zu machen, sollte es nun unsere Aufgabe sein, diese mittels des World Wide Webs und einem herkömmlichen Internetbrowser zugänglich zu machen. Somit sollte es auch den Händlern in den Filialen, die nicht im BMW-Intranet integriert sind, sondern nur über einen Modem-Anschluß verfügen, ermöglicht werden sich die Konstruktionszeichnungen anzusehen. Bei einer solchen Lösung sollte eine weitgehende Plattformunabhängigkeit gewährleistet sein. Auch würde bei einem derartigen Zugriff die Installation des bisher verwendeten Viewers auf der lokalen Festplatte des jeweiligen Händlers entfallen.

Funktionen

Für die Bedienung auf Client-Seite wurde beschlossen sich, soweit möglich bzw. sinnvoll, an den Funktionen des bereits existierenden MOSS-Viewers zu orientieren. Somit ergaben sich folgende weitere Anforderungen:

- Graphische Benutzeroberfläche für alle Funktionen
- Der Benutzer kann mit verschiedenen Suchkriterien nach einer Zeichnung suchen. Er kann eine unvollständige Zeichnungsnummer angeben und erhält hierauf vom Server eine Liste aller zur angegebenen Nummer korrespondierenden Zeichnungsnummern. Diese Suche kann er weiter einschränken, indem er noch Angaben wie die Seitennummer oder den Stand der Zeichnung angibt.
- Hat der Benutzer so eine bestimmte Zeichnungsnummer gefunden, so kann er sich zu dieser zunächst eine verkleinerte Übersichtszeichnung anzeigen lassen.
- Aus dieser Übersichtszeichnung können sich beliebige Ausschnitte vergrößert darstellen lassen.
- Der MOSS-Viewer bietet die Möglichkeit, die oft sehr großen Zeichnungen auf großen Plottern etc. auszudrucken. Für unsere Aufgabe würde es aber ausreichen, wenn nur Standarddrucker unterstützt werden würden, da die Händler in den Filialen auch meistens nur über solche verfügen.

Security

In Bezug auf Sicherheit sollte eine globale Login-Funktion für alle potentiellen Nutzer des DZAs zur Verfügung stehen. Außerdem wäre es noch wünschenswert, bestimmte Zeichnungen, die nicht von jedem Mitarbeiter gesehen werden sollen, verschlüsseln zu können. Beide Vorkehrungen erfordern natürlich einen gewissen dauerhaften Verwaltungsaufwand, der nicht in unseren Aufgabenbereich hineinfällt. Für eine Lösung dieses Problems hatte BMW bisher jedoch keine Vorschläge. Vor allem im Bezug auf die Verschlüsselung scheint dies ein nur schwer lösbares Problem zu sein, da es bei BMW bereits Versuche hierfür gab, die allerdings fehlschlugen.

Da dezeit kein Konzept über Einführung und Verwaltung von Zugriffskontrolllisten und Schlüsseln existiert, konnte für dieses Problem auch keine Lösung erarbeitet werden.

Server

Als Web-Server sollte die vorher erwähnte Sparc-Station 1000 verwendet werden, auf der der Netscape Enterprise Server zum Einsatz kommen sollte. Wie vorher bereits erwähnt müssen wir uns nicht um die Bereitstellung der Zeichnungen aus dem Archiv kümmern, sondern können direkt auf das Filesystem zugreifen. Um die Zeichnungen zu finden sollten, wenn irgendwie möglich die vorhin erwähnten Programme „dza_rechserver“ und „dza_rechclient“ benützt werden. Da viele Benutzer auf das DZA zugreifen, ist es auch wichtig eine möglichst performante Lösung zur Verfügung zu stellen.

Einschränkung

Als Einschränkung des Themas sollte es nicht zwingend zu unserer Aufgabe gehören einen neuen Viewer zum Anzeigen der tiled tiff Dateien zu programmieren. Hierfür könnte der bereits existierende Viewer als Netscape „Helper-Application“, oder irgendein anderes Netscape Plug-In verwendet werden.

Außerdem wurden uns keine Einschränkungen bei der Wahl der Programmiersprache etc. von BMW vorgegeben.

KAPITEL 2

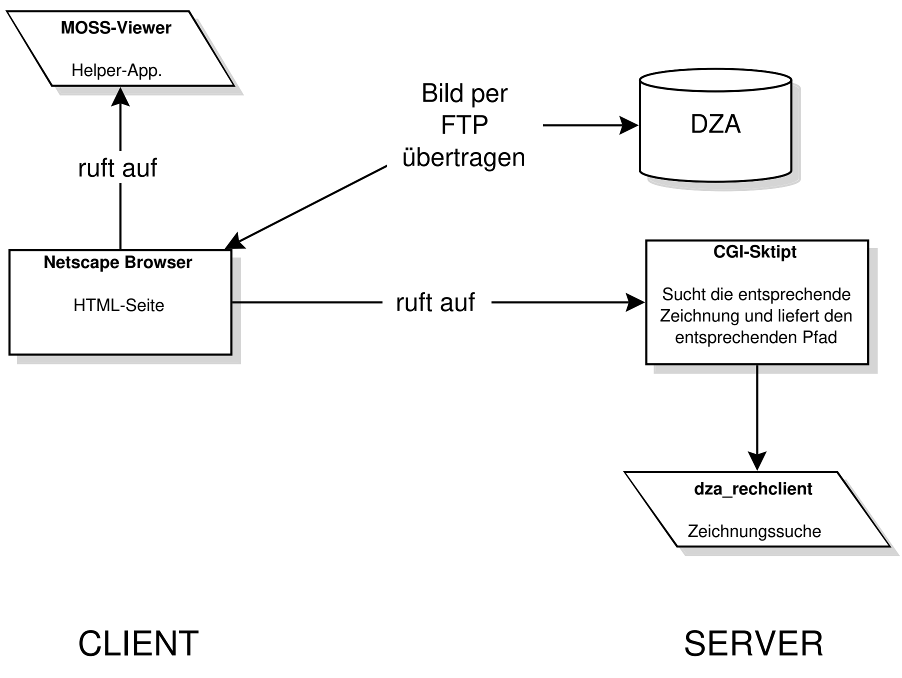
Lösungsmöglichkeiten

2.1 Lösung mit Zugriff über CGI

Als herkömmliche Lösung der Augabenstellung hätte sich eine Implementierung mit Zugriff über das CGI (Common Gateway Interface) angeboten.

Eine solche Lösung hätte prinzipiell folgenden Aufbau:

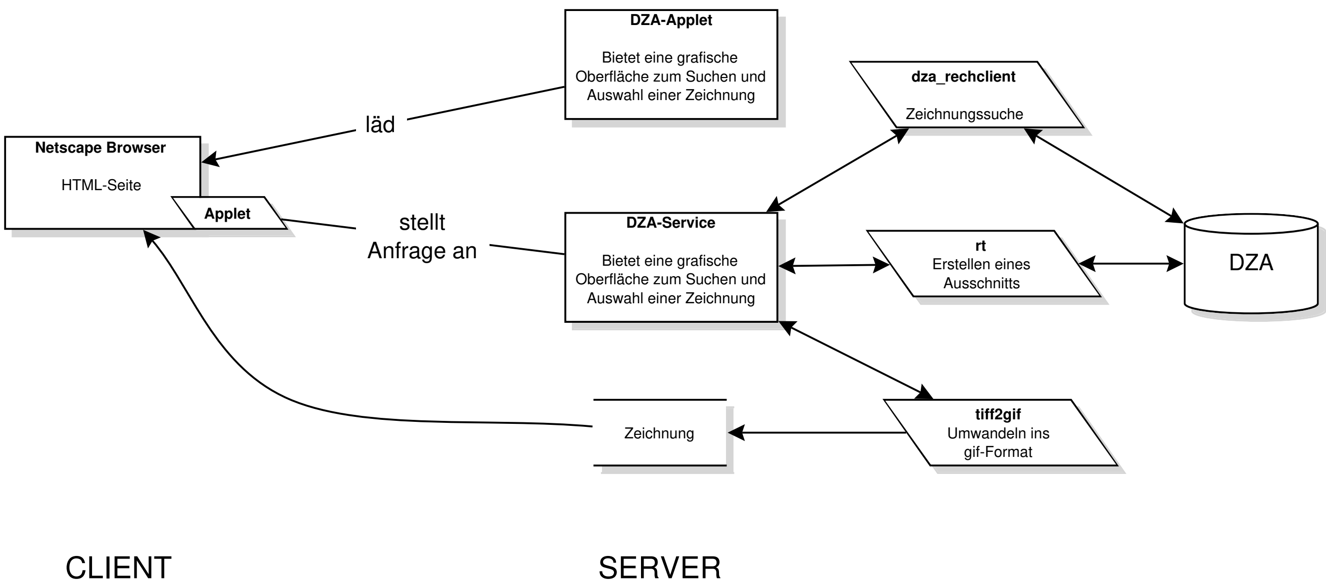
- Auf Clientseite könnte man mit einem Javascript-fähigen Internetbrowser (Netscape 3.x oder Microsoft Internet Explorer 3.x) die graphische Benutzeroberfläche mittels HTML-Forms und die nötigen Abfragen mit Javascript-Programmen realisieren.
- Die so eingegebenen Daten zur Suche einer Zeichnung würden dann mittels HTTP an Port 80 des Servers geschickt werden. Ein oder mehrere CGI-Skripten (voraussichtlich Perl-Programme) würden dann die Daten auswerten und das „dza_rechclient“-Programm aufrufen.
- Die so erhaltenen Informationen über die gesuchte(n) Zeichnung(en), wie z.B. den Pfad im Filesystem, würden dann an den Client zurückgeschickt werden.
- Dieser könnte dann die entsprechende Datei mittels ftp vom entsprechenden Pfad auf dem Server herunterladen.
- Um die Datei anzuzeigen, müßte der Benutzer den MOSS-Viewer lokal installieren und in seinem Internetbrowser als „Helper-Application“ eintragen.
- Mit Hilfe dessen könnte er dann die heruntergeladene Datei anzeigen und alle weiteren Funktionen des Viewers benutzen.
- Als Möglichkeit zur Authentifikation am Server würde es sich anbieten, die, vom bei BMW verwendeten Netscape Enterprise Server gegebene, User-Verwaltung zu benutzen.



2.2 Lösung mit Java

Als Alternative ist aber auch eine Lösung unter Verwendung von Java möglich. Eine solche würde aus einem Java-Applet auf Client-Seite bestehen, das mittels einer Java-Socket-Connection mit einem beliebigem Port des Servers kommuniziert.

- Der Benutzer würde mit einem java-fähigen Internetbrowser die entsprechende HTML-Seite aufrufen, die das Java-Applet enthält. Dieses würde eine unter Verwendung der Java-GUI-Klassen programmierte Oberfläche zur Verfügung stellen.
- Hat der Benutzer seine Angaben für die Zeichnungssuche gemacht, baut der Client mittels Java eine Socket-Verbindung zum Server auf, über die im folgenden die Kommunikation mit dem Server abläuft.
- Auf Serverseite würden ein oder mehrere Threads den entsprechenden Port überwachen und für jede zu bearbeitende Anfrage einen neuen Thread erzeugen.
- Ein Thread würde das „dza_rechclient“-Programm aufrufen, um unter anderem den Pfad für die gesuchte Zeichnung zu ermitteln. Die erhaltenen Informationen würden wieder über die Socket-Verbindung an den Client zurückgeschickt.
- Will der Client eine neue Zeichnung auswählen, so würde ein neuer Thread auf dem Server erzeugt, der eine verkleinerte Übersichtszeichnung bereitstellt, die der Client in einem neuen Java-Fenster angezeigt bekommt.
- Aus diesem Fenster kann der Client jetzt beliebige Bereiche vergrößern. Hat der Server diesen Ausschnitt berechnet, so wird dieser vom Client in einem neuen Netscape-Fenster angezeigt.
- Das Drucken und Speichern dieser Ausschnitte wäre mit den vom Internetbrowser gegebenen Funktionen möglich.
- Die Authentifikation am Server würde sich auch hier mit Hilfe des Netscape-Servers anbieten.



2.3 Vergleich

Die implementierte Lösung mit Java hat im Vergleich zu einer Lösung mit CGI-Programmierung folgende Vor-und Nachteile:

Vorteile:

- Es wird kein Speicherplatz auf der lokalen Festplatte des Clients benötigt, da sowohl die Installation des MOSS-Viewers entfällt, als auch das Speichern der Zeichnungen.
- Einfacher zu nutzen, da kein Installationsaufwand für Viewer.
- Bessere Benutzeroberfläche, da mit den Java-GUI-Klassen mehr Möglichkeiten zur Gestaltung dieser zur Verfügung stehen, als mit HTML-Forms.
- Schnelleres Anzeigen der Zeichnungen, da nie eine ganze Zeichnung (bis 3MB groß) geladen werden muß, sondern nur immer Teile davon bzw. eine verkleinerte Übersicht (<20kB).
- Schnellere Anfrageverarbeitung auf Serverseite, da kein Stau am Common Gateway Interface entsteht. Die Anfragen werden an verschiedenen Ports von verschiedenen Threads verarbeitet.

Nachteile:

- Höherer Aufwand bei der Implementation, da es einerseits für CGI-Programmierung schon viele Beispiele gäbe, und andererseits auch noch ein eigener Viewer programmiert werden muß.
- Auf Serverseite wird zusätzlicher Speicherplatz für die berechneten Ausschnitte benötigt. Dieser dürfte aber, bei regelmäßigem Löschen dieser temporären Daten, auf keinen Fall mehr als 1GB betragen. (bei einer Ausschnitt-Dateigröße von ca. 10kB, ca. 1000 Benutzern pro Tag, 100 Zugriffen pro Benutzer und täglichem Löschen der temporären Dateien)
- Drucken nur mit Netscape-Mitteln (entspricht aber den Anforderungen)
- Auf Clientseite kann der Internet Explorer 3.x nicht mehr benutzt werden, sonder nur noch Netscape 3.x. Diese Einschränkung resultiert daraus, daß eine Verbindung zwischen Java und Javascript über Netscape LiveConnect benötigt wird, die der Internet Explorer nicht unterstützt. Allerdings kommt bei BMW standardmäßig auch der Netscape Browser zum Einsatz.

Aufgrund dieser Überlegungen beschlossen wir, die Implementation mit Java vorzunehmen.

KAPITEL 3

Umsetzung

3.1 Übersicht

3.1.1 Speicherungsformat der Konstruktionszeichnungen

Alle Konstruktionszeichnungen sind bei BMW als Multiple Image Tiled Tiff Files gespeichert. Dieses spezielle Grafikformat hat im konkreten Fall einer bei BMW vorliegenden Datei folgenden Aufbau:

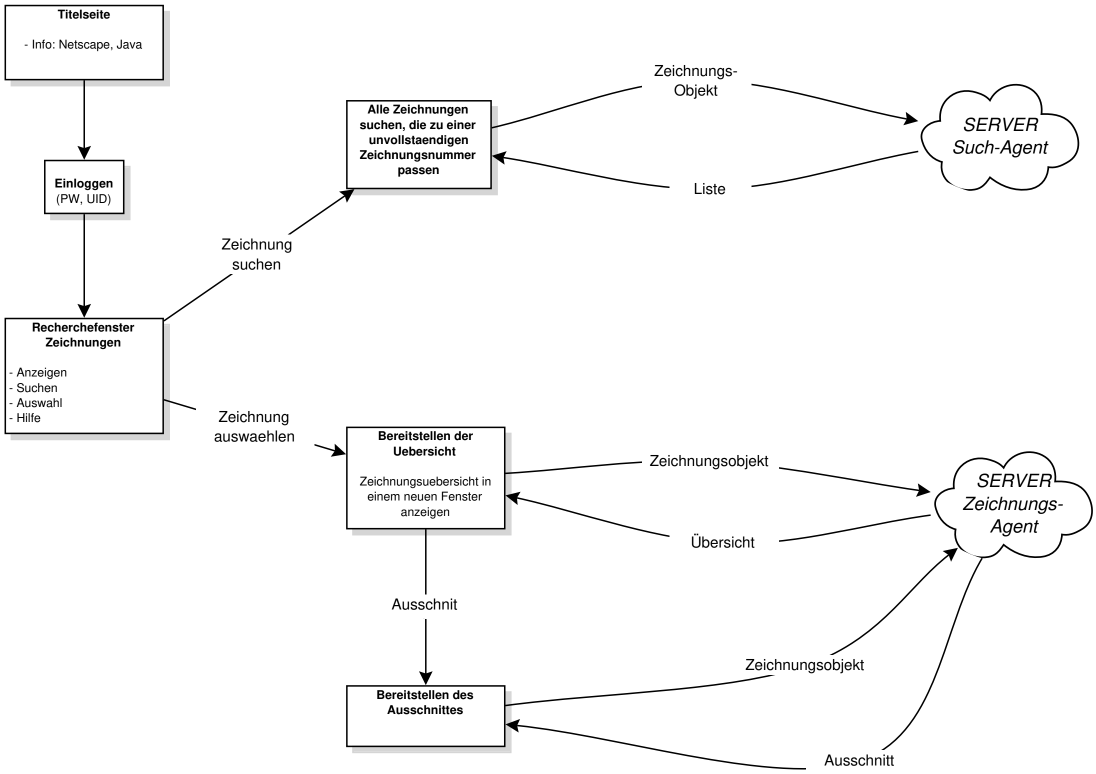
Ein Multiple Image Tiled Tiff File kann mehrere Bilder enthalten. Hierbei kann es sich entweder um verschiedene Seiten handeln oder um verschieden große Ansichten einer bestimmten Konstruktionszeichnung. Auf jeden Fall enthält jedes File immer die Komplettansicht der jeweiligen Zeichnung. Jedes dieser Bilder kann wieder in Kacheln gleicher Größe unterteilt sein. Im Fall von BMW ist jede dieser Kacheln 512x512 Pixel groß.

Leider unterstützen weder Java noch der Netscape-Browser dieses Bildformat. Da wir wegen der vorher aufgelisteten Nachteile auch nicht den MOSS-Viewer verwenden wollten, mußte entweder ein eigener Viewer für das tiled tiff Format geschrieben werden, oder eine andere Lösung gefunden werden.

Nach unseren Überlegungen dürfte es nicht möglich sein, einen Viewer in Java für das tiled tiff Format zu schreiben, da dies wahrscheinlich an der Geschwindigkeit der interpretierten Sprache scheitern würde. Diese Vermutung wurde auch durch die Aussage von Herrn Baer von der Münchner Softwarefirma MOSS bekräftigt. Dieser hat den bei BMW verwendeten Viewer programmiert und verfügt somit über genaue Kenntnisse über die Komplexität des Bildformats.

Daher mußte das Anzeigen von Bilder im tiled tiff Format umgangen werden. Die im folgenden beschriebene Lösung konvertiert die benötigten Zeichnungen, bzw. Ausschnitte davon, auf Serverseite in das gif-Graphikformat. Graphiken in diesem Format können dann auf der Clientseite leicht vom Java-Applet, oder vom Netscape-Browser angezeigt werden.

3.2 Client-Seite



Auf Seite des Clients ist es notwendig Netscape 3.x als Browser zu verwenden, da nur hiermit die später erklärte Kommunikation zwischen Java und Javascript möglich ist.

Wählt ein Benutzer die URL des BMW-Server für das DZA an, dann erscheint zunächst eine Einstiegsseite mit einer allgemeinen Information. Wählt er hier den Hypertext-Link an, der auf die Seite für die Zeichnungrecherche verweist, so erfolgt zunächst die Authentifizierung am Netscape-Server. Nachdem der Benutzer hier ein gültiges Login und Passwort eingegeben hat, lädt der Netscape Browser die HTML-Seite für die Zeichnungsrecherche. Ab diesem Zeitpunkt ist der Benutzer am Server authentifiziert und hat Zugriff auf die dort benötigten Verzeichnisse. Auf der aufgerufenen Seite wird zunächst das Java Applet geladen, das dann die weitere Zeichnungsauswahl und das Anzeigen der Zeichnungen ermöglicht.

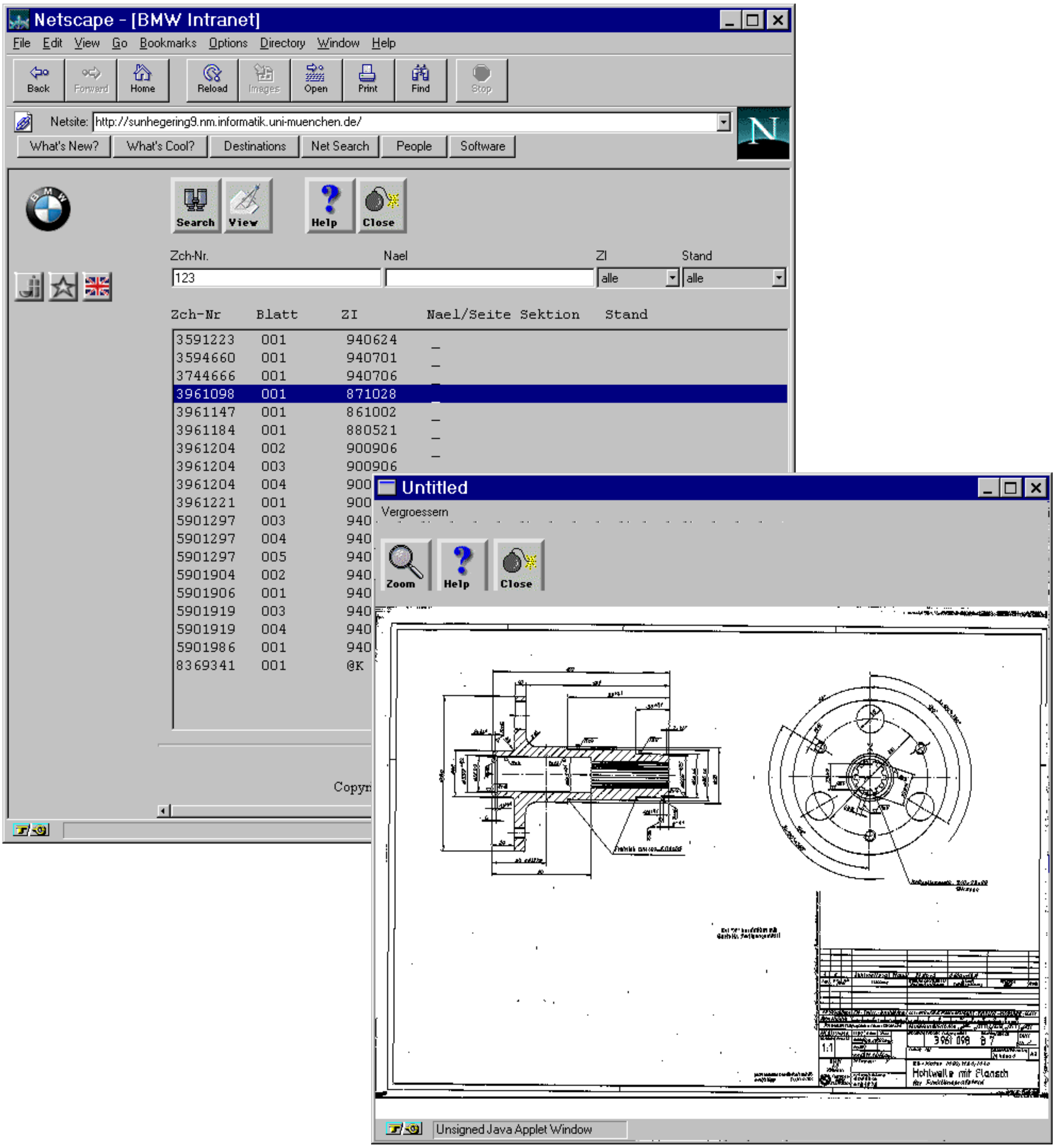
3.2.1 Graphische Benutzeroberfläche

Dieses Java-Applet stellt dem Benutzer im aktuellen Browserfenster eine graphische Benutzeroberfläche zur Verfügung. Beim Aufbau der Oberfläche wurde versucht sich, soweit möglich, an der des MOSS-Viewers zu orientieren.

Der Bildschirmaufbau enthält folgende Komponenten:

- vier Imagebuttons
  - **(i)** „Search"-Button: zum Suchen nach einer oder mehreren Zeichnungen
  - **(ii)** „View"-Button: zum Anzeigen einer Übersichtszeichnung für die ausgewählte Zeichnungsnummer
  - **(iii)** „Help"-Button: zum Anzeigen einer Hilfefunktion
  - **(iv)** „Close"-Button: zum schließen der Java-Fenster
- zwei Textfelder zum Eingeben einer Zeichnungsnummer und einer Seitenzahl

- zwei Auswahlmenüs zum Einstellen des Zeichnungsindexes und des Stands der Zeichnung
- eine Listbox zum Anzeigen einer Liste von Zeichnungen



3.2.2 Suchen einer Zeichnung

„Search"-Button

Um nach einer Zeichnung zu suchen, kann der Benutzer im dafür vorgesehenen Textfeld eine, auch unvollständige, Zeichnungsnummer angeben. Diese wurde auf maximal sieben Stellen beschränkt. Auch müssen mindestens drei Stellen angegeben werden, da eine folgende Suche sonst evtl. zu viele Ergebnisse liefern würde. Weitere Angaben kann er im Textfeld „NAEL/ Seite“ und in den Auswahlmenüs „ZI“ und „Stand“ angeben.

Drückt er auf den „Search“-Button, so wird zunächst ein neues Zeichnungsobjekt erzeugt, in welches die vorher spezifizierten Angaben übernommen werden. Hierauf baut der Client eine Verbindung zu einem bestimmten Port des Servers auf. Mittels dieser schickt er dem Server dieses Zeichnungsobjekt. Falls auf dem Server genau ein Zeichnungsnummer mit den Angaben korrespondiert, erhält er genau eine Zeichnungsobjekt vom Server. Wurde eine unvollständige und nicht eindeutige Nummer angegeben, so erhält er eine Liste von Zeichnungsobjekten, deren Zeichnungsnummer mit der angegebenen beginnen. Ein so erhaltenes Zeichnungsobjekt enthält alle auf dem Server zur gewünschten Zeichnung verfügbaren Angaben, wie Seitenanzahl, Index und Größe. Die erhaltenen Informationen werden zeilenweise in der dafür vorgesehenen Listbox ausgegeben.

Bei den Socket-Verbindungen zum Server handelt es sich um datagramm-orientierte Verbindungen. Diese kann keine ganzen Objekte übertragen. Deswegen mußten Methoden geschrieben werden, die ein zu übertragendes Zeichnungobjekt in einen String umwandeln bzw. aus einem String ein Zeichnungsobjekt erzeugen. Hierzu wurden die verschiedenen Variablen eines Zeichnungsobjektes mit einem Trennzeichen zu einem einzigen String konkateniert Dieser wurde dann mit einem String abgeschlossen, der dem Empfänger signalisiert, daß er alle Informationen über das jeweilige Zeichnungsobjekt erhalten hat. Da der Empfänger das Trennzeichnen und das abschließende Zeichnen kennt, kann er dann aus dem String wieder das Zeichnungobjekt erzeugen.

3.2.3 Auswählen und Vergrößern einer Zeichnung

„View“-Button

Hat der Benutzer eine Zeichnung, aus der Liste der möglichen, in der Listbox ausgewählt, dann kann er die Grobansicht laden, indem er den „View“-Button anwählt. Nun schickt der Clientrechner auf einer neuen Client-Server-Socketverbindung das ausgewählte Zeichnungsobjekt an den Server. Als Antwort erhält er wieder ein Zeichnungsobjekt, das zusätzlich die URL gespeichert hat, an der die gewünschte Grobansicht liegt. Nun lädt das Java-Applet diese Übersichtszeichnung, und zeigt sie dann in einem neuen Frame an.

Das geöffnete Fenster enthält folgende Komponenten:

- ein Auswahlmenü: Hier kann man einstellen, ob man einen frei definierbaren Ausschnitt, oder eine Kachel fester Größe markieren will. Bei Kacheln fester Größe kann man entweder das ganze Bild anwählen, oder Ausschnitte, deren Kanten 1/2, 1/4, 1/8 oder 1/16 der Kantenlänge der Originalzeichnung lang sind.
- drei Imagebuttons: den „Zoom“-Button, um den markierten Bereich zu vergrößern und einen „Help“- und einen „Close“-Button.

„Zoom“-Button

Hat man zum Vergrößern „frei“ gewählt, dann kann man mit der Maus nun ein beliebiges Rechteck markieren. Bei einem anderen Vergrößerungsmodus wird immer das entsprechende Rechteck angewählt, in das der Benutzer mit der Maus klickt. Wählt man dann den „Zoom“-Button, so werden die Koordinaten des gewünschten Ausschnitts in die entsprechenden Felder des aktuellen Zeichnungsobjektes eingetragen. Dieses wird dann auf einer neuen Socket-Verbindung an den Server geschickt. Dieser berechnet den Ausschnitt und schickt das Zeichnungsobjekt wieder zurück. Dieses enthält jetzt den Pfad an dem der gewünschte Ausschnitt auf Serverseite abgelegt ist.

Nun wird der entsprechende Ausschnitt in einem neuen Netscape-Fenster angezeigt. Ein solcher Ausschnitt hat immer die Größe von 640x480 Pixeln. Wurde ein Ausschnitt markiert, der nicht diesem Größenverhältnis entspricht, so wurde vom Server an einer Seite des Ausschnitts ein weißer Rand erzeugt.

Aus folgendem Grund wird hier ein Netscape-Fenster und kein Java-Fenster verwendet:

Aus Sicherheitsgründen dürfen Java-Applets nicht auf lokale Ressourcen zugreifen. Hätte man die Übersicht in einem neuen Java-Fenster geöffnet, so wäre es nicht möglich, diese zu drucken und zu speichern. Deshalb übergibt das Java-Applet die URL, an welcher der Zeichnungsausschnitt auf dem Server liegt, an eine Javascript Funktion. Diese öffnet dann das Netscape-Fenster und lädt von der übergebenen URL den Zeichnungsausschnitt mittels HTTP. Um diesen Datenaustausch zu ermöglichen, mußten die LiveConnect-Klassen verwendet werden, die nur der Netscape-Browser beinhaltet. Aus diesem Grund läuft das Applet nur auf diesem Browser und nicht in Microsofts Internet Explorer, da letzterer keinen Datenaustausch zwischen Java-Applets und Javascript-Funktionen erlaubt.

Desweiteren bringt der Umweg über Javascript den Vorteil, daß die bereits geladenen Ausschnitte im lokalen Cache von Netscape gehalten werden, und somit schnell wieder darauf zugegriffen werden kann.

3.2.4 Weitere Funktionen

„Help“-Button

Falls das Programm nicht selbsterklärend sein sollte, so kann der Benutzer den „Help“-Button anwählen. Hierauf öffnet eine Javascript Funktion ein neues Netscape-Fenster und der Benutzer kann dort zu den verschiedenen Funktionen kurze Erklärungen abrufen. Um hier die Steuerung zu erleichtern gibt es drei verschiedene, mit Javascript-Funktionen belegte, Buttons. Mit diesen kann man durch die Hilfethemen vor, zurück und zur Einstiegsseite navigieren.

„Close“-Button

Der zweimal vorhandene „Close“-Button wurde eingeführt, um schnell die zusätzlich erzeugten Fenster zu schließen. Die wichtigste Funktion von diesem ist aber das Schließen des Java-Fensters mit der Übersichtzeichnung. Aufgrund eines Bugs in den GUI-Klassen des JDK 1.02 funktionieren bei von Java erzeugten Fenstern die vom Betriebssystem vorgegebenen Buttons in der Menüleiste zum schließen des Fensters nicht. Das Java-Fenster könnte also nur geschlossen werden, wenn man den ganzen Netscape-Browser beenden würde.

Drucken und Speichern

Wie schon vorher erwähnt, kann der Benutzer die ausgewählten Ausschnitte einer Zeichnung mit den im Netscape-Browser vorgegebenen Funktionen „Print ...“ drucken, oder mit „Save_As ...“ auf seine lokale Festplatte speichern.

Fehlerabfrage

Auf der Client-Seite wurde versucht so viele Fehler wie möglich abzufangen, oder dem Benutzer anzuzeigen. Falls ein Fehler nicht abgefangen werden konnte, öffnet sich eine Dialog-Box, die dem Benutzer eine kurze Mitteilung über die Art des Fehlers macht. Dies kann z.B. der Fall sein, wenn der Benutzer eine falsche Eingabe macht oder die Verbindung zum Server nicht hergestellt werden kann. Falls auf Server-Seite ein Problem auftritt, so übergibt dieser im aktuellen Zeichnungsobjekt eine Fehlermeldung, die der Client dann wieder in einer Dialog-Box ausgibt.

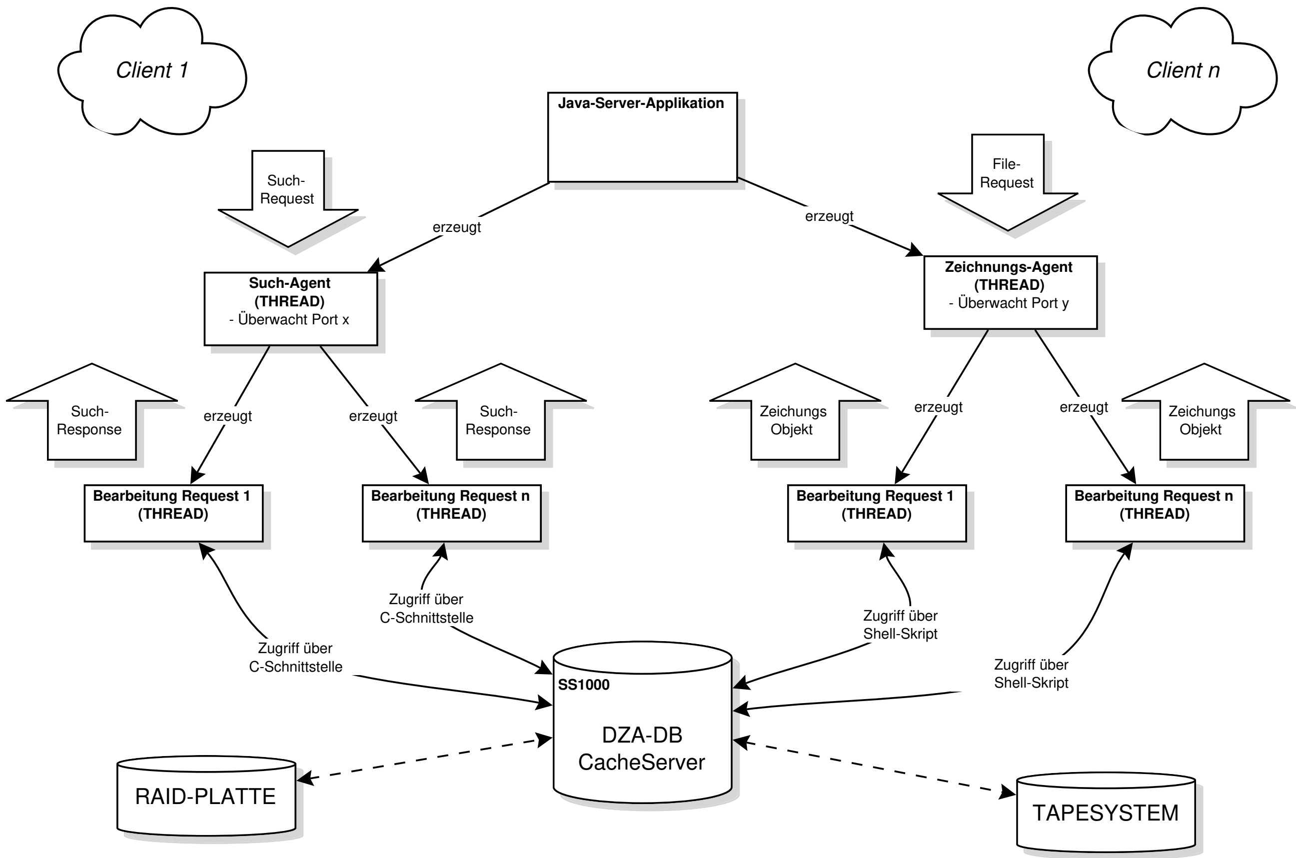
3.3 Server-Seite

Der Server ist ein Java-Programm, welches zwei Dienste zur Verfügung stellt:

3.3.1 Such-Agent

Er hat die Aufgabe, alle Zeichnungsnummern, die zu einem vorgegebenen Suchpräfix (z.B. 123) passen, herauszufinden. Darüberhinaus soll auch der Pfad, an dem die Zeichung zu finden ist, ermittelt werden.

Der Suchagent wartet am TCP-Port 6789 auf Anfragen. Verbindet sich ein Client mit dem Server, startet der Server einen Thread eigens für diese Anfrage. Der Hauptprozeß kann dann weitere Anfragen entgegennehmen.



Zunächst übergibt der Client dem Server ein Object der Klasse „DzaZeichnungsobjekt“. Dieses Objekt enthält folgende Instanz-Variablen:

String nummer;

String blatt;

String zi;

String nael;

String sektion;

String stand;

In „nummer“ steht die zu suchende Zeichnungsnummer. Der Server stößt jetzt den „dza_rechclient“ an, der das Suchergebnis ausgibt, welches beispielweise wie folgt aussieht:

/DZA/P003/940419/1230000.001.@@._._

/DZA/P004/941205/1230001.001.@C.E94895._

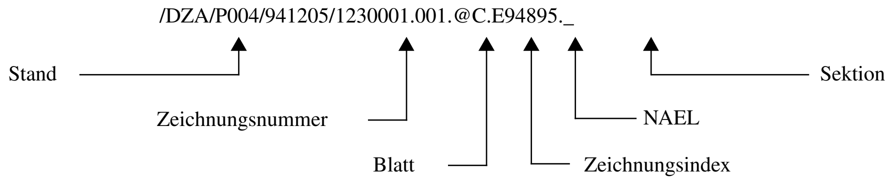
/DZA/P003/940419/1230003.001.@@._._

/DZA/P004/941205/1230006.001.@C.E9149B._

/DZA/P004/941205/1230007.001.@C._._

/DZA/P004/940428/1230008.001.@F._._

Dies sind Pfadnamen auf Verzeichnisse, in denen sich jewails das TTF-Image befindet. Es heißt in jedem Fall „ttf“. Dem Pfadnamen können jetzt die benötigten Informationen entnommen werden:



All diese Informationen können in Instanzvariablen, die das DzaZeichnungsobjekt zur Verfügung stellt, gespeichert werden. Alle so gewonnenen Zeichnungsobjekte werden dann in einem Objekt der Klasse „Vector“ abgelegt. Die Klasse Vector ist im JDK enthalten.

Abschließend wird der gesamte Vector an den Client übertragen und die Verbindung beendet.

Nachdem der Benutzer eine Zeichnung zur Ansicht ausgewählt hat, kann er sie sich anzeigen lassen, doch zuvor muß sie vom Zeichnungs-Agenten auf dem Server bereitgestellt werden.

3.3.2 Zeichnungs-Agent

Der Zeichnungs-Agent stellt eine Zeichnung aus dem DZA so zur Verfügung, daß sie direkt vom Browser heruntergeladen und angezeigt werden kann. Der Agent ist dabei so flexibel gehalten, daß nicht nur eine Gesamtübersicht angezeigt werden kann, sondern auch beliebige Ausschnitte.

Ebenso wie beim Such-Agenten wird für jeden “Request” ein neuer Thread erstellt, so daß parallele Anfragen möglich sind.

Zunächst wird dem Server ein Zeichnungsobjekt übergeben.

In diesem Zeichnungsobjekt befinden sich die durch den Such-Agenten ermittelten Informationen, wie physikalischer Verzeichnisname, Erstellungsdatum etc..., darüberhinaus aber auch den gewünschten Ausschnitt. Wenn kein Ausschnitt angegeben wird (z.B. 0,0,0,0), so wird eine Übersichtszeichnung bereitgestellt.

Im einzelnen sind folgende Schritte nötig:

Abfragen der Bildinformationen

Hierzu wird das von der Firma MOSS erstellt Programm ’TIFFTAGS’ verwendet, welches als Aufrufparameter lediglich den Bildnamen benötigt. Von TIFFTAGS wird dann insbesondere die Breite und die Höhe des Bildes ausgegeben und vom Server abgefragt.

Verkleinerung

Anschließend wird das Bild bzw. der Ausschnitt auf die Größe des Client-Fensters gebracht. Hierfür steht auch ein Programm der Firma MOSS zur Verfügung: das Programm für Rastertransformation ’RT’.

Umwandeln ins GIF-Format

Da das Programm RT nur Bilder im TIFF-Format ausgeben kann und der Netscape-Browser ohne Plugin keine TIFF-Bilder anzeigen kann, konvertiert der Server das extrahierte Bild noch ins gebräuchliche GIF-Format. Dazu wird das start geschwindigkeitsoptimiertes Programm ’TIF2GIF’ verwendet.

KAPITEL 4 Testphase und Installation

4.1 Systemvoraussetzungen

Bei der Entwicklung des Java-DZA-Applets stand am Institut für Informatik folgendes System zur Verfügung:

- Netscape Enterprise Server 2.0
- SUN SPARCstation 20, SOLARIS 2.5

Folgende Software-Komponenten sind für den Einsatz des DZA notwendig:

- SUN JDK 1.02
- Installiertes Konvertierungsprogramm TIF2GIF (ggf. muß es neu übersetzt werden)
- Installierte MOSS-Utilities: RT und TIFFTAGS
- BMW-DZA-Umgebung: dza_rechclient
- WWW-Server

4.2 Konfiguration

Um die Anpassungen an eine neue Umgebung möglichst gering zu halten, wurden alle globalen Einstellungen in der Java-Klasse DZA.java konzentriert, die anzupassen ist.

4.2.1 Such-Agent

Port-Nummer: `public final static int suchport = 6789;`

DZA-Suchroutine: `public final static String rechclient =`

`"/usr/local/ns-home/docs/dza/bin/dza_rechclient";`

4.2.2 Zeichnungs-Agent

Port-Nummer: `public final static int zeichnungsport = 6790;`

TIFFTAGS-Programm: `public final static String tifftags =`

`"/usr/local/ns-home/docs/dza/bin/tifftags";`

Transformationsprogramm: `public final static String rt =`

`"/usr/local/ns-home/docs/dza/bin/rt";`

Temporäres Verzeichnis, in dem die bereitgestellten Bilder abgelegt werden. Wichtig dabei ist nur, daß es ein für den WWW-Server ’öffentliches’ Verzeichnis ist.

`public final static String urlpath = "/usr/local/ns-home/docs/dza/tmp/";`

Relativer Pfad, der als URL benutzt werden kann:

`public final static String urlpathrel = "dza/tmp/";`

Konvertierungsprogramm: public final static String tiff2gif =
"/usr/local/ns-home/docs/dza/bin/tif2gif";

WWW-Server, DZA-Server: public final static String server =
"sunhegering9.nm.informatik.uni-muenchen.de";

WWW-Adresse des Servers: public final static String home =
"http://sunhegering9.nm.informatik.uni-muenchen.de/";

4.3 Compilieren

Wenn Änderungen an den Java-Quellen vorgenommen wurden oder die Konfigurationsdatei DZA.java verändert wurde, müssen sowohl der Client als auch der Server neu compiliert werden:

Client:

```
sunhegering9:/docs/dza> javac DzaMain.java
```

Server:

```
sunhegering9:/docs/dza/agent> javac Agent.java
```

4.4 Installation

Um das Applet zu starten, muß auf dem Server lediglich der Agent gestartet werden, der beide Server-Dienste zur Verfügung stellt. Zu diesem Zweck existiert im Verzeichnis des Agenten eine Skript-Datei. (Um dieses Skript zu benutzen, muß darin gegebenenfalls der DZA-Pfad angepaßt werden)

Aufruf:

```
sunhegering9:/docs/dza/agent> dza_service start
```

Um den Server jederzeit benutzen zu können, sollte der Agent durch den Unix-Cron-Mechanismus gestartet werden - auch hierfür kann das oben erwähnte Skript verwendet werden:

```
dza_service restart
```

Um regelmäßig die temporären Dateien zu löschen, empfielt es sich, zusätzlich folgendes Kommando in den Cron mit aufzunehmen:

```
dza_service cleanup
```

ANHANG A

Informationen über Java und Case-Tools

4.5 Test verschiedener GUI-Tools

Zu Beginn der Implementierung unseres Applets wollten wir uns die Arbeit durch den Einsatz eines der vielen zu dieser Zeit entstandenen Java-GUI-Tools, oder durch Einsatz einer kompletten Programmierumgebung erleichtern. Vielleicht können unsere, im folgenden zusammengefaßten Erfahrungen, jemandem, der in Zukunft an etwas ähnlichem arbeitet, helfen etwas Zeit zu sparen:

Wir probierten einige als Shareware erhältliche GUI-Tools aus. Mit diesen ordnet man direkt am Bildschirm die gewünschten GUI Komponenten an, und das Programm erstellt danach den dazugehörigen Sourcecode. Die meisten dieser Tools waren sehr einfach und schöpften oft nicht einmal die Möglichkeiten der standardmäßig in den java.awt.* Klassen enthaltenen Komponenten aus.

Am besten gefiel uns hier noch das Programm SpecJava. Dieses ist in Tcl geschrieben und war zunächst nur für Windows Betriebssysteme lauffähig. Theoretisch wäre es sicher möglich, dieses auch unter einem anderen Betriebssystem laufen zu lassen, da es sich bei Tcl ja um eine interpretierte Sprache handelt. Leider bietet das Tool keine Möglichkeit einen Event-Handler zu erstellen. Außerdem gelang es uns nicht immer alle Komponenten so genau zu plazieren, wie wir das wollten. Somit schien es leider für unsere Arbeit nicht besonders hilfreich zu sein.

Um mehr Unterstützung zu erhalten, als nur bei der Gestaltung der graphischen Oberfläche, hätte eine komplette Java-Entwicklungsumgebung unsere Arbeit erleichtern können. Hier bot sich natürlich der JavaWorkshop der Firma Sun an, da dieser unter vielen Betriebssystemen erhältlich ist. Von diesem ist auch eine Lizenz für Solaris-Betriebssysteme am Lehrstuhl vorhanden. Während der kurzen Zeit, in der wir mit dem Java Workshop 1.0 arbeiteten, konnten wir folgende Vor-und Nachteile feststellen:

Vorteile:

- einfache Bedienung, aufgrund einer browserähnlichen Oberfläche
- kurze Eingewöhnungszeit
- leichtes Erstellen von GUI-Komponenten mittels Drag und Drop
- erweiterte Palette von GUI-Komponenten verfügbar (z.B. Imagebuttons, MultipleColumn-Lists)

Nachteile:

- unübersichtliche Anordnung der Fenster auf dem Bildschirm
- geringe Softwarezuverlässigkeit (stürzt häufig ab)
- geringe Geschwindigkeit, da ganz in Java programmiert
- hoher Ressourcenbedarf
- kein Syntaxhighlighting im Editor
- die für eine mit dem Java Workshop erstellte graphische Benutzeroberfläche benötigten Runtime Klassen sind über 400kB (!!!) groß

Somit ist es möglich, einfache graphische Benutzerschnittstellen zu erstellen. Diese sind auch optisch sehr ansprechend, da der Java Workshop eigene GUI-Klassen verwendet, und somit nicht die, oft nicht sehr ausgereiften, *java.awt.** Klassen benutzt. Somit stehen hier auch Komponenten wie Imagebuttons und MultipleColumnLists zur Verfügung. Leider müssen beim Laden eines so erzeugten Applets die Runtime-Klassen des Java Workshop mitgeladen werden. Diese sind zusammen mehr als 400 kB groß. Somit würde, in unserem Fall, unser maximal

50kB großes Applet um 400kB größer, nur um zwei erweiterte GUI-Komponenten zu benutzen. Dadurch würden natürlich die Netzlast und vor allem die Downloadzeit unseres Applets zu stark erhöht werden.

Deshalb wurden die graphische Benutzeroberfläche, sowie auch der Rest des Programms, ohne Unterstützung einer kompletten Programmierumgebung erstellt. Als Compiler wurde der normale javac im JDK 1.02 verwendet. Somit waren wir leider beim Erstellen der graphischen Oberfläche auf die im java.awt.* Paket enthaltenen GUI-Klassen angewiesen. Als Erweiterung dieser wurde eine im Netz gefundene Imagebutton-Klasse verwendet. Eine MultipleColumn List konnten wir leider nicht finden. Eine solche selbst zu programmieren erschien uns, verglichen mit dem daraus resultierenden Nutzen, unangemessen.

4.6 Kurzer Überblick über Java

Zuerst einmal zu einer Definition der Sprache Java, die von den gering vermessenen Entwicklern der Firma SUN stammt:

> *JAVA: A simple, object-oriented, distributet, interpreted, robust, secure, architecture neutral, portable, high-performance, multi-threaded and dynamic language*

Diese gut klingenden Attribute machen JAVA natürlich besonders für die Nutzung im Internet sehr interessant, da die dafür besonders wichtigen Aspekte gegeben sind: Plattform-Unabhängigkeit und Verteiltheit.

Auf die einzelnen Punkt soll im Folgenden kurz eingegangen werden. Hierbei werden auch inhärente Vorteile zu Skript-Sprachen deutlich:

Einfach

JAVA sollte eine Sprache sein, die einfach zu erlernen ist. Daher wurden die verfügbaren Sprachkonstrukte möglichst gering gehalten und diverse Möglichkeiten entfernt, die C und C++ zur Verfügung stellen: Zum Beispiel bietet JAVA kleine goto-Anweisung. Statt dessen werden gekennzeichnete break und continue-Befehle geboten sowie Ausnahmebehandlung. Java benutzt keine Header-Dateien und es eliminiert den C Preprozessor. Weil JAVA objektorientiert ist, wurden C-Konstrukte wie struct oder union entfernt. JAVA ermöglicht auch nicht das Überladen von Operatoren und auch nicht das mehrfache Vererben von C++.

Die vielleicht größte Vereinfachung aber ist, daß JAVA keine Zeiger benutzt. Damit diese auch nicht benötigt werden, bietet JAVA eine Garbage Collection, damit sich der Entwickler keine Gedanken um die Speicherverwaltung machen muß.

Objektorientiert

In einem objektorientierten System ist eine Klasse eine Sammlung von Daten und Methoden, die auf Daten arbeiten. Zusammengenommen beschreiben die Daten und die Methoden den Zustand und das Verhalten eines Objektes. Die Klassen sind in einer Hirarchie angeordnet, so daß eine Subklasse das Verhalten von einer Superklasse erben kann. Eine Klassenhierarchie hat immer eine Root-Klasse. Dies ist eine Klasse mit sehr allgemeinem Verhalten.

JAVA enthält einen umfangreichen Satz an Klassen, die in Pakete zusammengefaßt sind, so daß Sie sie in Programmen benutzen können. Zum Beispiel enthält JAVA Klassen, um Komponenten grafischer Benutzerschnittstellen zu erzeugen (das java.awt Paket), Klassen, die die Ein/Ausgabe regeln (das java.io Paket) und Klassen, die die Netzfunktionalität unterstützen (java.net).

Verteilt

Java unterstützt Netzverbindungen auf verschiedenen Ebenen durch Klassen im java.net-Paket. Mit Java ist es genauso einfach, eine entfernte Datei (z.B. durch URL-Connection) zu öffnen wie eine lokale.

Interpretiert

Der Javacompiler erzeugt Bytecode statt direkten Maschinencode. Um ein Javaprogramm tatsächlich ablaufen zu lassen, benutzen sie den Java-Interpreter. Java Bytecodes stellen ein architekturunabhängiges Objektdateiformat zur Verfügung. Ein Java-Programm kann auf jedem System laufen, das einen Java-Interpreter und ein Laufzeitsystem bereitstellt.

Robust

Java, ursprünglich Sprache für Elektrogeräte wurde entworfen, um sehr zuverlässige robuste Software zu schreiben. Java macht natürlich die Qualitätskontrolle nicht unnötig, aber von vornherein sind bestimmte Fehlerquellen eliminiert (siehe Abschnitt „Einfach“).

Sicherheit

Java bietet einen Sicherheitsmechanismus, der sicherstellt, daß der über das Netz geladene Java-Bytecode keine Einschränkungen der Sprache verletzt. Ein Teil des Sicherheitsmechanismus hat damit zu tun, wie Klassen über das Netzwerk geladen werden. Zum Beispiel werden geladene Klassen in einem anderen Namensraum als Standard Java-Klassen gehalten, damit ein gefährliches Applet keine Java-Klassen überschreiben kann.

Geschwindigkeit

Ein sehr bedeutsamer Entscheidungsgrund pro JAVA ist aber der Geschwindigkeitsaspekt. Da der Server eine Vielzahl von Anfragen pro Tag bearbeiten muß, sollte die Verarbeitungsgeschwindigkeit einer Anfrage auch möglichst schnell sein und JAVA ist heute bereits schneller als PERL und soll (laut sun) einmal annähernd die Geschwindigkeit von binären C-Programmen erreichen (derzeit aber noch ca. 20x langsamer als C).

4.7 Java-Probleme

Unter Solaris haben Dialog-Boxen, die mit der *java.awt.Window.pack()*-Methode auf ihre optimale Größe gebracht werden sollen, keinen Rand. Dieser wird erst sichtbar, wenn man die *resize()*-Methode aufruft.

unterschiedliche Koordinatenzählung unter Windows und UNIX-Systemen: In einem Java Frame beginnen Windows-Betriebssysteme die Koordinatenzählung unterhalb der Menüleiste, UNIX-Betriebssysteme zählen die Breite der Menüleiste aber schon zur Höhe des Fensters.

Die *java.awt.Component.update()*-Methode zeichnet die Übersichtszeichnung zwar neu, invertiert dann aber nur den vorher verdeckten Bereich. Deshalb mußte diese überschrieben werden.

Unter dem Windowmanager HP-VUE gibt es unter Java keine Dialog-Boxen.

Die *java.awt.Frame.isResizeable()*-Methode funktioniert nicht richtig: Unter Java erzeugte Fenster haben immer variable Größe.

ANHANG B # Literaturverzeichnis

[1] JAVA in a NUTSHELL, David Flanagan
1996 O'REILLY International Thomson Verlag,
ISBN 3-930673-46-0

[2] DZA Handbuch, Mühlbauer PD-140
09/1996 BMW AG München, Version 3.50

[3] Unix Manual Pages