Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

RAFAŁ KĘDZIORSKI, DETLEF KEHL, OLAF PAETSCH

Client-Server-Anwendung zur Steuerung von Ausgabegeräten

Technical Report TR 98-04 (Dezember 1998)

Client-Server-Anwendung zur Steuerung von Ausgabegeräten

Rafał Kędziorski, Detlef Kehl, Olaf Paetsch

Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)
Takustr. 7, D-14195 Berlin

Email: {kedziorski,kehl,paetsch}@zib.de

Zusammenfassung

Die Steuerung von grafischen Ausgabegeräten erfordert oft mehr als ein oder zwei Optionen, wie man sie vom Unix-*lpr*-Kommando kennt. Hier bietet sich eine grafische Benutzungsschnittstelle zur übersichtlichen Einstellung aller möglichen Parameter und Optionen an. Wenn dieses Gerät zudem nur über eine besondere Treiber-Software zu bedienen ist und in einem heterogenen Intranet zur Verfügung gestellt werden soll, ist eine Steuerung mittels eines WWW-Servers naheliegend. Im Folgenden sollen Lösungsansätze diskutiert werden, die auf HTML/CGI- bzw. Java-Basis eine derartige Steuerung realisieren.

Abstract

It is often the case that graphical hardcopy devices need more than one or two control options like the Unix-*lpr*-command. A driver programme with a graphical user interface (GUI) can satisfy this requirement. If such a device shall be accessible in a heterogenous network a solution using a WWW-server can be used. Possible HTML/CGI resp. Java-based solutions are discussed in the following.

Stichworte: Client-Server-Anwendungen, Ausgabegerätesteuerung.

1 Einführung

Die Möglichkeiten grafischer Ausgabegeräte[1] erfordern häufig die Einstellung verschiedenster Parameter und Optionen. Klassische Steuerungen, die das in Unix-Umgebungen verbreitete Kommando *lpr* bzw. *lp* verwenden, können nicht mehr adäquat eingesetzt werden. In heterogenen Netzwerken, in denen auch Nicht-Unix-Rechner Verwendung finden, können sie überhaupt nicht verwendet werden. Spezielle Ausgabegeräte, wie z.B. Diabelichter, benötigen in vielen Fällen lizenzierte Treiberprogramme, die eine eigene grafische Benutzungsschnittstelle (GUI) mitbringen. Aus lizenzrechtlichen oder technischen Gründen ist es oft auch unmöglich, derartige Treiberprogramme netzweit mit ihrem eigenen GUI anzubieten.

[1] hiermit sind im Folgenden Geräte wie Drucker, Plotter, Diabelichter gemeint, die meist mit dem englischen Begriff Hardcopy Devices bezeichnet werden.

Hier bieten sich Lösungen an, die ein fast immer vorhandenes Intranet verwenden. Mit Intranet wird hier ein aus einem oder mehreren Web-Servern aufgebautes hausinternes Informationssystem verstanden. Der Zugang erfolgt mit Hilfe von Standard-Web-Browsern[2]. Einem Benutzer werden über eine grafische Benutzungsoberfläche (GUI) die notwendigen Einstellungsmöglichkeiten angeboten. Technisch bieten sich hier zwei Lösungsmöglichkeiten an:

- GUI, erzeugt aus HTML-Formularen mit serverseitigen *CGI*[3]-Skripten.
- Mit Java erzeugtes GUI, welches mit einem eigenständigen Dienstprogramm kommuniziert.

Mischformen, die beide o.g. Techniken verwenden, sind natürlich ebenso denkbar. Im Folgenden sollen diese Möglichkeiten sowohl generell als auch im Hinblick auf die Steuerung eines hochwertigen Diabelichters diskutiert werden.

2 Steuerung grafischer Ausgabegeräte

In vielen Fällen müssen vom Benutzer erhebliche Datenmengen zu dem Ausgabegerät geschafft werden, insbesondere wenn es sich um Bildsequenzen handelt. Genauer gesagt: In Netzwerken werden die Bilddaten erst zu dem Treiberrechner (Spooler) übertragen und von dort weiter auf das eigentliche Gerät. Dabei werden die Bilder von den Treiberprogrammen bearbeitet, d.h. ein sog. Raster Image Prozessor (RIP) wandelt das ursprüngliche Bildformat in das gerätespezifische Format um. Neben der Formatwandlung wird vom RIP meist auch noch eine Skalierung der Bilder durchgeführt. Weitere Bildverarbeitungsoperationen wie z.B. Gamma-Korrektur, Farbanpassung, Bildrotation, ... können hinzukommen. Zur Erzielung hochwertiger Ausgaben werden oft Rasterbilder in hoher Auflösung verwendet, so dass pro Einzelbild bei einer Auflösung von z.B. 4k unkomprimiert mehr als 33 MByte übertragen werden müssen. Mögliche Kompressionen sind bildabhängig und bringen erfahrungsgemäß bei verlustfreier Kompression eine Reduktion auf 50 - 20%.

Bei der Übertragung der Bilder finden verschiedene Möglichkeiten Anwendung:

- Aus den Anwendungsprogrammen heraus werden die Bilder an den Drucker geschickt, bzw. in die Druckerwarteschlange gestellt, z.B.über das Print-Menü von *Photoshop*(tm) oder *Word*(tm).

[2] z.B. *Netscape Navigator* oder *Internet Explorer*
[3] Common Gateway Interface

- Systemspezifische Druckkommandos, z.B. Unix-lpr.
- Die auszugebenden Bilddateien werden in ein bestimmtes Dateisystem oder -verzeichnis kopiert und dort von einem Dienstprogramm auf dem Ausgabegerät ausgegeben.

Bei hochwertigen Geräten, deren Möglichkeiten weit über denen 'normaler' Drucker liegen, kommt oft noch eine erhebliche Vielfalt von Parametern und Optionen hinzu. Die Treiberprogramme sind oft ebenfalls mit vielen Einstellmöglichkeiten zur Vorverarbeitung (Bildbearbeitung) versehen. Diese machen die Verwendung eines GUI zur Bedienung unumgänglich. Bei nicht lokal angeschlossenen Geräten bzw. vorhandenen Treiberprogrammen muss das GUI netzweit zur Verfügung stehen. In heterogenen Netzwerken ist das eine nicht immer erfüllbare Forderung. Aus Kostengründen wird oft auch nur eine rechnergebundene Lizenz verwendet, so dass von dem lizenzierten Rechner das GUI des Treiberprogramms netzweit aufgerufen werden muss. Die Darstellung des GUI scheitert dann oft an unterschiedlichen Window-Systemen der beteiligten Rechner.

In kleineren, überschaubaren internen Netzen (Intranet) lassen sich Probleme der genannten Art auf informelle Weise noch lösen und bedürfen keiner Spezialanwendung. So kann ein Mitarbeiter durchaus bei einem nur wenige Räume weiter befindlichen Rechner seine Ausgaben machen, wenn sie von seinem lokalen Rechner nicht möglich sind. Wenn aber ein zentrales Rechenzentrum einer Hochschule z.B. einen hochwertigen Filmbelichter betreibt und diesen campusweit Mitarbeitern und Studenten anbieten will, muss dafür eine spezielle Lösung entwickelt werden. In einem derartigen Umfeld muss eine benutzerbezogene Datenhaltung der auszugebenden Dateien entwickelt werden. Ein Szenario könnte so aussehen, dass ein Benutzer zuerst alle seine Dateien auf den Server überträgt, anschließend daraus Aufträge zusammenstellt und diese dann abarbeiten lässt. Eine Fortschrittskontrolle der Auftragsverarbeitung ist dabei unerlässlich.

Hier bieten sich GUI's an, die über Web-Browser dargestellt werden, deren Eingaben von einem Serverprogramm verarbeitet werden. Auch wenn eine völlige Systemunabhängigkeit nicht immer ganz erreicht werden kann, hat eine derartige Lösung doch viele Vorteile:

- GUI's können gleichartig auf verschiedenen Plattformen mit verschiedenen Web-Browsern dargestellt werden. Ein derartiges GUI muss nur einmal für alle Plattformen implementiert werden.
- Client-unabhängige Verarbeitung der GUI-Eingaben auf dem Server.
- Das GUI auf dem Web-Browser kann dem Benutzer nur die wichtigsten Geräteeinstellungen in intuitiver Form anbieten und so die Bedienung für den gelegentlichen Benutzer vereinfachen.

- Aufbereitung von Statusmeldungen für alle potentiellen Nutzer.

Ein gewichtiger Nachteil bei derartigen Lösungsansätzen ist der doch erhebliche Entwicklungs- und Implementierungsaufwand für ein GUI und den dazugehörigen Server. So müssen Datenübertragung und serverseitige Speicherung sicher und benutzerbezogen gewährleistet sein. Kein Benutzer darf die Daten und Ausgabeaufträge anderer verändern oder löschen können.

Eine Conditio sine qua non an ein Treiberprogramm ist: Es muss in irgendeiner Weise auch ohne sein eigenes GUI betrieben werden können, andernfalls kann es serverseitig nicht angesteuert werden. Nur so kann es in einem Konzept, wie es hier vorgestellt wird, Verwendung finden.

3 HTML/CGI-basiert versus Java-basiert

Für die genannten Forderungen bieten sich im Intranet/Internet-Kontext zwei grobe Lösungsmöglichkeiten an:

HTML/CGI-basiert , oder

Java-basiert

Beide Möglichkeiten sollen nachfolgend näher beleuchtet werden, ihre jeweiligen Inplementierungsprinzipien und ihre Vor- und Nachteile diskutiert werden.

3.1 Client-Server-Applikationen mit HTML-Formularen und CGI-Skripten

Mit HTML aufgebaute Formulare, deren Benutzereingaben an den Server gesendet und dort von CGI-Skripten[4] weiterverarbeitet werden, sind die im Internet gängigste Form, Client-Server-Applikationen zu realisieren. Eingaben über den Web-Browser starten ein Programm auf dem Rechner auf dem auch der Web-Server läuft über eine definierte Schnittstelle. Die Programmiersprache zur Implementierung des Programms ist nicht vorgeschrieben. Allerdings hat sich *Perl* als am weitesten verbreitete (Skript-)Sprache durchgesetzt. Perl wird interpretiert und entsprechende Interpreter sind für alle gängigen Plattformen verfügbar, so dass eine hohe Portabilität garantiert ist.

Ein Ablauf sieht wie folgt aus:

[4]Von Skripten spricht man bei Programmen, die nicht in einen Binärcode übersetzt, sondern als Quelltext direkt interpretiert werden. Im Gegensatz zu den eigentlichen Programmen, die erst in einen maschinenabhängigen Binärcode übersetzt werden müssen, bevor sie ausgeführt werden können. Im Zusammenhang von CGI wird meist die Bezeichnung Skript verwendet, obwohl sich ein Binärprogramm dahinter verbergen kann.

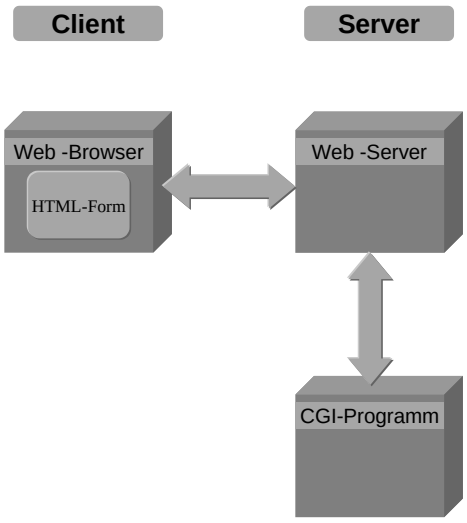

Abbildung 1: Datenflüsse bei Verwendung von Formularen und CGI

Web-Browser (Client)	**Web-Server**
1. fordert HTML-Dokument an	
	2. sendet HTML-Dokument mit Formular
3. sendet Formulareingabe	
	4. verarbeitet Eingabe mit CGI-Skript und sendet Antwort als HTML-Dokument

Falls die Benutzereingabe (3.) Fehler enthält, werden diese erst auf dem Server (4.) erkannt. Dessen Antwort kann dann ein erneutes Formular enthalten, so dass der Zyklus (3. - 4. - 3.) mehrfach durchlaufen wird. In diesem Szenario liegt die Intelligenz ganz auf der Serverseite. Je nach Anforderung der konkreten Applikation kann durch Verwendung von *JavaScript* in den HTML-Dokumenten ein Teil der Verarbeitung auf die Client-Seite verlegt werden, um so das GUI flexibler zu machen. Ebenso kann auch eine gewisse Eingabekontrolle durch JavaScript auf dem Client durchgeführt werden und so die Übertragung vom Client auf den Server minimiert werden. Unabhängig von der Verwendung von JavaScript bleibt die Menge der möglichen GUI-Elemente, mit denen ein Formular aufgebaut werden kann. Hier kann nur aus dem Repertoire der HTML-Formular-Eingabeelemente (`INPUT TYPE=TEXT`, `INPUT TYPE=BUTTON`, `INPUT TYPE=CHECKBOX`, `SELECT`, `TEXTAREA`, `...`) ausgewählt werden, welches die wichtigsten Elemente enthält, deren Erscheinungsbild aber kaum geändert werden kann. Ein im Zusammenhang mit der Steuerung von Ausgabegeräten wichtiges Element ist das Datei-Eingabeelement (`INPUT TYPE=FILE`), mit dessen Hilfe eine Datei vom Client auf den Server übertragen (File upload)

werden kann. Da der Benutzer dabei aktiv eine Datei zur Übertragung auswählt und diese nicht durch Server-Anweisung ohne Benutzereingriff geholt wird, ist hier auch kein Sicherheitskonzept verletzt.

Die Implementierung von Client-Server-Applikation ist mit diesen Mitteln schnell und problemlos möglich. Für einfache Anwendungen ist sie auch gut geeignet, etwa wenn ein Benutzer eine Anfrage an eine Suchmaschine startet. Wenn aber die Anwendung einen sitzungsorientierten Dialog erfordert, dann sind hiermit schnell die Grenzen des Machbaren erreicht, denn HTTP ist ein zustandsloses Protokoll. Durch Verwendung von nicht angezeigten Eingabeelementen (`INPUT TYPE=HIDDEN,VALUE=...`) in Formularen oder durch Setzen von Cookies kann teilweise dieses Manko umgangen werden. Ein echtes sitzungsorientiertes Protokoll kann aber nicht erreicht werden, da der Benutzer sowohl keine Cookies zulassen als auch jederzeit durch Browser-Navigation zwischen den Web-Seiten springen kann.

3.2 Client-Server-Applikationen auf der Basis von Java-Applets

Die Entwicklung und Implementierung von Client-Server-Applikationen mit client-seitigen Java-Applets[5] erfordert einen wesentlich größeren Aufwand als HTML-Formular/CGI-basierte Anwendungen. Um ein in seinen Möglichkeiten wesentlich flexibleres Formular mit Java zu programmieren, muss ein vollständiges Anwendungsprogramm erstellt werden. Java ist eine komplette objektorientierte Programmiersprache, die übersetzt in den sog. Byte-Code in der Virtual Machine (VM) des Web-Browsers läuft. Ein HTML-Form dagegen bedarf nur weniger Zeilen, die die gewünschten Eingabeelemente des Formulars aufzählen und keinerlei weitere Logik enthält.

Wenn die Anwendung eine Kommunikation mit einem weiteren Applikations-Server voraussetzt oder wenn ein sitzungsorientierter Dialog verwendet werden soll, dann muss auf seiten des Clients ein Java-Applet verwendet werden.

Das Java-Applet wird vom Web-Server an den Browser geschickt und dort ausgeführt. Während das Applet läuft, kommuniziert es mit einem weiteren Server-Programm, welches auf dem gleichen Rechner wie der Web-Server laufen muss, da das Sicherheitskonzept[6] von Java Netzverbindungen nur zu dem Rechner zulässt, von dem das Applet geladen wurde. Durch einen Proxy-Server kann aber bei Bedarf eine Verbindung zu einem anderen Rechner ermöglicht werden. Im Zusammenhang mit der Steuerung von Ausgabegeräten hat der weitere Applikations-Server die Aufgabe, die eigentliche

[5] in Web-Seiten integrierte Java-Anwendungsprogramme.

[6] Die hier erwähnten Einschränkungen durch das Java-Sicherheitskonzept beziehen sich auf Java-Applets, die vom Server auf den Browser übertragen werden und vom Browser in der sog. *Sandbox* ausgeführt werden. Stand-alone Java-Programme unterliegen diesem Sicherheitskonzept nicht.

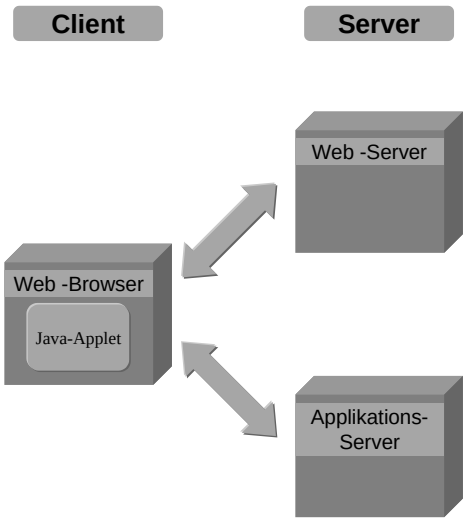

Abbildung 2: Datenflüsse bei client-seitigem Java-Programm und zusätzlichem Applikations-Server

Geräteansteuerung zu realisieren. Das kann sowohl direkt geschehen als auch durch Zusammenstellung von Parametern und Optionen für zeitlich spätere Ausgaben. Auch hier muss wieder mehr Implementierungsarbeit geleistet werden als bei einer CGI-Lösung, denn die Kommunikation zwischen dem Applet und dem Applikations-Server wird nicht wie bei CGI-Skripten vom Web-Server übernommen, sondern muss zusätzlich programmiert werden. Allerdings ist es nicht notwendig, dass der Server eine HTML-Seite zurück sendet, sondern die Kommunikation kann der Anwendung entsprechend aufgebaut werden. Für die Client-Server-Kommunikation auf Basis von Sockets bietet Java das `java.net`-package.

Ein Problem bei der Verwendung von Java ist das starke Sicherheitskonzept, das es nicht ermöglicht, lokale Benutzerdateien zu lesen oder zu schreiben. Da es aber absolut notwendig ist, eine Möglichkeit zu haben, die auszugebenden Dateien auf den Server zu kopieren, der die Ausgabe steuert, kann entweder

- zusätzlich zum Java-GUI ein HTML-Form mit dem Datei-Browser-Button verwendet werden, oder
- es muss ein zertifiziertes Applet[7] verwendet werden. Das zertifizierte Applet liest dann die lokale[n] Benutzerdatei[en] und sendet sie zu einem Server.

[7]Zertifiziert bedeutet, dass die Herkunft des Applets dem Benutzer bekannt ist und dass er dieser Herkunft auch in Bezug auf die Sicherheit seiner Daten vertraut.

- Als Weiteres besteht die Möglichkeit, ohne Verwendung von HTML und Java, sondern z.B. mit Ftp die Dateien zu übertragen. Die damit vorher übertragenen Dateien würden dann von einem Java-Applet zur Weiterbearbeitung angezeigt.

3.3 Andere Lösungsmöglichkeiten

Unter der Voraussetzung, dass sowohl auf Client- als auch auf Server-Seite Java Verwendung findet, kann eine Client-Server-Applikation mit *RMI*[8] implementiert werden. Bei RMI handelt es sich um eine Art objektorientierte Version der Remote Procedure Calls, die ab Java 1.1 verfügbar ist.

3.4 Fazit

Für Anwendungen, die nicht eines sitzungsorientierten Dialogs bedürfen, sind HTML-Formulare in Verbindung mit CGI-Skripten die am einfachsten und am schnellsten zu realisierende Lösung. Dabei wird allerdings bei jedem Absenden auf Seiten des Servers ein Programm neu gestartet, was eine zusätzliche Rechnerbelastung bedeutet und die Antwortzeiten verlängert.

Für komplexere Anwendungen muss auf eine 'echte' Programmiersprache mit einem speziellen Applikations-Server neben dem eigentlichen Web-Server zurückgegriffen werden. Im Zusammenhang mit der Steuerung grafischer Ausgabegeräte muss immer dann auf Java für das GUI im Browser zurückgegriffen werden, wenn der Benutzer sich Aufträge interaktiv zusammenstellen können soll. GUI's können bei einer Java-Implementierung wesentlich umfangreichere, flexiblere, und insbesondere problemangepasstere Eingabeelemente enthalten. Der Entwickler ist nicht allein auf die Standardelemente von HTML angewiesen. Er kann das schon recht umfangreiche `java.awt`-package auf spezielle Bedürfnisse hin erweitern.

Es muss allerdings in Betracht gezogen werden, dass Java nicht von jedem Browser unterstützt wird, bzw. dass Benutzer die Ausführung von Java nicht gestatten können.

Bei jeder Art von Client-Server-Applikation muss entschieden werden, wie die Aufgabenverteilung von Client und Server aussieht. Im Allgemeinen sollte so entschieden werden, dass eine möglichst kleine Datenmenge zwischen Client und Server übertragen werden muss und eine schnelle Bearbeitung gewährleistet ist.

Insbesondere bei der Entwicklung von Netzanwendung läuft man schnell in Gefahr, nicht mehr aktuelle Verfahren zu verwenden oder auf scheinbar moderne Verfahren zu setzen, die sich doch nicht allgemein durchsetzen und dann innerhalb kurzer Zeit von anderen Verfahren abgelöst werden.

[8] Remote Method Invocation

Im folgenden Kapitel wird eine konkrete Realisierung einer Client-Server-Applikation beschrieben.

4 Java-basierte Steuerung eines Diabelichters

4.1 Überblick

Im Jahre 1997 wurde durch das Konrad-Zuse-Zentrum ein Diabelichter, vorrangig für die interne Nutzung, beschafft. Der Lieferumfang des Belichters umfasst die Steuersoftware *Conga*. Dabei handelt es sich um ein Programm, das auf einem Unix-Arbeitsplatzrechner mit lokal über SCSI angeschlossenem Diabelichter dessen Steuerung übernimmt. Die Lizenz erlaubt nur die Ausführung auf diesem einen Rechner. Das umfangreiche GUI ist, wie unter Unix üblich, als *X-Applikation* implementiert. Zusätzlich dazu ist eine Steuerung über eine Kommandozeilen-Schnittstelle möglich.

Ziel des Projekts war die Erstellung einer Software-Umgebung, die die Ansteuerung des Diabelichters von einer Vielzahl von Arbeitsplatzrechnern aus ermöglicht, ohne dort jeweils die Conga-Software direkt verfügbar machen zu müssen. Außerdem sollte die große Zahl der in Conga vorgesehenen Steuerungsmöglichkeiten gekapselt werden, so dass sich der Anwender nur mit den für eine Standardnutzung erforderlichen Parametern vertraut machen muss.

Der Service zur Ansteuerung des Diabelichters (*DiaServ*) teilt sich auf in eine Client- und eine Server-Komponente. Der Kern des auf dem lokalen Rechner des Nutzers laufenden Clients wurde als Applet in der Programmiersprache *Java* implementiert. Das DiaServ-Applet kommuniziert über eine Socket-Verbindung mit einer Server-Komponente, die in der Skriptsprache *Perl* implementiert wurde. Auf die Vor- und Nachteile der einzelnen Lösungen wurde in diesem Report weiter oben eingegangen.

Die Funktion des DiaServ-Applets besteht beinahe ausschließlich in der Steuerung eines leicht bedienbaren GUIs, welches im Wesentlichen die Ein- und Ausgabe von Zeichenketten (z.B. Dateinamen) übernimmt. Da die längerfristige (über die Dauer einer Sitzung hinausgehende) Verwaltung der Daten ohnehin vom Server übernommen wird, liegt es nahe, die algorithmisch aufwendigeren Teile nicht im Client, sondern im Server zu implementieren, zumal die Skriptsprache Perl über sehr mächtige Funktionen zur Bearbeitung von Zeichenketten verfügt.

4.2 Konzeption

Ausgangsbasis für die Erstellung einer Diasequenz sind Bilddateien, die in unterschiedlichen Grafikformaten auf dem Heimatrechner des Anwenders,

in der Regel auf seinem Arbeitsplatzrechner, vorliegen. Damit die Belichtung einer solchen Sequenz auf dem Diabelichter problemlos asynchron und damit unabhängig von Benutzermaßnahmen erfolgen kann, sollen die Bilddateien vor der Belichtung auf einem entsprechenden Serversystem gesammelt werden. Dazu sind auf dem Serversystem für die einzelnen Anwender voneinander abgegrenzte Bereiche einzurichten.

Auf Basis der bereits in unserem Hause vorhandenen Rechnerarchitekturen wurde das Serversystem auf einem Unix-Rechner eingerichtet. Der administrative Aufwand bei der Einrichtung von Benutzerbereichen mittels Unix-Kennungen erschien als zu hoch, zumal es sich um sporadisch auftretende Anwender aus unterschiedlichen Arbeitsbereichen handelt. Ein Anwender sollte eine Kennung, im Folgenden DiaServ-Kennung genannt, selbst für sich einrichten und wieder entfernen können, während bei einer Unix-Kennung immer ein Systemadministrator tätig werden muss. Darüber hinaus kann auf die relativ guten Sicherheitseigenschaften einer Unix-Kennung verzichtet werden, da es sich vorrangig um eine Nutzung in einem Intranet handelt.

Der Server ordnet mit Hilfe der DiaServ-Kennung Bilddateien dem Projekt eines Anwenders zu. Ein Anwender kann für mehrere Projekte mehrere Kennungen einrichten. Jeder Kennung ist ein weitgehend frei wählbares Passwort zugeordnet, mit dem sich ein Anwender zusammen mit einer Kennung legitimiert. Das DiaServ-Passwort wird im Klartext übertragen und gespeichert und dient lediglich dazu, einen zufälligen, unbeabsichtigten Zugriff in einer kooperativen Nutzungsumgebung zu verhindern.

Ein gewisses Problem stellt der Bilddatei-Transfer zum Server dar, da der Anwender nicht über eine Unix-Kennung auf dem Server verfügt und damit z.B. die Nutzungsmöglichkeit des Ftp-Dienstes entfällt. Eine Auslegung des Java-Clients für diesen Zweck wird durch die starken Sicherheitsrestriktionen für Java-Applets verhindert (siehe oben). Einen Ausweg bietet der Dateitransfer, wie er über ein HTML-Formular vermittelt werden kann (siehe oben). Unter Sicherheitsgesichtspunkten ist dieser im Wesentlichen wie ein vom Anwender initiierter Ftp-Transfer hin zu einem anonymen Ftp-Server zu beurteilen. Damit wird neben dem Applet der DiaServ-Client von einer kleinen Gruppe von HTML-Formularen gebildet, die server-seitig mit entsprechenden CGI-Skripten kommunizieren und damit den Dateitransfer ermöglichen.

Während der Aktivität des DiaServ-Clients muss das Datenbild von Client und Server, bezogen auf die gerade benutzte Kennung, synchron gehalten werden. Dies kann der Client nur dann sicherstellen, wenn nicht gleichzeitig ein schreibender Datenzugriff von anderer Seite, über deren Aktivität der Client nichts weiß, erfolgt. Aus diesem Grund folgt die Anwendung des DiaServ-Clients einem Sitzungsmodell: Eine Aktivität des DiaServ-Clients

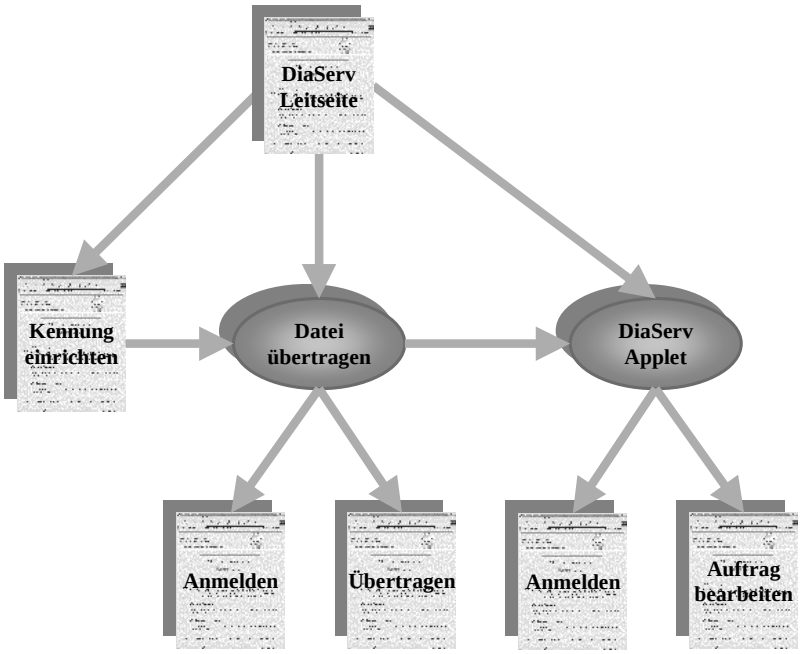

Abbildung 3: Struktur des DiaServ-Clients

muss durch Anmeldung und Abmeldung abgegrenzt werden, wodurch eine Sitzung etabliert wird. Zu einer Kennung kann immer nur höchstens eine Sitzung aktiv sein. Sollte z.B. durch einen Verbindungsfehler eine Sitzung nicht regulär beendet worden sein, so erhält der Anwender bei einem erneuten Anmeldeversuch die Möglichkeit, die alte Sitzung durch die gewünschte neue zu ersetzen. Dieses Sitzungskonzept kann auf einen Anwender ungewohnt wirken, da die übliche Nutzung des WWW über Browser und Applets einem zustandslosen Paradigma folgt.

Ein wesentliches Strukturelement des Diabelichtungs-Services ist der Belichtungsauftrag. Ein Auftrag umfasst eine Reihe von Bilddateien, wobei jede Datei mit den Attributen *Anzahl der Belichtungen* und *Auflösung* ausgestattet ist. Aufträge sind über einen Namen eindeutig identifizierbar und bleiben über Sitzungen hinweg bestehen. Sie können in späteren Sitzungen geändert und ein weiteres Mal zur Belichtung gesendet werden.

4.3 DiaServ-Client

Der DiaServ-Client wird zentral von einem Java-Applet und zusätzlich einigen HTML-Formularen zur Unterstützung des Bilddatei-Transfers zum Server gebildet. Hauptfunktion des Applets ist die Realisierung eines *Gra-*

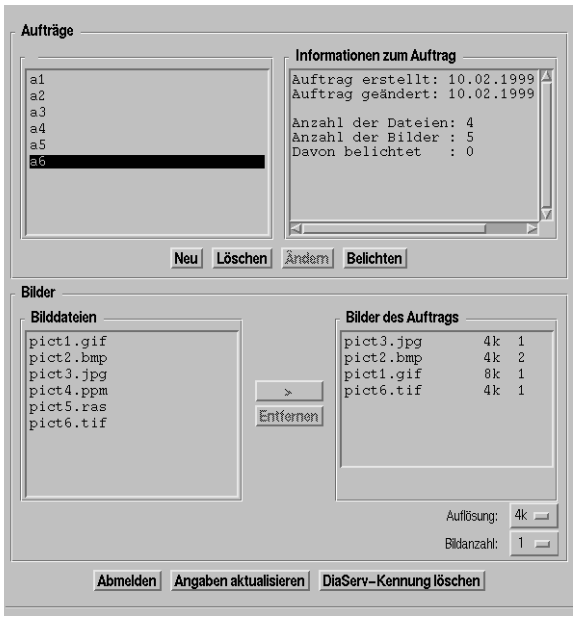

Abbildung 4: Bedienelemente des DiaServ-Applets

phical User Interfaces (GUIs)[9], welches die Erstellung und Modifikation von Aufträgen, das Absenden dieser Aufträge zum Belichter und die Verfolgung der Ausführung von Belichtungsaufträgen ermöglicht.

Da sich die Kommunikation mit dem Server fast ausschließlich auf den Austausch von im GUI angezeigten Textelementen beschränkt, waren an den Durchsatz des entsprechenden Protokolls keine besonderen Anforderungen zu stellen: Die Antwortzeiten des Systems sind im Allgemeinen im Vergleich zur Reaktionszeit des bedienenden Menschen vernachlässigbar. Die Kommunikation wurde auf der Basis von Mitteilungen mit strukturierten ASCII-Texten implementiert, die der Client mit dem Server über eine in Unix-Umgebungen übliche Socket-Schnittstelle austauscht. Vorbild für die Textstruktur der Mitteilungen war das *Summary Object Interchange Format (SOIF)*, wie es bei Harvest-Datensammlungen zur Literatur-Recherche eingesetzt wird [11].

[9]Wegen der zeitlichen Verzögerung, mit der neue Java-Versionen in die gängigen WWW-Browser Eingang finden, bleibt die Ausführbarkeit trotz des Einsatzes der bereits gut etablierten Version 1.1.6 des JDKs auf die aktuellsten Browser-Versionen beschränkt; wir empfehlen deshalb zur Anwendung des Dienstes den Browser Netscape Navigator in (mindestens) der Version 4.5.

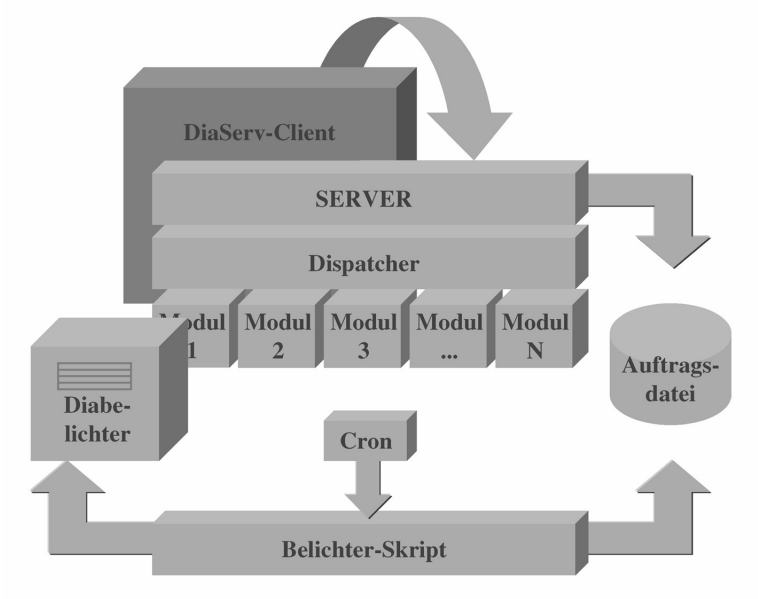

Abbildung 5: Struktur des DiaServ-Servers

4.4 DiaServ-Server

Die Implementierung des Servers erfolgte in der Skriptsprache Perl. Hier konnten wir vor allem von der guten Unterstützung der Socket-Programmierung, den bekannt mächtigen Möglichkeiten zur Zeichenkettenverarbeitung und der schnellen Erstellbarkeit prototypischer Komponenten profitieren. Nachteile von Perl, wie sie im Verzicht auf das Erzwingen hoher Strukturiertheit (wie z.B. bei C++) zu sehen sind und die bei großen, sehr komplexen Projekten zu Problemen führen, konnten wegen der moderaten Größe unseres Projekts (einige Tausend Zeilen Quellcode) unberücksichtigt bleiben.

Der Server ist, wie in Unix-Umgebungen üblich, als resident laufendes Programm (Daemon) ausgelegt, und folgt in der äußeren Struktur dem Beispiel in [10]: Ein übergeordneter Prozess hört die Kommunikation auf dem Socket ab. Trifft eine Mitteilung des Clients ein, wird über den Systemaufruf *fork* ein abgeleiteter Prozess generiert, der diese Mitteilung (und nur diese) bearbeitet. Diese Vorgehensweise erlaubt die (pseudo-)gleichzeitige Bearbeitung simultaner Anforderungen; wegen der Bearbeitung einer jeden Mitteilung durch einen eigenen Prozess werden Störungen des Gesamtablaufs bei einer lokalen Fehlfunktion weitgehend verhindert.

Einzelnen Operationen auf dem GUI wie Anmelden, Abmelden, Einrichten eines Auftrags, Setzen einer Eigenschaft für eine Bilddatei, sind entsprechende Mitteilungstypen zugeordnet, in denen die vom Server geforderte

Leistung durch Attribut-Wert-Paare näher gekennzeichnet ist. Ein *Dispatcher*-Modul verteilt die Anforderungen auf entsprechende Module, welche dann die einzelnen Leistungen erbringen.

Über das GUI abgeschickte Belichtungsaufträge werden in einer Datei zusammengefasst, die von einem alleinstehenden Perl-Skript bearbeitet wird. Die Funktion dieses Skripts wird von einem Cron-Job gesteuert. Das Skript erkennt, ob der Belichter zur Entgegennahme einer Bilddatei bereit ist, und fällt andernfalls in einen Wartezustand. Gleichzeitig mit der Steuerung des Belichters wird das DiaServ-Applet und damit der Anwender mit einer entsprechenden Verlaufsinformation versorgt.

5 Ausblick

Der modulare Aufbau des Servers ermöglicht es, auf einfache Weise die Funktionalität zu erhöhen. Eine neue Funktion erfordert nur die Implementierung eines entsprechenden Moduls, welches dem *Dispatcher* bekannt gemacht werden muss. Dazu ist keine Programmänderung des Dispatchers notwendig, sondern das neue Modul wird nur in einem bestimmten Dateiverzeichnis abgelegt. Anschließend kann diese neue Funktion genutzt werden, ohne dass die schon bestehenden Module geändert werden müssen. Auf der Client-Seite müsste natürlich die Anforderung der neuen Funktion implementiert werden. Die bisherigen Erfahrungen mit dem Diabelichtungs-Service zeigten, dass die angebotene Funktionalität fast immer ausreicht, so dass eine Funktionserweiterung eher im Bereich weiterer möglicher Endgeräte stattfinden wird. Ein Kandidat dafür wäre z.B. die automatische Videoaufzeichnung einer Bildsequenz.

Literatur

[1] J. David, M. Grave: *WWW and Virtual Reality for Scientific Visualization*, in: W. Lefer, M. Grave: *Visualization in Scientific Computing '97*, Springer Wien NewYork, 1997.

[2] C. Evans, D. Lacey, D. Harvey, D. Gibbons, A. Krasun: *CLINET/SERVER: A Handbook of Modern Computer Design*, Prentice Hall, 1995.

[3] D. Flanagan: *Java in a Nutshell, 2. Ed.*, O'Reilly & Associates, May 1997.

[4] M. Jern: *Information Drill-Down Using Web Tools*, in: W. Lefer, M. Grave: *Visualization in Scientific Computing '97*, Springer Wien New York, 1997.

[5] M. Hall: *CORE Web Programming*, Prentice Hall, 1998.

[6] P. Müller, I. Decker: *The Electronic Visualization Library*, Technical Report TR 97-09, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Dezember 1997.

[7] S. Münz: *SELFHTML HTML-Dateien selbst erstellen*, `http://www.teamone.de/selfhtml/selfhtml.htm`, 1998.

[8] R. Orfali, D. Harkey, J. Edwards: *The Essential Distributed Objects Survival Guide*, John Wiley & Sons, 1996.

[9] J. C. Trapp, H.-G. Pagendarm: *A Prototype for a WWW-based Visualization Service*, in: W. Lefer, M. Grave: *Visualization in Scientific Computing '97*, Springer Wien New York, 1997.

[10] L. Wall, T. Christiansen, R. L. Schwartz: *Programming Perl, 2. Ed.*, O'Reilly & Associates, September 1996.

[11] *Review of Summary Object Interchange Format (SOIF)*, `http://www.roads.lut.ac.uk/RADAR/soif-review.html`, 1996.