4 Umsetzung

In diesem Kapitel werde ich den, zum Zeitpunkt der Erstellung dieser Arbeit aktuellen Entwicklungsstand der liFe.f0rm beschreiben sowie Anleitungen zum Umgang mit derselbigen geben, bzw. aufzeigen, wie man diese (mit der beiligenden CD) zum *Leben* erwecken kann. Die Abbildung 2 enthält eine Übersicht des Weges, von der Benutzer-Eingabe bis zur resultierenden Klangdatei, in der Form eines Flußdiagramms.

4.1 Der Server Prototyp

4.1.1 Konfiguration

Die beiden Arrays `$config`[13] (siehe Abbildung 3) und `$bin` (siehe Abbildung 4) werden in der Datei `server_config.inc.php` (vgl. A.1 auf Seite 77) gesetzt, die zur Steuerung des Servers, u.a. den Umgang mit den Dateien betreffend, dienen[14].

Über `$config[sr]` ist es beispielsweise möglich, eine globale Samplerate[15] von 48000 Hz[16] zu definieren, welche sich in den resultierenden Dateien niederschlägt. Somit werden die heruntergeladenen Dateien beim Normalisierungsvorgang (vgl. 4.1.8 auf Seite 29) in die entsprechende Samplerate umgewandelt (*resampled*), falls deren originale davon abweicht. Dabei ist darauf zu achten, daß eine neue liFe.f0rm mit geändertem Sampleraten-Parameter gestartet werden muß[17], da im aktuellen Status des Servers vorhandene Dateien nicht auf die richtige Samplerate überprüft und somit auch nicht ggf. umgewandelt werden, was wiederum zu Fehlermeldungen führt, welche die

[13] Variablen aller Art werden in PHP mit einem vorangestellten „`$`“ gekennzeichnet

[14] vgl. **sh** ▷ `server.php -show config`

[15] Anzahl der Samples pro Sekunde

[16] Hertz = 1/Sekunde

[17] also ein leeres, bzw. nicht existentes Verzeichnis beim Server-Start gewählt werden muß (vgl. 4.1.2 auf Seite 19)

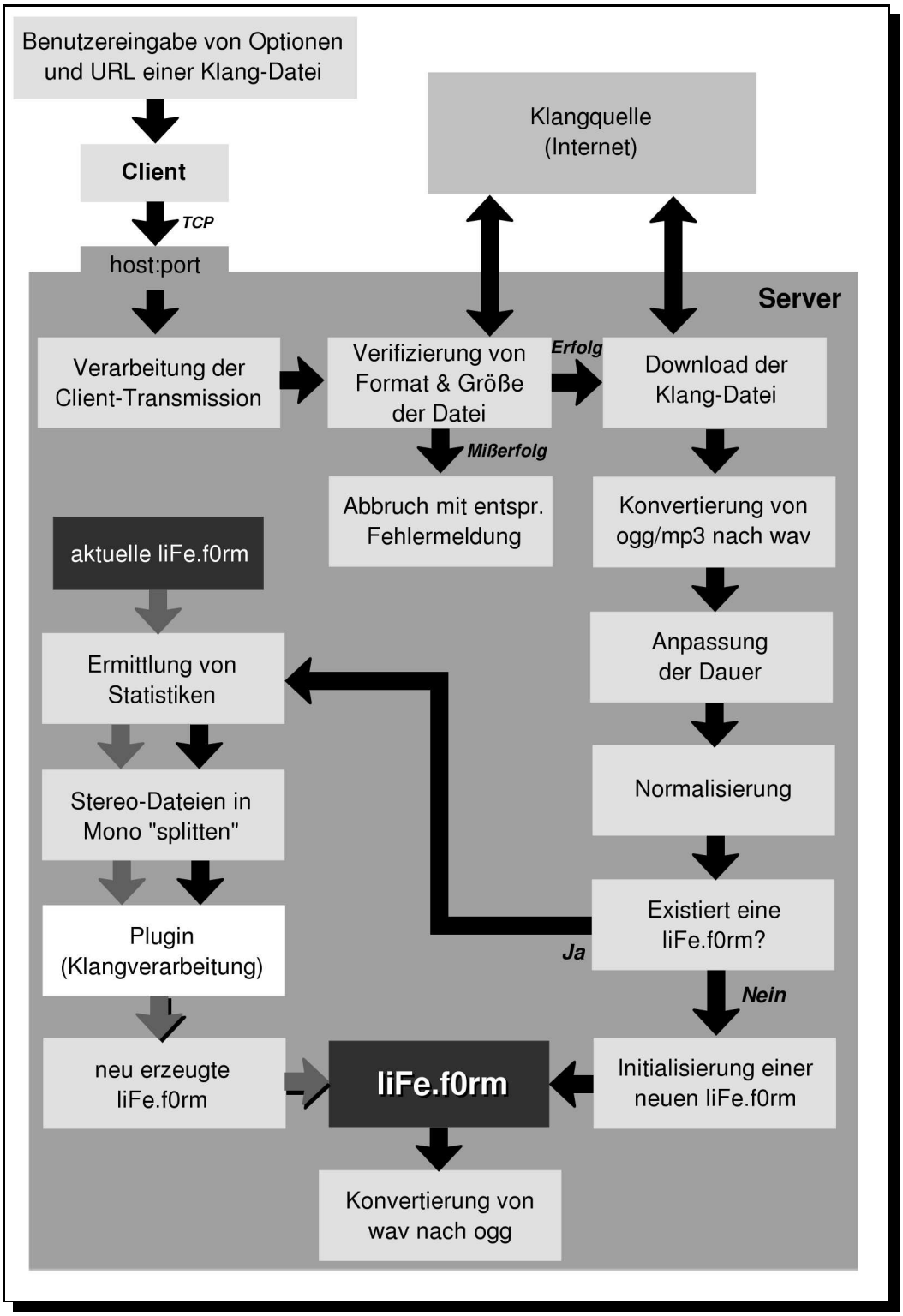

Abbildung 2: *liFe.f0rm Flußdiagramm*

```
$config: Array
(
    [sr] => 44100
    [bit] => 16
    [rootdir] => ..
    [bindir] => ../organs
    [tmpFilename] => lf_download_tmpfile.wav
    [mimetype] => (audio|application)\/(x-)?(mpeg|ogg|wav)
    [hr] =>
=================================================================

    [show] => all
    [host] => 0.0.0.0
    [port] => 12965
    [tmpdir] => /tmp
    [liFedir] => ../liFe
    [maxsize] => 2.0
    [maxlength] => 8.0
    [logfile] => ../liFe.log
    [email] => ruebler@folkwang-hochschule.de
    [plugin] => grain
    [verbose] => 2
    [liFe_maxlength] => 8.0
    [file_maxlength] => 8.0
    [liFe] => ../liFe/liFe.wav
)
```

Abbildung 3: *Konfigurations-Anweisungen für den Server*

```
$bin: Array
(
    [sndinfo] => ../organs/csound -U sndinfo
    [pvanal] => ../organs/csound -U pvanal
    [csound] => ../organs/csound -m2 -d -W -o
    [sox] => ../organs/sox
    [sox_paraIn] => -t auto
    [sox_paraMonoOut] => -r 44100 -w -c 1
    [soxmix] => ../organs/soxmix
    [lame] => ../organs/lame
    [oggdec] => ../organs/oggdec
    [oggenc] => ../organs/oggenc
    [oggenc_quality] => --quality 10
    [ogginfo] => ../organs/ogginfo
    [stderr2stdout] => 2>&1
    [no_stderr] => 2>/dev/null
    [wget] => /usr/bin/wget --tries=3
    [file] => /usr/bin/file
    [dd] => /bin/dd
)
```

Abbildung 4: *Externe Programme („Organe“)*

Klangverarbeitung verhindern. Es besteht jedoch die Möglichkeit, eine vorhandene liFe.f0rm mit differierender Samplerate als Basis zu nehmen, wenn man diese als erste zu verarbeitende Datei benutzt. Denn wenn sich noch keine Datei namens `liFe.wav`[18] in dem Verzeichnis, welches das *Zuhause* der liFe.f0rm darstellt, befindet, wird die erste zu verarbeitende Datei als Initial-Version behandelt (vgl. 4.1.5 auf Seite 25).

Gegenwärtig unterstützte Klangdatei-Formate sind *Ogg-Vorbis (ogg)* [51], *MPEG Layer I,II,III (mp3)* [52] und *WAVE (wav)* [53]. Diese werden durch den in der Konfigurations-Variablen `$config[mimetype]` befindlichen Regulären Ausdruck auf ihr Format überprüft. *Mimetype* [56] ist ein

[18] gesetzt in der Variablen `$config[liFe]`

Internet-Standard zur Bezeichnung von Datei-Formaten, der sich aus einer Datei-Gruppe gepaart mit dem eigentlichen Format zusammensetzt. Der Mimetyp einer HTML-Datei beispielsweise wird der Gruppe *text* zugeordnet. Somit lautet die komplette Bezeichnung für eine solche Datei *text/html*. Bei ersten Testdurchläufen mußte ich jedoch feststellen, daß dieser Standard in Bezug auf Klangdateien nicht eindeutig definiert ist. Somit mußte ich einen Regulären Ausdruck hinzuziehen, um unterschiedliche Gruppenzuordnungen, bzw. Format-Schreibweisen, des Mimetyps zuzulassen. Folgender Ausdruck wird gegenwärtig verwendet[19]:

```
$config[mimetype] = "(audio|application)\/(x-)?(mpeg|ogg|wav)";
```

Löst man diesen Regulären Ausdruck auf, erhält man folgende mögliche Schreibweisen:

- audio/mpeg, audio/x-mpeg, application/mpeg, application/x-mpeg
- audio/ogg, audio/x-ogg, application/ogg, application/x-ogg
- audio/wav, audio/x-wav, application/wav, application/x-wav

Im schon zuvor erwähnten Array `$bin` (siehe Abbildung 4) sind alle extern verfügbaren Programme, die zur eigentlichen Klang- und Dateibearbeitung genutzt werden, gesammelt. Im einzelnen werden folgende Programme benutzt:

- *Csound* [44], zur Ermittlung von Statistiken sowie zur eigentlichen Klangverarbeitung in den Plugins (vgl. 4.2, Seite 31 ff.)

[19] Der *Slash* („/“) hat eine spezielle Bedeutung innerhalb Regulärer Ausdrücke und muß deshalb mit einem vorangestellten *Backslash* („\“) der Interpretation durch den Regulären Ausdruck entzogen werden.

- *SoX* [45], zur Ermittlung von Statistiken, Format-Umwandlungen (Normalisierung) sowie Längenanpassungen der Dateien
- *lame* [46], zum Dekodieren von MP3 Dateien
- *oggdec*, zum Dekodieren von Ogg-Vorbis Dateien
- *oggenc*, zum Enkodieren der liFe.f0rm in das Ogg-Vorbis Format
- *wget*, zur Ermittlung des Mimetyps und der Dateigröße über das Protokoll „http“ (HyperText Transfer Protocol) [64]
- *file*, zur Ermittlung des Mimetyps lokaler Dateien

In der Konfigurations-Datei besteht weiterhin die Möglichkeit, Standardwerte (siehe Abbildung 5) zu setzen, die benutzt werden, falls kein Argument für den jeweiligen Wert auf der Kommandozeile übergeben wurde (vgl. 4.1.2 auf Seite 19). Diese Standards werden in den `$config`-Array (vgl. Abbildung 3) übernommen, auch wenn kein entsprechendes Argument übergeben werden kann. Um Argumente *freizuschalten*, also über die Kommandozeile nutzbar zu machen, müssen sie Element des Arrays `$options` sein.

Zuletzt wird die Nachricht definiert, mit der der, u.U. im Hintergrund aktive, Server-Prozess vom Client gestoppt werden kann (`$serverDie`) sowie die Konfigurations-Datei der Plugins eingebunden, die die verfügbaren Plugins bereitstellt und das Standard-Plugin definiert (`$default[plugin]`).

4.1.2 Kommandozeilen-Argumente

Nachdem im ausführbaren Skript `server.php` (vgl. A.3 auf Seite 94) die Konfiguration sowie die nötigen Funktionen eingelesen wurden, werden die auf der Kommandozeile an das Skript übergebenen Argumente verarbeitet[20].

[20]Die Argumente können sowohl mit einem, als auch mit zwei vorangestellten Bindestrichen geschrieben werden.

```
$default: Array
(
    [host] => 0.0.0.0
    [port] => 12965
    [tmpdir] => /tmp
    [liFedir] => ../liFe
    [maxsize] => 2.0
    [maxlength] => 8.0
    [logfile] => ../liFe.log
    [email] => ruebler@folkwang-hochschule.de
    [plugin] => grain
)
```

Abbildung 5: *Für den Server-Betrieb gesetzte Standardwerte*

Dies schließt auch eine Überprüfung auf sinnvolle Werte ein, wie z.B., ob angegebene Verzeichnisse existieren oder der Wert einer Zahl entspricht, falls das Argument auf diese beschränkt ist.

Die in Abbildung 6 aufgelisteten Argumente stehen für den Aufruf von `server.php` zur Verfügung[21]. Falls eines der optionalen Argumente nicht gesetzt wird, wird der in der Server-Konfiguration definierte (vgl. 4.1.1 auf Seite 14), in Abbildung 6 in Klammern angegebene, Standardwert benutzt.

Um mehrere liFe.f0rm-Instanzen parallel auf demselben Rechner *existieren* zu lassen, benötigt man mindestens zwei der optionalen Argumente. Zum einen `-liFedir`, über welches man das Heimat-Verzeichnis der liFe.f0rm definiert, zum anderen `-port`, was gewissermaßen die *Telefonnummer* der liFe.f0rm darstellt. Denn es ist nicht möglich, mehrere Server auf denselben TCP-Port [63] *hören* zu lassen, über den die Kommunikation mit den Clients stattfindet.

Über `-host` lässt sich die IP-Adresse [62], bzw. der Domain-Name [60] des lokalen Rechners setzen. Standardwert hierfür ist die IP-Nummer „0.0.0.0“,

[21] vgl. **sh** ▷ `server.php --help`

```
Usage:  ./server.php [OPTION]...

options:
        --host HOST      use HOST (default: 0.0.0.0)
        --port PORT      PORT to listen on (default: 12965)
        --tmpdir DIR     use DIR to store downloaded soundfiles
                         (default: /tmp)
        --liFedir DIR    use DIR to store liFe-soundfiles
                         (default: ../liFe)
        --maxsize SIZE   maximum SIZE [in MB] of the download file
                         (default: 2.0)
        --maxlength DUR  maximum DURation [in hh:mm:ss.frac] of
                         soundfiles (default: 8.0)
        --show OUTPUT    'details' for verbose output
                         'all' for more verbose output
                         'ALL' for debug output
                         'plugins' for all available plugins
                         'config' for a dump of config variables
                         (verbose level can be overwritten by
                          client integer argument '--verbose')
        --help, -h       display this help and exit
```

Abbildung 6: *Kommandozeilen-Argumente für den Server*

welche alle lokal verfügbaren Adressen umfasst. Diese schließt beispielsweise eine Netzwerkkarte, die mit einer bestimmten IP-Nummer konfiguriert ist, automatisch ein. Somit ist es möglich, mit einem auf einem anderen Rechner befindlichen Client den Server anzusprechen. Falls man die Adresse „localhost“, bzw. die entsprechende IP-Nummer „127.0.0.1“, für das `host`-Argument wählt, ist es ausschließlich mit auf dem selben Rechner ausgeführten Clients möglich, den Server anzusprechen. Der Server ist somit nicht von *außen* erreichbar.

Um die Argument-Eingabe zu erleichtern, kann man das Skript `server+dir+port.sh` (vgl. A.4 auf Seite 102) zum Starten des Servers verwenden. Dieses erwartet ein Verzeichnis auf dem lokalen Rechner, welches ggf. erzeugt wird, sowie optional einen Port, falls der Standard-Port (12965) schon belegt ist. Die Ausgabe des Servers, samt möglicher Fehlermeldungen (welche u.a. verraten, ob der zu benutzende Port schon verwendet wird), wird in eine Log-Datei umgeleitet, die im angegeben Verzeichnis angelegt wird[22]. Ein Aufruf dieses Skriptes könnte folgendermaßen aussehen:

```
sh ▷ server+dir+port.sh /tmp/lform 11111
```

Dieser resultiert in folgendem Kommando (welches sich auch auf diese Weise direkt eingeben ließe):

```
server.php -port 11111 -liFedir /tmp/lform \
           -tmpdir /tmp/lform/download \
           -maxsize 2.0 -maxlength 8.0 \
           >> /tmp/lform/lf.log 2>&1 &
```

Es werden automatisch die Argumente `-tmpdir`, `-maxsize` und `-maxlength`

[22]Um die Ausgabe kontinuierlich beobachten zu können, benutzt man am besten das Kommando `tail` in einem gesonderten Teminal-Fenster:
`sh ▷ tail -f /tmp/lform/lf.log`

hinzugefügt, die durch Variablen innerhalb des Skriptes definiert sind und durch das Editieren desselbigen verändert werden können.

Durch `-maxsize` wird die maximale Größe (in Megabyte) der zu verabeitenden Datei bestimmt, durch `-maxlength` die Dauer (in Sekunden), auf die die Klangdateien, abhängig von ihrer Dauer, gekürzt, bzw. gestreckt werden. Dies betrifft auch die Initial-Version der liFe.f0rm (vgl. 4.1.5 auf Seite 25), sodaß sich über `-maxlength` die Anfangsdauer einer neuen liFe.f0rm initialisieren läßt.

4.1.3 Kommunikation mit dem Client

Wenn keine Probleme bei der Konfiguration auftreten, was zu einem Abbruch des Programms führt, wird der Server im nächsten Schritt mit der definierten Adresse in Form von `tcp://Host:Port` verknüpft, vorausgesetzt, daß diese nicht schon durch ein anderes Programm verwendet wird. Ab diesem Zeitpunkt wartet der Server auf eingehende Nachrichten, die ihm von einem Client übermittelt werden. Sobald eine Transmission eintrifft, also der Server auf seiner TCP-Adresse vom Client angesprochen wird, beginnt er mit der Verarbeitung dieser. Dabei wird als erstes die Transmission, die als eine durch Leerzeichen getrennte Liste *kodiert* ist, in ihre Einzelteile zerlegt:

```
$transmission = fread($conn, 8192);
$client_trans = explode(" ", $transmission);
for ($i=0; $i < count($client_trans); $i++) {
    $key = $client_trans[$i];
    $i++;
    $val = $client_trans[$i];
    $client[$key] = $val;
}
```

Die Funktion `fread()` liest die angegebene maximale Anzahl Bytes von der Verbindung `$conn` in die Variable `$transmission` ein, die Funk-

tion `explode()` zerlegt diese auf Basis des angegebenen Trennzeichens (hier das Leerzeichen) und schreibt das Ergebnis in einzelne Felder des Arrays `$client_trans`. In der `for`-Schleife wird dieser Array wiederum in einen neuen Array namens `$client` übertragen, der daraufhin die übermittelten Argument/Werte-Paare enthält (siehe Abbildung 7) und schlußendlich im weiteren Verlauf des Programms benutzt wird. Danach wird die Verbindung zum Client beendet sowie die ggf. gesetzten Optionen, die sich nun im `$client`-Array befinden, in entsprechende Felder des `$config`-Arrays (siehe Abbildung 3) übernommen.

```
$client: Array
(
    [file] => http://test.test/test.ogg
    [plugin] => grain
    [resize] => truncate
    [verbose] => 2
    [host] => 127.0.0.1
    [port] => 12965
    [remoteName] => 127.0.0.1
)
```

Abbildung 7: *Vom Server empfangene Client-Transmission*

4.1.4 Download der übermittelten Datei

Danach wird die Funktion `get_file()` (vgl. A.2, Seite 80 ff.) aufgerufen, mit der vom Client übergebenen Datei (`$client[file]`) als Argument. Diese Funktion gibt nach erfolgreichem Download [57] einen Array mit dem Pfad auf die lokal gespeicherte Datei sowie diverser gesammelter Informationen über dieselbige zurück. Diese Informationen umfassen u.a. die Dateigröße sowie den -Mimetyp, die vor dem Download mit den Konfigurations-Werten verglichen werden. Falls beispielsweise die Datei zu groß ist, oder der Mi-

metyp nicht mit den erlaubten übereinstimmt, wird kein Download durchgeführt und stattdessen eine entsprechende Fehlermeldung ausgegeben. Dies gilt ebenfalls für lokale Dateien, die nur kopiert werden, falls sie den zuvor genannten Kriterien entsprechen.

Die heruntergeladenen, bzw. kopierten Dateien werden in dem Verzeichnis `$config[tmpdir]` (vgl. Abbildung 3) gespeichert, welches über das Argument `-tmpdir` gesetzt werden kann.

4.1.5 Initial-Version

War der Download erfolgreich, wird als erstes überprüft, ob eine liFe.f0rm, deren Datei-Name und -Pfad in `$config[liFe]` (vgl. Abbildung 3) abgelegt ist, existiert. Der Pfad (`$config[liFedir]`) wird, wie schon zuvor erwähnt, entweder durch das Argument `-liFedir` oder durch den Standardwert definiert. Wird keine entsprechende Datei gefunden, wird die heruntergeladene Datei als „Initial-Version“ benutzt. Bevor sie diesen Status erreicht, muß sie folgende Stufen durchlaufen:

- sie wird erst in das WAVE-Format umgewandelt, falls ihr Mimetyp dem nicht entspricht
- auf die definierte Dauer (`$config[liFe_maxlength]`) gebracht (vgl. 4.1.7 auf Seite 29)
- normalisiert (vgl. 4.1.8 auf Seite 29)
- und schließlich in den Dateinamen, der in der Variablen `$config[liFe]` gesetzt ist, umbenannt.

Somit erblickt eine neue liFe.f0rm das binäre Licht der Welt.

4.1.6 Präparation & Statistiken

Existiert schon eine liFe.f0rm im angegebenen Verzeichnis, wird die heruntergeladene Datei für die weitere Verarbeitung durch die Plugins (vgl. 4.2 auf Seite 31) vorbereitet. Dafür durchläuft sie prinzipiell dieselben Schritte wie die Initial-Version, immer vorausgesetzt sie entspricht den geforderten Konfigurations-Werten:

- Umwandlung in das WAVE-Format
- Anpassung der Dauer (vgl. 4.1.7 auf Seite 29)
- Normalisierung (vgl. 4.1.8 auf Seite 29)

Zudem werden noch Statistiken, sowohl der neu zu verarbeitenden Datei, als auch der liFe.f0rm, mit Hilfe der Programme *SoX* und *sndinfo*[23] ermittelt. Diese enthalten u.a. die Samplerate, die Anzahl der Kanäle und die Dauer in Sekunden (vgl. Abbildung 8). Ich habe hierfür überwiegend mit Regulären Ausdrücken gearbeitet, mit denen ich die Ausgabe der Programme zerlegen und in entsprechende Array-Felder schreiben ließ.
Ein Beispiel anhand eines Programmaufrufs von *SoX*:

```
sh ▷ sox datei.wav -e stat

Samples read:            827370
Length (seconds):      9.380612
Scaled by:         2147483647.0
Maximum amplitude:     0.841522
Minimum amplitude:    -0.939178
Midline amplitude:    -0.048828
Mean    norm:          0.106867
Mean    amplitude:     0.000838
RMS     amplitude:     0.147116
```

[23] Aufruf über `csound -U sndinfo`

```
   Maximum delta:         0.729340
   Minimum delta:         0.000000
   Mean    delta:         0.087476
   RMS     delta:         0.116720
   Rough   frequency:         5568
   Volume adjustment:        1.065
```

Diese Ausgabe, welche programmintern in der Variablen `$sox_output` aufgefangen wird, wird mit Hilfe folgender Zeilen in für mich interessante Daten, die im Array `$parse` definiert sind, zerlegt:

```
$parse = array(
    "dur"       => "Length",
    "maxamp"    => "Maximum\s+amp",
    "minamp"    => "Minimum\s+amp",
    "roughfreq" => "Rough\s+freq",
    "voladjust" => "Volume\s+adj");

foreach ($parse as $statvar => $val) {
    if (preg_match("/\n$val.*:\s*(\S+)\b/i", $sox_output, $match)) {
        if ($verbose > 2) {
            echo "\t(sox-stat) $statvar => $match[1]\n";
        }
        $stats[$statvar] = $match[1];
    }
}
```

Dies resultiert in den entsprechenden Einträgen im Statistik-Array, wie in Abbildung 8 ersichtlich wird.

Die Anzahl der Kanäle wird für den nächsten Schritt gebraucht, bei dem die Dateien in einzelne Mono-Dateien aufgeteilt werden[24], falls diese selbstverständlich nicht schon einkanalig sind. Dies wird nötig, da die liFe.f0rm

[24]Momentan werden maximal Stereo-Dateien unterstützt

```
$stats: Array
(
    [srate] => 44100
    [channels] => 2
    [bit] => 16
    [length] => 9.38
    [headersize] => 44
    [datasize] => 1654740
    [samples] => 413685
    [dur] => 9.380612
    [maxamp] => 0.841522
    [minamp] => -0.939178
    [roughfreq] => 5568
    [voladjust] => 1.065
    [size] => 1654784
    [sizeKB] => 1616
    [sizeMB] => 1.578125
    [md5] => ae1025013a1a9b0a8240f9b4402f5064
    [sha1] => 2012d9f7f9cce45dea64c070139f0f249a18678c
    [mimetype] => audio/x-wav
)
```

Abbildung 8: *Ermittelte Statistiken einer Klangdatei*

stereo angelegt ist[25], um eine möglichst große „Abhör-Kompatibilität“ zum Benutzer zu gewährleisten.

4.1.7 Anpassung der Dauer

Für die Anpassung an die, in der Konfiguration festgelegten Dauer (vgl. Abbildung 3), stehen zwei Funktionen zur Verfügung: Zum einen `stretch_file_2_length()`, zum anderen `truncate_file_2_length()` (vgl. A.2, Seite 80 ff.). Bei beiden kommt das Programm *SoX* zum Einsatz, welches die an die Funktion übergebene Klangdatei[26] bearbeitet.

Welche der beiden Funktionen benutzt werden soll, wird über ein entsprechendes Client-Argument definiert (vgl. 4.3 auf Seite 54). Wird auf diesem Wege nichts, bzw. etwas nicht vorhandenes definiert, bsp. durch einen Schreibfehler bedingt, wird auf den Standard zurückgegriffen und die `stretch`-Funktion verwendet. Hierbei wird die Klangdatei, mittels eines „Time-Stretching“ Algorithmus [73], auf die angegebene Dauer gestreckt, falls sie zu kurz ist und gestaucht, falls sie zu lang ist. Dabei wird, wie bei diesem Algorithmus üblich, nur die Dauer, unabhängig von der Tonhöhe, die im Original belassen wird, geändert.

Wurde hingegen die `truncate`-Funktion gewählt, wird die Klangdatei auf die, als Argument an die Funktion mitübergebene, Dauer gebracht, indem sie beschnitten wird, falls sie zu lang sein sollte. Falls sie kürzer sein sollte, wird keine Aktion durchgeführt.

4.1.8 Normalisierung

In der Funktion `normalize()` (vgl. A.2, Seite 80 ff.), wird die als Argument übergebene Klangdatei normalisiert. Zuerst wird die Funktion

[25] Mehrkanaligkeit ist in Planung (vgl. 6.5 auf Seite 73)
[26] im WAVE-Format

`get_file_stats()` (vgl. 4.1.6 & A.2) aufgerufen, die die in Abbildung 8 aufgelisteten Statistiken über diese Datei bereitstellt. Diese Statistikwerte werden daraufhin mit den in der Konfiguration festgelegten Werten verglichen und ggf. Schritte eingeleitet, um den Vorgaben zu entsprechen. Im einzelnen werden folgende Eigenschaften überprüft:
Zuerst, ob die Samplerate mit der global definierten übereinstimmt,

```
if ($wav_stats[srate] != $config[sr]) {
    $arg[srate] = "-r $config[sr]";
    $arg[resample] = "resample";
}
```

darauf die Anzahl der Bits[27], die momentan unabhängig von der Konfiguration, fest auf `16 Bits` eingestellt ist,

```
if ($wav_stats[bit] != 16) {
    // -w -> 16 bit words
    // -s -> signed linear data encoding
    $arg[bit] = "-w -s";
}
```

und schließlich die Lautstärke, mit Hilfe des von *SoX* gelieferten `voladjust`-Wertes, von dem noch „0.2“ abgezogen werden, um möglichen Übersteuerungen vorzubeugen.

```
$gain = $wav_stats[voladjust] - 0.2;
if ($gain != 1.0) {
    $arg[vol] = "vol $gain";
}
```

[27]Mit der Einheit *Bit* [70] wird die Anzahl der zur Verfügung stehenden Amplitudenwerte definiert, die eine Wellenform repräsentieren. 16 Bits entsprechen dabei 2^{16} (65536) Werten, die sich zur Hälfte auf den positiven sowie den negativen Amplituden-Bereich aufteilen, also jeweils 32768 Werte umfassen.

Falls beispielsweise Samplerate und Lautstärke normalisiert werden müssen, wird folgendes *SoX*-Kommando, durch die eben erwähnten Schritte zusammengesetzt:

```
sox datei.wav.orig -r 44100 datei.wav resample vol 1.355
```

4.2 Plugins

Darauf folgt die eigentliche Klangverarbeitung, welche von den Plugins übernommen wird. Dem jeweiligen Plugin stehen, mit dem Aufruf der entsprechenden Funktion in der das Plugin *verpackt* ist, Informationen in Form von zwei Arrays zur Verfügung. Zum einen die, für die heruntergeladene und für die weitere Verarbeitung präparierte Datei (siehe Abbildung 9), zum anderen die Informationen über den aktuellen Zustand der liFe.f0rm (siehe Abbildung 10). Diese beiden Arrays werden der Plugin-Funktion als Argumente übergeben.

Desweiteren stehen den Server betreffende Informationen in den Arrays `$config` (vgl. Abbildung 3) und `$bin` (vgl. Abbildung 4) zur Verfügung. Diese müssen, bevor sie benutzt werden können, über folgenden Aufruf der Funktion *bekannt* gemacht werden:

```
function plugin_Funktion($liFe-Array, $file-Array) {
    global $config, $bin;
    ...
}
```

Zudem erwarten die Plugins, daß ihnen, in den übergebenen Arrays, vier Variablen zur Verfügung stehen, die die Pfade der entsprechenden Mono-Dateien (`$liFe[wavL], $liFe[wavR], $file[wavL], $file[wavR]`) enthalten[28]. Diese werden in den unten beschriebenen Plugins zu einer neuen

[28] Welche unter Umständen mit dem eigentlichen Dateinamen übereinstimmen, falls die Datei nur einkanalig ist.

```
$file: Array
(
    [remote] => /stuf/tmp/ogg/demo05.ogg
    [name] => demo05.ogg
    [local] => /tmp/zgrain/tmp/1096079786_demo05.ogg
    [size] => 170994
    [sizeKB] => 166.986328125
    [sizeMB] => 0.16307258605957
    [mimetype] => audio/x-ogg
    [md5] => a12e9dc1808e20ea0542645733d4ced2
    [sha1] => 04a1e612dd12012aea3f6668b606f5b89e5ca05a
    [wav] => /tmp/zgrain/tmp/lf_download_tmpfile.wav
    [basename] => lf_download_tmpfile.wav
    [dirname] => /tmp/zgrain/tmp
    [filename] => /tmp/zgrain/tmp/lf_download_tmpfile
    [wavL] => /tmp/zgrain/tmp/lf_download_tmpfile-L.wav
    [wavR] => /tmp/zgrain/tmp/lf_download_tmpfile-R.wav
    [wav_stat] => Array
        (
            [srate] => 44100
            [channels] => 2
            [bit] => 16
            [length] => 8.00
            [headersize] => 44
            [datasize] => 1411200
            [samples] => 352800
            [dur] => 8.000000
            [maxamp] => 0.909271
            [minamp] => -0.781586
            [roughfreq] => 7848
            [voladjust] => 1.100
            [size] => 1411244
            [sizeKB] => 1378.16796875
            [sizeMB] => 1.3458671569824
            [md5] => 863f79ccf6905a6718c6c37d311609a8
            [sha1] => 4e02d0688575c78ed1f3145bf886ba6ec7ebc66c
            [mimetype] => audio/x-wav
        )
)
```

Abbildung 9: *Array mit Informationen der heruntergeladenen Datei*

```
$liFe: Array
(
    [wav] => /tmp/zgrain/liFe.wav
    [alt] => /tmp/zgrain/1096079786.wav
    [basename] => liFe.wav
    [dirname] => /tmp/zgrain
    [filename] => /tmp/zgrain/liFe
    [wavL] => /tmp/zgrain/liFe-L.wav
    [wavR] => /tmp/zgrain/liFe-R.wav
    [wav_stat] => Array
        (
            [srate] => 44100
            [channels] => 2
            [bit] => 16
            [length] => 9.71
            [headersize] => 44
            [datasize] => 1713600
            [samples] => 428400
            [dur] => 9.714286
            [maxamp] => 0.746429
            [minamp] => -0.939026
            [roughfreq] => 7951
            [voladjust] => 1.065
            [size] => 1713644
            [sizeKB] => 1673.48046875
            [sizeMB] => 1.6342582702637
            [md5] => 21f4ae23c01f2d15ddc4b066d2ee5f93
            [sha1] => c5a1a44afb64c82f200a5f394b2798e01f120fbf
            [mimetype] => audio/x-wav
        )
    [neu] => /tmp/zgrain/liFe_new.wav
)
```

Abbildung 10: *Array mit Informationen der liFe.f0rm Datei*

Stereo-Datei verarbeitet.

Hat das Plugin seine Arbeit erfolgreich verrichtet, wird der Pfad der neu erzeugten liFe.f0rm (vgl. Abbildung 11) am Ende der Funktion mittels

```
return $conf[newliFe];
```

zurückgegeben. Der Server überprüft, ob tatsächlich eine solche Datei existiert und leitet die weiteren Schritte ein. Falls dem nicht so ist, also die Plugin-Funktion *gescheitert* ist, wird die liFe.f0rm in ihrem Stadium belassen und eine entsprechende Fehlermeldung ausgegeben. Ansonsten wird die alte liFe.f0rm in `$timestamp.wav` (vgl. 3.6 auf Seite 11) sowie die neue in `liFe.wav` (`$liFe[wav]`) umbenannt, diese wiederum in das Ogg-Vorbis-Format enkodiert (`liFe.ogg`) sowie eventuell vorhandene temporäre Dateien gelöscht.

4.2.1 grain

Wie in Abbildung 11 ersichtlich, wird zur Konfiguration des Plugins „grain“ ebenfalls ein Array (namens `$conf`) benutzt, der nur innerhalb des Plugins Gültigkeit hat und als Argument an die *Sub-Funktionen*, wie beispielsweise `generate_grain_orc()`, übergeben wird, damit diese über alle notwendigen Informationen verfügen können (vgl. B.3.2, Seite 109 ff.). Dieser Array enthält u.a. die Definition für die Gesamtdauer der resultierenden Datei (`$conf[TotalDur]`), die aus der aktuellen Dauer der liFe.f0rm, der Dauer der heruntergeladenen Datei sowie dem in der Variablen `$conf[growing]` gesetzten Wert errechnet wird:

```
$conf[growing]  = $conf[timestamp] * 0.0000000002;
$file_durations = $conf[liFe_dur] + ($conf[usr_dur] * 0.0001);
$conf[TotalDur] = $file_durations + $conf[growing];
```

Wie daraus ersichtlich wird, wird der Wert für `$conf[growing]` auf Basis des aktuellen Timestamps, also der Zeit in Sekunden seit dem 01.01.1970, berechnet. Dies bedeutet, daß dieser Wert mit der Zeit wächst, also einer Art Wachstum der liFe.f0rm gleichzusetzen ist. Vereinfacht gesprochen, wird die liFe.f0rm dadurch mit der Zeit, bzw. bei jeder erneuten Klangverarbeitung, immer länger.

Nachdem die Konfigurations-Datei `plugin_grain_config.inc.php` (vgl. B.3.1 auf Seite 107) eingelesen wurde, beginnt der Prozess der Klangverarbeitung mit dem Programm *Csound*. Dieses erwartet eine Orchester-Datei (`datei.orc`), welche die Definition der Klangerzeugung, das sogenannte *Instrument*, beinhaltet sowie eine Partitur-Datei (`datei.sco`), in der Anweisungen enthalten sind, wann und mit welchen Parametern, wie beispielsweise Tonhöhe und Dauer, die Klangerzeugung erfolgen, also das Instrument aktiviert werden soll. Auf Basis dieser beiden Dateien, die dem Programm als Kommandozeilen-Argumente übergeben werden, wird eine, ebenfalls über ein Argument definierte, Klangdatei generiert.

Der im folgenden beschriebene Ablauf des Plugins läßt sich dem Flußdiagramm in Abbildung 12 entnehmen.

Die Funktion `generate_grain_orc()` (vgl. B.3.2, Seite 109 ff.) ist für die Erzeugung der Orchester-Datei zuständig. Zuerst werden die nötigen Definitionen für die resultierende Datei, welche u.a. die in der Server-Konfiguration (vgl. 4.1.1 auf Seite 14) global gesetzte Samplerate (`$config[sr]`) sowie die Anzahl der Kanäle enthalten, in die in der Konfigurations-Variablen `$conf[usr_orc]` definierten Datei geschrieben. Desweiteren wird das klangerzeugende Csound-Instrument (`instr 1`) in die Orchester-Datei geschrieben, welches folgende Parameter[29] von der Partitur-Datei, in der diese in

[29] sogenannte „p-Felder“

```
$conf: Array
(
    [usr_orc] => /tmp/test/download/lf_download_tmpfile.orc
    [usr_sco] => /tmp/test/download/lf_download_tmpfile.sco
    [usr_out] => /tmp/test/download/lf_download_tmpfile_GenEratEd.wav
    [usr_outL] => /tmp/test/download/lf_download_tmpfile_GenEratEd-L.wav
    [usr_outR] => /tmp/test/download/lf_download_tmpfile_GenEratEd-R.wav
    [liFe_out] => /tmp/test/liFe.wav
    [mix_orc] => /tmp/test/download/grain_mix.orc
    [mix_sco] => /tmp/test/download/grain_mix.sco
    [newliFe] => /tmp/test/liFe_new.wav
    [liFe_wavL] => /tmp/test/liFe-L.wav
    [liFe_wavR] => /tmp/test/liFe-R.wav
    [usr_wavL] => /tmp/test/download/lf_download_tmpfile-L.wav
    [usr_wavR] => /tmp/test/download/lf_download_tmpfile-R.wav
    [liFe_dur] => 8.063492
    [usr_dur] => 8.000000
    [liFe_samples] => 355600
    [usr_samples] => 352800
    [liFe_ftableSize] => 524288
    [usr_ftableSize] => 524288
    [timestamp] => 1097008800
    [growing] => 0.21940176
    [TotalDur] => 8.28369376
    [sr] => 44100
    [ksmps] => 100
    [kr] => 441
    [GenEratE_bin] => ../organs/GenEratE
    [soxmix_bin] => ../organs/soxmix
    [verbose] => 2
)
```

Abbildung 11: *Konfiguration des Plugins „grain“*

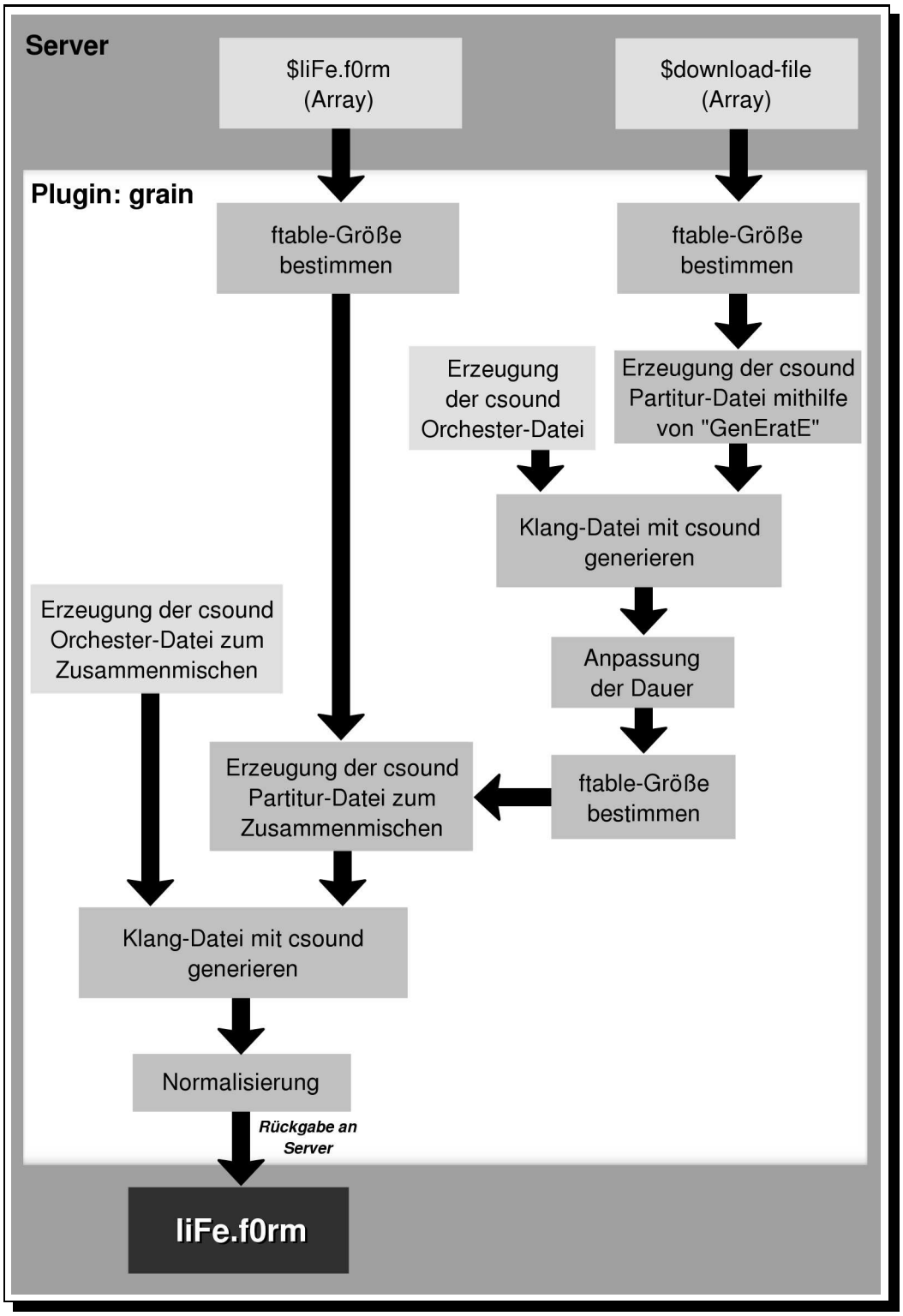

Abbildung 12: *Flußdiagramm Plugin „grain“*

einer Zeile von links nach rechts aufgelistet werden, erwartet:

1. das Instrument, welches benutzt werden soll
2. der Startzeitpunkt (in Sekunden)
3. die Dauer des zu erzeugenden *Events* (in Sekunden)
4. die Amplitude (in dB)
5. die Startposition in der Klangdatei, ab der ausgelesen werden soll (in Samples)
6. die Transpositions-Stufe (ein Wert von 1.0 entspricht der originalen Transposition)
7. die *Attack*-Zeit der Amplituden-Hüllkurve [68] (in Sekunden)
8. die *Release*-Zeit der Amplituden-Hüllkurve (in Sekunden)
9. die Panorama-Position (0.0 = links, 1.0 = rechts)

Auf Basis dieser Parameter wird der mit einer Amplituden-Hüllkurve versehene Sample-Abschnitt, mit Hilfe des Csound-Opcodes `tablei` [44], in ein entsprechendes Klang-Event umgesetzt.

Für die Erzeugung der Partitur-Dateien wird im ersten Schritt die Größe[30] der `ftable` bestimmt, die dazu dienen, die jeweilige zu verarbeitende Klangdatei Csound-intern verfügbar zu machen. Die Größe dieser muß im Vorfeld bestimmt werden, damit entsprechender Speicherraum reserviert, also die Klangdatei in ihrer gesamten Länge in den Hauptspeicher eingelesen, bzw. in dem `ftable` abgelegt werden kann. Der jeweilige `ftable` wird mit dem oben erwähnten `tablei` im Orchester-Instrument ausgelesen.

[30] Anzahl der Samples

Nach der Ermittlung der `ftable`-Größeninformation wird die Funktion `generate_grain_sco()` aufgerufen, die für die Erzeugung der Partitur-Datei verantwortlich ist. In dieser wird als erstes der *Kopf* der Partitur-Datei (gesetzt in der Variablen `$conf[usr_sco]`) in dieselbige geschrieben, in der die `ftable`, mit der zuvor berechneten Größe sowie den zugehörigen Dateinamen des linken sowie des rechten Kanals, stehen:

```
f2 0 $conf[usr_ftableSize] 1 "$conf[usr_wavL]" 0 4 1
f3 0 $conf[usr_ftableSize] 1 "$conf[usr_wavR]" 0 4 1
```

Da ich die Klangdateien als eine Aneinanderreihung von Chromosomen betrachte (vgl. 2.1 auf Seite 3), wird im nächsten Schritt ein Array namens `$ChromosomLaengen` mit 23 Werten definiert, die auf der relativen Länge der menschlichen Chromosomenpaare basieren und deren Summe annähernd 1.0 ergibt[31]:

```
$ChromosomLaengen=array(
    0.083, 0.072, 0.067,
    0.061, 0.061,
    0.056, 0.050, 0.044, 0.050, 0.044, 0.044, 0.039,
    0.033, 0.033, 0.033,
    0.033, 0.028, 0.028,
    0.022, 0.022,
    0.022, 0.017,
    0.056);
```

Die Array-Elemente werden nach einer zufälligen Neuanordnung (*Rekombination*) dafür benutzt, um einen weiteren Array namens `$usrSampleGrenze` zu füllen:

```
shuffle($ChromosomLaengen);
```

[31] Der Wert „1.0“ entspricht der relativen Gesamtlänge (100%) der Klangdatei.

```
$usrSampleGrenze[0]=0;
$i=1;
foreach($ChromosomLaengen as $length) {
    $uSmpLg = $length * $conf[usr_samples]; // [samples]
    $usrSampleGrenze[$i] = round($usrSampleGrenze[$i-1] + $uSmpLg);
    $i++;
}
```

Somit erhält man eine Aufteilung der gesamten Klangdatei in 23 Sample-Bereiche (vgl. Abbildung 13), die durch den `$ChromosomLaengen`-Array vordefiniert wurden. Der erste sowie der letzte Wert entsprechen dabei dem ersten und dem letzten Sample der Klangdatei.

Die eigentlichen Klang-Events werden mit Hilfe eines Programms namens „GenEratE“ erzeugt, welches ich in der Programmiersprache *C* [5] geschrieben habe (vgl. B.3.3, Seite 120 ff.). Dieses produziert auf der Basis der ihm übergebenen Argumente eine, durch die Gesamtdauer definierte, Anzahl von Ausschnitten („grains“) der Klangdatei in Form von Csound-Partitur-Events, die wiederum durch das oben beschriebene Csound-Instrument in tatsächlichen Klang umgesetzt werden. Folgende Argumente sind für den Aufruf von *GenEratE* nötig:

- Start-Sample
- End-Sample
- Anzahl der gleichzeitig erklingenden, granularen Stränge
- Start-Zeitpunkt (in Sekunden)
- Gesamte Dauer (in Sekunden)
- Untere Grenze der Transpositionsstufe

```
$usrSampleGrenze: Array
(
    [0] => 0
    [1] => 29282
    [2] => 54684
    [3] => 78322
    [4] => 99843
    [5] => 121364
    [6] => 141121
    [7] => 158761
    [8] => 174284
    [9] => 191924
    [10] => 207447
    [11] => 222970
    [12] => 236729
    [13] => 248371
    [14] => 260013
    [15] => 271655
    [16] => 283297
    [17] => 293175
    [18] => 303053
    [19] => 310815
    [20] => 318577
    [21] => 326339
    [22] => 332337
    [23] => 352800
)
```

Abbildung 13: *Errechnete „Samplegrenzen-Chromosome“ der heruntergeladenen Datei*

- Obere Grenze der Transpositionsstufe
- Untere Grenze der Amplitude (in dB)
- Obere Grenze der Amplitude (in dB)
- Panorama-Position
- *Random-Seed*[32]
- Samplerate (zur Berechnung des einzelnen Ausschnitts)
- Untere Grenze der *grain*-Dauer (in Sekunden)
- Obere Grenze der *grain*-Dauer (in Sekunden)
- *grain*-Attack (in Sekunden)
- *grain*-Release (in Sekunden)

GenEratE wird, der Anzahl der zuvor errechneten Chromosom-Abschnitte entsprechend oft (vgl. Abbildung 13), innerhalb einer Schleife aufgerufen, in der teilweise die oben aufgezählten Argumente jeweils neu gesetzt werden. Beispielsweise werden die in Abbildung 14 aufgelisteten Werte, die während des 7. Schleifen-Durchlaufs erzeugt wurden, benutzt, um die 16 Kommandozeilen-Argumente für folgenden Aufruf von *GenEratE* zusammenzusetzen:

```
GenEratE 158761 174284 40 2.0660773466667 0.34515390666667 \
         0.4 0.7 60 88 0.5 1097008800 44100 0.070399092970522 \
         0.1759977324263 0.028159637188209 0.028159637188209
```

[32]Initialwert für den Zufallsgenerator

Die daraus resultierende Ausgabe des Programms, welche die erzeugten Csound-Events beinhaltet, wird in einem Array aufgefangen und mittels der, in der Datei `plugin_functions.inc.php` (vgl. B.2 auf Seite 105) definierten, allen Plugins zur Verfügung stehenden Funktion `Array2File()` an die Csound-Partitur-Datei angefügt.

```
[7] $arg: [[altered values]] Array
(
    [MinSample] => 158761
    [MaxSample] => 174284
    [GrainstreamNumber] => 40
    [StartTime] => 2.0660773466667
    [TotalDur] => 0.34515390666667
    [MinTrans] => 0.4
    [MaxTrans] => 0.7
    [MinAmp] => 60
    [MaxAmp] => 88
    [Pan] => 0.5
    [Seed] => 1097008800
    [Samplerate] => 44100
    [MinDur] => 0.070399092970522
    [MaxDur] => 0.1759977324263
    [Attack] => 0.028159637188209
    [Release] => 0.028159637188209
)
```

Abbildung 14: *Argumente für den Aufruf des C-Programms „GenEratE“*

Wurden die Orchester- sowie die Partitur-Datei erzeugt, erfolgt der Aufruf von Csound, also der Generierung der ersten Klangdatei, welche die oben beschriebene, granulare Verarbeitung der heruntergeladenen Datei beinhaltet. Die Dauer der neu erzeugten Datei wird daraufhin mit Hilfe der Funktion `stretch_file_2_length()` (vgl. A.2, Seite 80 ff.) auf den, in der Plugin-Konfigurations-Variablen `$conf[TotalDur]` definierten Wert gebracht, damit die unter Umständen zu kurz oder zu lang geratene Datei in ihrer Ge-

samtheit weiterverarbeitet werden kann.

Denn im nächsten Schritt wird die zuvor neu erzeugte Datei mit der vorhandenen liFe.f0rm auf granularer, bzw. *genetischer* Basis vereinigt. Dazu werden abermals die Größe der `ftable` der beiden Dateien benötigt, da die Vorgehensweise sehr ähnlich zu der zuvor beschriebenen ist[33].

Das Csound-Instrument, welches von der (Plugin-Sub-)Funktion `generate_grainMix_orc()` (vgl. B.3.2, Seite 109 ff.) in die Orchester-Datei (`$conf[mix_orc]`) geschrieben wird, erwartet ebenfalls neun Parameter, die soweit identisch sind, wobei jedoch kein p-Feld für das Panorama vorhanden ist, da dieses durch das vorhandene Klangmaterial selber bestimmt wird. Anstelle dessen wird über das neunte p-Feld die Auswahl der `ftable`, also der Datei die erklingen soll, gesetzt:

```
isamplL     = p9          ; ftable links
isamplR     = isamplL + 2 ; ftable rechts
```

Die Funktion `generate_grainMix_sco()` ist analog dazu verantwortlich für die Erzeugung der Csound-Partitur-Datei (`$conf[mix_sco]`). Diese schreibt zuerst die `ftable` mit Größe und Dateinamen für die zu verwendenden Klangdateien in den Kopf der Partitur-Datei:

```
f2 0 $conf[liFe_ftableSize] 1 "$conf[liFe_wavL]" 0 4 1
f4 0 $conf[liFe_ftableSize] 1 "$conf[liFe_wavR]" 0 4 1

f3 0 $conf[usr_out_ftablesize] 1 "$conf[usr_outL]" 0 4 1
f5 0 $conf[usr_out_ftablesize] 1 "$conf[usr_outR]" 0 4 1
```

Ist dies geschehen, wird solange eine `while`-Schleife durchlaufen, bis die in der Variablen `$conf[TotalDur]` (s.o.) definierte Dauer erreicht ist. In dieser Schleife werden die, aus den oben erwähnten neun p-Felder bestehenden,

[33] Es wird dabei jedoch nicht das Programm *GenEratE* genutzt.

Partitur-Events erzeugt, wobei ein Schleifen-Durchlauf der Erzeugung eines Events entspricht. Auf Basis von vorher definierten Minima und Maxima, wird den Parametern Dauer und Amplitude ein zufälliger Wert zugewiesen. Die Werte für die Dauer bewegen sich dabei im Bereich von 100 bis 250 Millisekunden. Auf Basis dieser generierten Werte werden die übrigen Parameter, wie Attack, Release, Startzeitpunkt oder die Sample-Startposition, bei jedem Durchlauf neu errechnet. Für den Parameter `ftable` wird ebenfalls der Zufall hinzugezogen und entweder der Wert „2“ oder „3“ ausgewählt, was wie oben ersichtlich der einen oder der anderen Datei entspricht. Die erzeugten Events werden in einen Array geschrieben, der nach Beendigung der `while`-Schleife, wiederum mit der Funktion `Array2File()`, in die Csound-Partitur-Datei geschrieben wird.

Somit findet also eine Zufallsauswahl beider Dateien auf granularer Ebene statt, die mittels eines weiteren Aufrufs von Csound mit der zuvor erzeugten Orchester- sowie Partitur-Datei, in der neuen Generation der liFe.f0rm resultiert. Diese neu erzeugte Datei wird, bevor sie schließlich an den Server zurückgegeben wird, mit der in 4.1.8 beschriebenen Funktion normalisiert.

4.2.2 pvCross

Das Plugin „pvCross“ basiert ebenfalls auf einer Klangverarbeitung durch das Programm Csound, wie dem Flußdiagramm in Abbildung 15 zu entnehmen ist. Hierbei kommt der Csound-Opcode *pvcross* zum Einsatz, der zur Gruppe der Resynthese-Generatoren gehört. Das heißt, die Klangdateien werden nicht als solche sampleweise eingelesen und verarbeitet, also als Amplitudenwerte über der Zeit behandelt, sondern es werden vorher Analyse-Dateien mit dem Programm *pvanal*[34] erstellt. Bei der Analyse handelt es sich um ei-

[34] Aufruf über `csound -U pvanal`

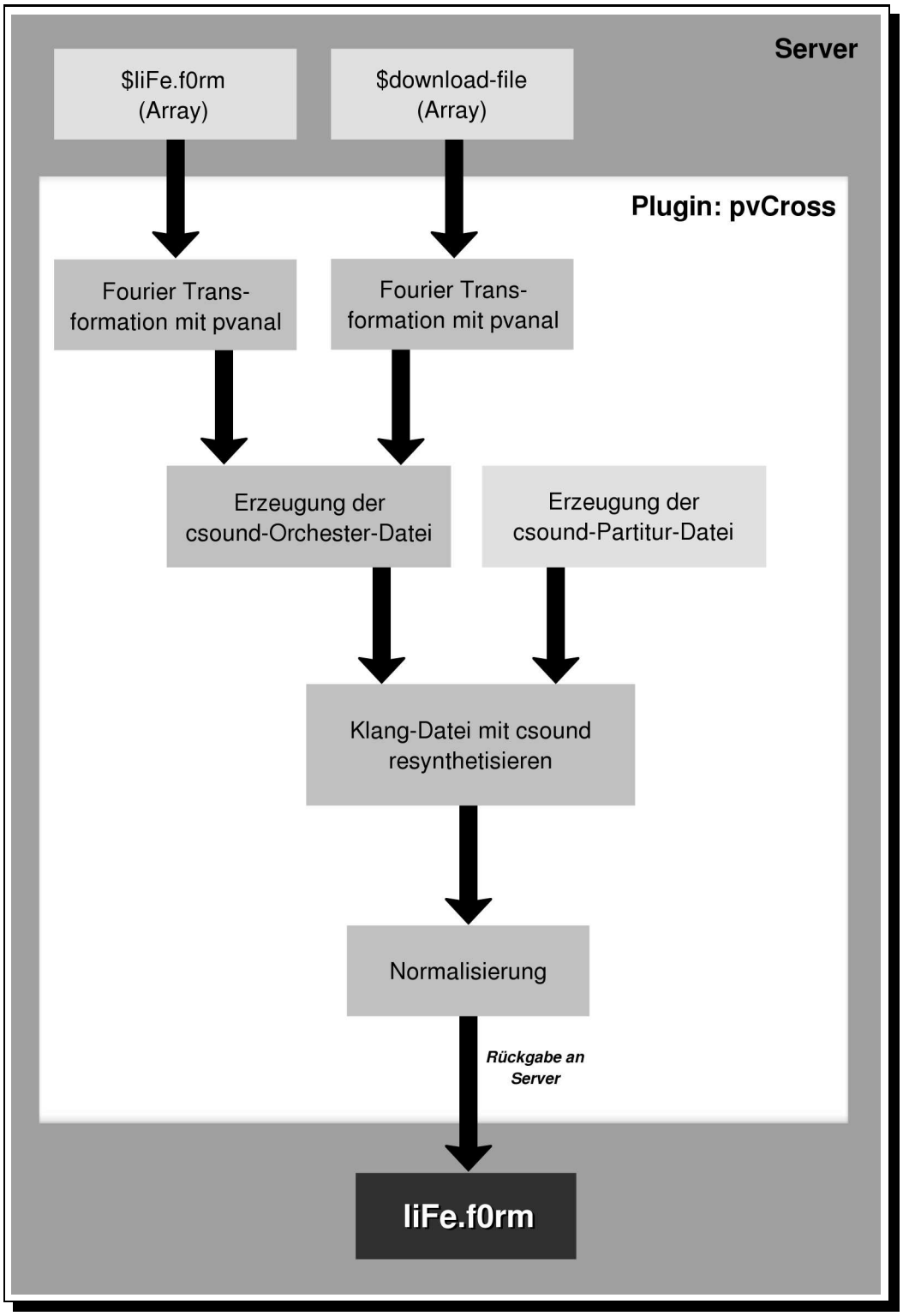

Abbildung 15: *Flußdiagramm Plugin „pvCross“*

ne *Fourier Transformation* [71], die in eine Repräsentation der Klangdatei in Form von Frequenzen über der Zeit resultiert. Diese Analyse-Informationen werden im Csound-Instrument eingelesen und zur Resynthese eines neuen Klanges benutzt.

Bevor dies geschieht, wird, wie auch bei dem Plugin „grain“ (vgl. 4.2.1 auf Seite 34), die Konfigurations-Datei `plugin_pvCross_config.inc.php` (vgl. B.4.1 auf Seite 127), in der der Plugin-interne Array `$conf` mit Informationen gefüllt wird, eingelesen. Dieser enthält, wie in Abbildung 16 ersichtlich, neben weiteren Konfigurations-Werten die notwendigen Dateinamen der Analyse-Dateien (*datei.pv*) sowie die Gesamtdauer der resultierenden Datei (`$conf[TotalDur]`), die auf die in 4.2.1 beschriebene Art errechnet wird.

```
$conf: Array
(
    [usr_pvL] => /tmp/test/download/lf_download_tmpfile-L.pv
    [usr_pvR] => /tmp/test/download/lf_download_tmpfile-R.pv
    [liFe_pvL] => /tmp/test/liFe-L.pv
    [liFe_pvR] => /tmp/test/liFe-R.pv
    [orc] => /tmp/test/download/pvCross.orc
    [sco] => /tmp/test/download/pvCross.sco
    [newliFe] => /tmp/test/liFe_new.wav
    [liFe_channels] => 2
    [usr_channels] => 2
    [liFe_dur] => 10.208617
    [usr_dur] => 8.000000
    [growing] => 0.2195041438
    [TotalDur] => 10.4289211438
    [sr] => 44100
    [ksmps] => 100
    [kr] => 441
    [verbose] => 2
)
```

Abbildung 16: *Konfiguration des Plugins „pvCross“*

Daraufhin erfolgt die Erzeugung der notwendigen pv-Analyse-Dateien der liFe.f0rm sowie der heruntergeladenen Datei, mit Hilfe der Funktion `pvanal()`, die die Datei `plugin_functions.inc.php` (vgl. B.2 auf Seite 105) beherbergt. Die Analyse-Dateien werden für die Csound-Orchester-Datei (`$conf[orc]`) benötigt, die durch den Aufruf der Funktion `generate_pvCross_orc()` (vgl. B.4.2, Seite 128 ff.) im nächsten Schritt generiert wird. Hierfür wird zuerst ein Chromosom-Array mit 23 Werten, die wiederum auf der relativen Länge der menschlichen Chromosomenpaare basieren (vgl. 2.1 auf Seite 3), gefüllt, deren Reihenfolge mit der Funktion `shuffle()` zufällig geändert wird (vgl. Abbildung 17).

Auf Basis der Summe (`$sum`) aller Chromosom-Werte werden diese in Werte aufsteigend von 0 bis 1 umgesetzt und in den neuen Array `$ChromoIndx` geschrieben:

```
$sum=0;
foreach ($chromosom as $value) { $sum = $sum + $value; }
$x=0;
foreach ($chromosom as $indx => $wert) {
    $x = $x + $wert;
    $ChromoIndx[$indx] = ($x/$sum);
}
```

Dies resultiert in dem in Abbildung 18 dargestellten Array, dessen letzter Wert „1“ ist, da die Summe gleich dem hochgezählten Wert `$x` ist. Diese Indexwerte repräsentieren die Position in der jeweiligen pv-Analyse-Datei, wobei „0“ dem Anfang und „1“ dem Ende der Klangdatei entspricht. Sie werden innerhalb des Csound-Instruments zur Generierung einer Hüllkurve mit dem Opcode *linseg* genutzt:

```
$chromosom: Array
(
    [0] => 3
    [1] => 6
    [2] => 6
    [3] => 4
    [4] => 10
    [5] => 6
    [6] => 8
    [7] => 11
    [8] => 7
    [9] => 4
    [10] => 5
    [11] => 6
    [12] => 5
    [13] => 9
    [14] => 8
    [15] => 9
    [16] => 8
    [17] => 13
    [18] => 10
    [19] => 11
    [20] => 12
    [21] => 4
    [22] => 15
)
```

Abbildung 17: *Chromosom-Array des Plugins „pvCross“*

```
fwrite($orc_file, "kindx linseg 0, ");
for($i=0; $i<$ChromosomAnzahl-1; $i++) {
    fwrite ($orc_file, "idur/$ChromosomAnzahl, $ChromoIndx[$i], ");
}
fwrite ($orc_file, "idur/$ChromosomAnzahl, 1");
```

Der in der `for`-Schleife so erzeugte `kindx` wird im nächsten Schritt mit den Gesamtdauern der beiden Klangdateien multipliziert, was in *realen* Sekunden-Zeitangaben für die jeweilige Datei resultiert:

```
kfiLePointer = kindx * ifiLeDur
kUserPointer = kindx * iUserDur
```

Ansonsten erwartet das Csound-Instrument fünf p-Felder aus der Partitur-Datei:

1. das Instrument, welches benutzt werden soll
2. der Startzeitpunkt (in Sekunden)
3. die Dauer des zu erzeugenden Events (in Sekunden)
4. die Lautstärke (ein Wert von 1.0 entspricht der originalen Lautstärke)
5. der zu verwendende `ftable` (*iCrossTable*)

Der in der Partitur-Datei befindliche *iCrossTable* (s.u.) enthält die Werte, die die „Kreuzung“ der beiden Klangdateien darstellt. Es handelt sich hierbei um Lautstärke-Werte, die sich auf beide Klangdateien auswirken, wobei der Wert „0“ dem ausschließlichen Erklingen der liFe.f0rm und der Wert „1“ der heruntergeladenen Datei entspricht:

```
iTabSize = 23   ; Elemente-Anzahl von iCrossTable

kphasor phasor 1/idur  ; ein table-Durchgang in p3 sec
```

```
$ChromoIndx: Array
(
    [0] => 0.016666666666667
    [1] => 0.05
    [2] => 0.083333333333333
    [3] => 0.10555555555556
    [4] => 0.16111111111111
    [5] => 0.19444444444444
    [6] => 0.23888888888889
    [7] => 0.3
    [8] => 0.33888888888889
    [9] => 0.36111111111111
    [10] => 0.38888888888889
    [11] => 0.42222222222222
    [12] => 0.45
    [13] => 0.5
    [14] => 0.54444444444444
    [15] => 0.59444444444444
    [16] => 0.63888888888889
    [17] => 0.71111111111111
    [18] => 0.76666666666667
    [19] => 0.82777777777778
    [20] => 0.89444444444444
    [21] => 0.91666666666667
    [22] => 1
)
```

Abbildung 18: *Chromosom-Index-Array des Plugins „pvCross“*

```
kcross  tablei  kphasor*iTabSize, iCrossTable

kfiLeAmp = 1-kcross ; 0 => fiLe
kUserAmp = kcross   ; 1 => user-File
```

Schließlich münden die zuvor generierten k-Variablen im Aufruf von *pvbufread* und *pvcross*, was zur Resynthese, also der eigentlichen Klangerzeugung führt:

```
     pvbufread kUserPointer, "$conf[usr_pvL]"
autL pvcross kfiLePointer, kfmod, "$conf[liFe_pvL]", kUserAmp, kfiLeAmp

     pvbufread kUserPointer, "$conf[usr_pvR]"
autR pvcross kfiLePointer, kfmod, "$conf[liFe_pvR]", kUserAmp, kfiLeAmp
```

Die Funktion `generate_pvCross_sco()` (vgl. B.4.2, Seite 128 ff.) generiert die Csound-Partitur-Datei, deren Dateiname in der Variablen `$conf[sco]` gesetzt ist. Sie besteht im wesentlichen aus nur einem Event, dessen Dauer der gesamten der resultierenden Datei (`$conf[TotalDur]`) entspricht sowie dem `ftable` (*iCrossTable*). Dieser `ftable` wird mit Hilfe der Zufalls-Funktion `mt_rand()` mit 23 Werten zwischen „0“ und „1“ gefüllt. Da, wie oben erwähnt, der Wert „0“ der liFe.f0rm sowie der Wert „1“ der heruntergeladenen Datei entspricht, kommt diese Auswahl einer zufälligen Vererbung der „Klang-Chromosome“ an die resultierende Datei gleich.

Nach der Erzeugung der Orchester- sowie der Partitur-Datei erfolgt der Aufruf von Csound, dessen Produkt in Form der neuen liFe.f0rm, nachdem diese ggf. normalisiert wurde, an den Server zurückgegeben wird.

4.2.3 template

Das Plugin „template“ (vgl. B.5.2 auf Seite 134) stellt eine Vorlage zur Implementation eines Plugins dar. Es ist durchaus *funktionsfähig* und läßt

sich somit auch als zu benutzendes Plugin mit dem entsprechenden Client-Argument (vgl. 4.3 auf Seite 54) aufrufen. Wenn dies geschieht, wird keine Klangverarbeitung, also demnach auch keine Veränderung der liFe.f0rm vorgenommen, sondern es werden alle zur Verfügung stehenden Variablen, wie u.a. die Statistiken der Dateien, ausgegeben. Somit erhält man einen Überblick über die vom Server bereitgestellten im Plugin nutzbaren Informationen, in Form der entsprechenden Arrays.

4.3 Der Client Prototyp

Der *Client* stellt die Schnittstelle zum Benutzer dar, über den Anweisungen an den jeweiligen Server übermittelt werden können (vgl. Abbildung 2). Für die Verbindung zum Server wird das Protokoll *TCP/IP* [63] genutzt, welche auf einer Adressierung in der Form „host:port“ beruht (vgl. 4.1.3 auf Seite 23). Falls diese nicht über die Argumente `-host` und `-port` beim Client-Aufruf definiert werden, werden die dafür gesetzten Standardwerte „127.0.0.1“ sowie „12965“ verwendet. Das heißt, es wird automatisch eine Verbindung zum auf demselben Rechner (dem sogenannten *localhost*) aktiven Server, der standardmäßig auf den Port „12965“ hört, etabliert. Sobald man einen Server auf einem anderen Rechner ansprechen möchte, muß man dies dem Client über die oben genannten Argumente explizit mitteilen, wobei die IP-Nummer durch einen entsprechenden Domain-Namen ersetzt werden kann[35]. Im einfachsten Fall sieht ein Client-Aufruf folgendermaßen aus[36]:

```
sh ▷ client.sh http://beispiel.domain/datei.ogg
```

Hiermit wird der Server angewiesen, die übergebene Datei herunterzuladen und für die Verarbeitung durch das Standard-Plugin vorzubereiten (vgl. 4.1.4 ff.). Möchte man ein anderes Plugin für die Klangverarbeitung definieren, erfolgt dies über das Argument `-plugin`[37].

Weiteren Einfluß auf die Handhabung der Klangdateien, läßt sich mit dem Argument `-resize` nehmen. Hierdurch wird der Server angewiesen, eine bestimmte Funktion zur Längen-Anpassung der Dateien zu

[35] **sh** ▷ `client.sh -host eine.beispiel.domain -port 12345 ...`

[36] Das in den Beispielen benutzte Shell-Skript *client.sh* dient dazu, das (die eigentliche Arbeit verrichtende) PHP-Skript *client.php* aus jedem beliebigen Verzeichnis aufrufen zu können.

[37] Alle implementierten Plugins lassen sich über das folgende Server-Kommando anzeigen: **sh** ▷ `server.sh -show plugins`

verwenden, falls diese der vorgegebenen maximalen Dauer nicht entsprechen. Zwei stehen dabei zur Auswahl: zum einen die standardmäßig benutzte Funktion `stretch_file_2_length()`, zum anderen die Funktion `truncate_file_2_length()` (vgl. 4.1.7 auf Seite 29). Die Verwendung dieser lassen sich analog dazu durch `-resize stretch`, bzw. `-resize truncate` mit dem Client-Aufruf definieren.

Hier ein Kommando-Beispiel zur Verwendung der `truncate`-Funktion und dem Plugin `pvCross`, mit einer lokal auf dem Server-Rechner befindlichen Datei:

```
sh ▷ client.sh -resize truncate -plugin pvCross /tmp/datei.ogg
```

Wie aus diesem Beispiel ersichtlich wird, stehen mehrere „Klangquellen“ in Form von drei unterstützten Protokollen zur Auswahl, über diese der Server auf Dateien zugreifen kann:

1. Hypertext Transfer Protocol [64] (`http://beispiel.domain/datei`)
2. File Transfer Protocol[38] [65] (`ftp://beispiel.domain/datei`)
3. Lokale Dateien (`file:///lokale/datei`, bzw. `/lokale/datei`)

Ein weiteres erwähnenswertes Argument ist `-verbose`. Hiermit wird die Ausgabe des Servers über einen *Integer*-Wert[39] gesteuert. Je größer der Wert, desto mehr Informationen werden ausgegeben. Dies dient überwiegend zur Auffindung von Programm-Fehlern, bzw. dem Verständnis für den Ablauf der vom Server ausgeführten Arbeitsschritte, die Verarbeitung der Klangdateien betreffend.

[38] noch nicht vollständig implementiert
[39] Ganzzahliger Wert

In Abbildung 19 sind noch einmal zusammenfassend alle momentan dem Aufruf des Clients zur Verfügung stehende Argumente aufgelistet[40].

```
Usage:  ./client.php [OPTION]... <URL> [OPTION]...

options:
        --host HOST      transmit URL to server on HOST
                         (default: 127.0.0.1)
        --port PORT      connect to PORT on host
                         (default: 12965)
        --plugin PLUGIN  use PLUGIN for soundprocessing
        --resize MODE    use 'truncate' or 'stretch' to resize
                         the sound-files (default: stretch)
        --email ADR      server sends email to ADR when
                         soundprocessing finished
        --ftpuser NAME   use NAME to log in ftp-Server
                         if this option is not given,
                         anonymous ftp will be used
                         (URL should certainly start with 'ftp://')
        --verbose INT    set verbose level for client AND server
                         (as INTeger number > 0)
        --help, -h       display this help and exit

<URL>:  http://some.server.org/file
        ftp://some.server.org/file
        ftp://user:password@some.server.org/file (NOT recommended!)
        file:///path/to/local/file
        /path/to/local/file

Note:   local file has to be on the server's host!
        file upload only available with the Web-client (under construction)


currently supported filetypes: MPEG Layer I/II/III (mp3), ogg-vorbis, wav
```

Abbildung 19: *Kommandozeilen-Argumente für den Client*

[40] vgl. **sh** ▷ `client.sh -help`

4.4 Bootfähige Morphix-liFe-CD

Die beiligende CD enthält alle zuvor beschriebenen Programme, deren Quelltexte sowie Klangbeispiele, die während diverser Testdurchläufe entstanden sind. Zudem sind noch Dateien im Ogg/Vorbis-Format vorhanden, die zur eigenen „Erstellung“ einer liFe.f0rm verwendet werden können.

Es handelt sich um eine ausschließlich von CD laufende GNU/Linux-Distribution, die sich „Morphix“ [48] nennt, welche wiederum auf „Knoppix“ [49] basiert. Bei diesen sogenannten „Live-CD's“ ist es üblich, daß die im Rechner vorhandenen Festplatten unangetastet bleiben, bis man dies explizit zuläßt (s.u.). Um Daten trotzdem temporär speichern zu können, wird ein Großteil des Hauptspeichers reserviert, auf den über beschreibbare Verzeichnisse[41] zugegriffen werden kann. Die meisten installierten, also nutzbaren Programme befinden sich auf der CD, welche ebenfalls über entsprechende (nur lesbare) Verzeichnisse in das System eingebunden werden.

Um die Morphix-liFe-CD starten zu können, muß man seinen Rechner[42] so einrichten, daß von CD *gebootet*[43] werden kann, die CD einlegen und den Rechner starten. Als erstes erscheint eine Eingabeaufforderung (*Bootprompt*), an dem Optionen für den Start des Betriebssystems eingegeben werden können. Für die meisten, nicht zu sehr veralteten Rechner, ist nur eine Option interessant, nämlich zur Konfiguration der deutschen Tastaturbelegung („QWERTZ“):

```
morphix lang=de
```
[44]

[41] wie z.B. `/home` oder `/tmp`

[42] unter der Voraussetzung, daß der Prozessor x86 kompatibel ist

[43] „Boot“ bezeichnet den Prozess des Startens eines Betriebssystems.

[44] Bei der standardmässig eingestellten englischen Tastaturbelegung am Bootprompt befindet sich das „=“ zwei Tasten weiter rechts (ohne Umschalttaste) verglichen mit der deutschen Belegung. Desweiteren ist darauf zu achten, daß das Wort „morphix“ miteingegeben werden muß.

Nach Betätigung der Eingabe-Taste beginnt der Boot-Vorgang. Wurde dieser erfolgreich abgeschlossen, befindet man sich in der grafischen Benutzeroberfläche.

Um nun einen liFe.f0rm-Server zu starten, geht man z.B. wie folgt vor:

- Terminal-Fenster öffnen (auf das Monitor-Symbol ganz links auf dem Panel am unteren Bildschirmrand klicken)[45]
- **sh** ▷ `server+dir+port.sh /tmp/life`
- **sh** ▷ `tail -f /tmp/life/lf.log`

Das optionale Kommando „tail“ dient zur fortlaufenden Beobachtung der in der Log-Datei „lf.log“ aufgefangenen Ausgaben des Servers. Wurde der Server erfolgreich gestartet, ist es ab sofort möglich, denselbigen mit dem Client anzusprechen (vgl. 4.3 auf Seite 54):

- ein weiteres Terminal-Fenster öffnen
- **sh** ▷ `client.sh test`
- **sh** ▷ `client.sh /cdrom/LforM/test-sounds/demo08.ogg`
- **sh** ▷ `client.sh /cdrom/LforM/test-sounds/demo07.ogg`
- **sh** ▷ `client.sh /tmp/life/liFe.wav`
- ...

[45] Es ist evtl. nötig ein Symbol mehrfach anzuklicken, bis das Programm tatsächlich gestartet wird. Dies ist auf das CD-Laufwerk zurückzuführen, welches zu viel Zeit benötigt, um das entsprechende Programm zu Verfügung zu stellen und somit einen „Input/Output Error“ verursacht.

Um die so generierte liFe.f0rm anhören zu können, benötigt man ein Programm wie beispielsweise „Xmms“, welches man durch Klicken auf das Lautsprecher-Symbol (das dritte von links auf dem Panel), bzw. durch Eingabe von

sh ▷ `xmms &`

starten kann. Oder man benutzt das Kommandozeilen-Programm „ogg123“, um die zusätzlich erzeugte Ogg-Version der liFe.f0rm abzuspielen:

sh ▷ `ogg123 /tmp/life/liFe.ogg`

Möchte man den Server stoppen, um ihn beispielsweise mit alterierten Argumenten erneut zu starten[46], verwendet man folgendes Kommando:

sh ▷ `client.sh server::DIE`

Zudem besteht natürlich die Möglichkeit weitere Server-Instanzen zu starten, die sowohl einen anderen Port, als auch ein anderes Verzeichnis benutzen:

sh ▷ `server+dir+port.sh /tmp/life2 11111`

Möchte man diesen Server ansprechen, muß man dies dem Client explizit über das Argument `-port` mitteilen (vgl. 4.3 auf Seite 54):

sh ▷ `client.sh -port 11111 dies_ist_nur_ein_test`

[46]Bei Verwendung des Skriptes `server+dir+port.sh` (vgl. A.4 auf Seite 102) ist es mittels eines Editors (wie beispielsweise „nano“) möglich, die in demselbigen gesetzten Argumente, wie `-maxlength` oder `-maxsize`, zu verändern:
sh ▷ `nano /home/morph/LforM/bin.sh/server+dir+port.sh`

Desweiteren ist es möglich, die lokal in dem Rechner befindlichen Festplatten über den Befehl `mount` einzubinden[47] und auf diesen befindliche Klangdateien dem Server zur Verarbeitung zu übergeben, bzw. generierte liFe.f0rm-Dateien in einem Verzeichnis darauf persistent zu speichern[48]:

- **sh** ▷ `mount /mnt/hda1`[49]
- **sh** ▷ `server+dir+port.sh /mnt/hda1/life`
- **sh** ▷ `client.sh /mnt/hda1/klang/datei.mp3`

Analog dazu lassen sich die so eingebundenen Festplatten mit dem Kommando `umount` wieder aus dem System entfernen:

sh ▷ `umount /mnt/hda1`

Ebenfalls läßt sich eine Netzwerkkarte einrichten, um auf andere Rechner zuzugreifen. Um die dafür notwendigen Kommandos ausführen zu können, muß man vorher zum Benutzer *root* werden, der über die nötigen administrativen Rechte hierfür verfügt:

- temporäres *root*-Passwort setzen:

  sh ▷ `sudo passwd`

- zum Benutzer *root* wechseln:

  sh ▷ `su`

[47] Es wird beim Bootvorgang automatisch nach vorhandenen Festplatten-Partitionen gesucht und in `/mnt` für jede gefundene ein entsprechendes Verzeichnis eingerichtet, welches sich über den `mount`-Befehl mit diesem verknüpfen läßt.

[48] Bei Morphix steht den meisten beschreibbaren Verzeichnissen (wie `/tmp`) standardmässig nur der Hauptspeicher zum Speichern von Daten zu Verfügung, der nach dem Ausschalten des Rechners erlischt und somit alle Daten verlorengehen.

[49] die erste Partition der primären Master-Festplatte mounten

Dann erst besteht die Möglichkeit, die Netzwerkkarte entweder mit DHCP (*Dynamic Host Configuration Protocol*) [67] mittles des Aufrufs von

```
sh ▷ dhclient
```

zu konfigurieren, falls dies noch nicht beim Bootvorgang automatisch erfolgt ist, oder man ordnet der Netzwerkkarte (hier „eth0“) eine feste IP-Adresse[50] zu:

```
sh ▷ ifconfig eth0 192.168.0.123 up
```

Bei fest vergebener IP ist es noch nötig, den *Gateway*[51] per

```
sh ▷ route add default gw 192.168.0.1
```

zu setzen sowie einen *Nameserver*[52] der Datei `/etc/resolv.conf` hinzuzufügen:

```
sh ▷ nano /etc/resolv.conf
nameserver 192.168.0.1
```

War die Netzwerk-Anbindung auf die ein oder andere Weise erfolgreich, ist es danach möglich, über das Internet zugängliche Dateien zu verwenden:

```
sh ▷ client.sh http://beispiel.domain/datei.ogg
```

[50]Die IP-Adresse ist abhängig von dem Netzwerk, in dem sich der Rechner befindet. Die angegebenen dienen nur als Beispiel.

[51]Rechner, der die Verbindung in ein anderes Netz (wie dem Internet) zur Verfügung stellt.

[52]Rechner, der Domainnamen in IP-Adressen übersetzt und diese somit erst verfügbar macht.