3 Struktur-Beschreibung

Wegen der Netzwerk- sowie Datenbank-Unterstützung, fiel meine Wahl auf *PHP* [43] als Programmiersprache. Die verwendete PHP-Version „5.0“ enthält Funktionen, mit denen sowohl Server- als auch Client-Anwendungen verwirklicht werden können. Um Skripte, die von der Kommandozeile aus aufgerufen werden können, also nicht die für PHP übliche HTML-Umgebung [54] voraussetzen, kompilierte ich eine auf die erforderlichen Bedürfnisse abgestimmte Version. Ab diesem Zeitpunkt war es mir möglich, PHP-Skripte zu schreiben, die wie ein Shell-Skript, im Fall des Servers auch im *Hintergrund*[3], ausgeführt werden können.

3.1 Client/Server-Architektur

Als ich mir Gedanken zur technischen Planung der liFe.f0rm machte, entschied ich mich für eine Client/Server-Architektur[4] um diese zu realisieren, da unter Umständen intensive, also zeitaufwendige Berechnungen zur Klanggenerierung entstehen können. Um diese zeitunabhängig durchführen lassen zu können, benötigte ich ein Programm, welches ununterbrochen aktiv ist, was eine grundlegende Eigenschaft eines Servers ist[5]. Die eigentliche Klangverarbeitung wird demnach vom Server übernommen, wobei der Client diese bei Bedarf in Gang setzen kann. Der Client dient somit nur als Steuerungseinheit, mit der der Server *befehligt* wird.

Ein weiterer Vorteil dabei ist, daß Server und Client sich nicht auf demselben Rechner befinden müssen, da sie über das Internet-Protokoll *TCP*

[3]Programme, die im Hintergrund ausgeführt werden, sind für den Anwender „unsichtbare“ (da nicht im Vordergrund befindliche) Prozesse, die unabhängig bis eigenständig ihre Arbeit verrichten sowie ihre speziellen Dienstleistungen jederzeit für den Anwender verfügbar halten.

[4]vgl. [58] & [59]

[5]und zudem eine Analogie zu einem lebenden Organismus darstellt

(Transmission Control Protocol) [63] miteinander kommunizieren. Somit ist es beispielsweise möglich, einen Server auf einem wenig belasteten Rechner seine Arbeit verrichten zu lassen, der wiederum von einem Client auf einem stark ausgelasteten Rechner kontrolliert werden kann.

Zudem werden durch solch eine Architektur mehrere Clients möglich, die alle denselben Server steuern können. Dabei ist es nur wichtig, was der Client (über das standardisierte Protokoll TCP) an den Server übermittelt, nicht jedoch in welcher Sprache dieser programmiert wurde. Demnach ist es machbar einen Client mit grafischer Benutzeroberfläche beispielsweise in *C* [5] zu programmieren, der den in PHP programmierten Server kontrollieren kann.

3.2 Interaktion

Wie oben erwähnt, findet die Benutzer-Interaktion mit der liFe.f0rm ausschließlich über einen Client statt. Der bisher verwirklichte (in 4.3 beschriebene) Client, ist rein kommandozeilenbasiert, was der angestrebten möglichst einfachen Benutzung ein wenig widerstrebt, da man Zugang zu einem Rechner haben muß, auf dem dieser Client installiert ist[6]. Aus diesem Grunde sind noch weitere Clients geplant, die von überall zugänglich, also ohne ein entsprechendes Programm auf seinem Rechner installiert zu haben, nutzbar sein sollen. Hierzu gehört ein auf HTML [54] basierendes Webinterface, welches mit jedem beliebigen Browser [55] über eine entsprechende URL[7] [61] aufgerufen werden kann. Dort soll dann die Möglichkeit bestehen, analog zu den Kommandozeilen-Argumenten des jetzigen Clients, über entsprechende Eingabefelder den Server mit Werten zu versorgen. Dazu gehört u.a. das Feld zur Angabe der zu verarbeitenden Klangdatei.

[6] Mit der beiligenden CD ist es jedoch möglich, prinzipiell jeden x86-Rechner dafür zu nutzen (vgl. 4.4 auf Seite 57).

[7] z.B.: *http://beispiel.domain/webclient.php*

Mit einem weiteren Client soll die Interaktion über eMails ermöglicht werden. Hierbei kann der Benutzer Kommandos, in einer speziellen Syntax verpackt, per eMail an eine bestimmte Adresse schicken, diese dann, von einem Client-Skript entgegengenommen und entsprechend aufbereitet, an den Server weitergeleitet werden. Im einfachsten Fall sendet der Benutzer eine URL einer über das Internet zugänglichen Klangdatei, die daraufhin vom Server heruntergeladen und verarbeitet wird. Der Server schickt nach Beendigung der Verarbeitung eine Antwort-eMail an den Benutzer, ob diese erfolgreich war oder nicht[8].

3.3 Datenbank-Anbindung

Alle Informationen über Dateien sowie Benutzer-Interaktionen sollen in einer *MySQL*-Datenbank [47] abgelegt werden. Somit wird es möglich, eine individuelle liFe.f0rm einem Benutzer zuzuordnen, über diese nur er die uneingeschränkte Kontrolle hat. Ein Benutzer muß sich hierfür einen, durch ein Passwort geschützten *Account* (Zugang) einrichten, was wiederum in einem entsprechenden Eintrag in der MySQL-Datenbank resultiert.

Es soll jedoch für jeden, der die liFe.f0rm-Webseite aufruft, ohne Anmeldung möglich sein, auf alle dort existierenden, in ihrem aktuellen Entwicklungsstadium befindlichen, liFe.f0rm-Dateien zugreifen zu können. Dies ermöglicht wiederum die Verbindung, bzw. Kreuzung zweier liFe.f0rm-Instanzen, indem der Benutzer einfach die URL einer anderen liFe.f0rm, mit den oben genannten Clients verwendet. Zudem soll es möglich sein, aus solch einer Verbindung eine neue liFe.f0rm entstehen zu lassen, also als Initial-Version zu verwenden (vgl. 4.1.5 auf Seite 25). Ein Benutzer kann demnach beliebig viele unterschiedliche liFe.f0rm-Instanzen erzeugen, die durch ent-

[8]Das Versenden einer eMail durch den Server ist schon ansatzweise im vorliegenden Prototyp vorbereitet.

sprechende Datenbank-Einträge repräsentiert werden. Ein Überblick in Form einer grafischen Darstellung von liFe.f0rm-Stammbäumen ist aufgrund dieser Daten ebenfalls denkbar (vgl. 3.6 auf Seite 11).

Desweiteren werden, durch die in der Datenbank verfügbaren Daten, eigenständige Aktionen des Servers möglich, wie beispielsweise das regelmäßige Abfragen des Status quo einer liFe.f0rm, zu dem u.a. das Datum der letzten Interaktion des zugehörigen Benutzers gehört. Liegt dies zu lange zurück, kann der Server mit einer entsprechenden eMail an den Benutzer reagieren, um ihn darauf aufmerksam zu machen.

Ebenfalls ist es denkbar, entsprechende Status-Felder einzurichten, die z.B. den Zustand der jeweiligen liFe.f0rm in Bezug auf die *Gesundheit*, bzw. damit einhergehend die *Krankheitsanfälligkeit* widerspiegeln. Ist eine liFe.f0rm erkrankt, kann dies durch Aktionen des Benutzers behoben werden, bzw. zum *Versterben* führen, falls diese Aktionen nicht rechtzeitig erfolgen.

Auf diese Weise lassen sich eine ganze Reihe von datenbankbasierten Parametern (wie Alter, Lebensdauer, Charakter usw.) implementieren, die den Zustand einer liFe.f0rm definieren und somit einer realen Lebensform ein Stück näher bringen.

3.4 Klangverarbeitung als Plugin-Architektur

Die eigentliche Klangverarbeitung habe ich als Plugin-Architektur verwirklicht. Die Plugins werden vom Server, der den *Wirt* für dieselbigen darstellt, bei Bedarf aufgerufen. Welches Plugin genutzt werden soll, wird durch ein entsprechendes Client-Argument definiert (vgl. 4.3 auf Seite 54) oder, falls keines explizit angegeben worden ist, durch einen in der Server-Konfiguration gesetzten Standardwert (vgl. 4.1.1 auf Seite 14).

Solch eine offene Architektur macht es möglich, nachträglich und un-

abhängig von dem Server, weitere Plugins mit wenig Aufwand zu implementieren. Hierbei ist der Plugin-Autor frei in der Wahl der zur Klanggenerierung verwendeten *externen* Programme. Die einzigen Voraussetzungen sind, daß die Programme kommandozeilenbasiert nutzbar sein müssen sowie ein geringes Grundwissen über PHP seitens des Autors vorhanden sein muß. Um den Einstieg zu erleichtern, existiert ein Plugin namens „template“ (vgl. 4.2.3 auf Seite 52), welches als Ausgangspunkt zur Implementation eigener Plugins dienen kann.

Die Schnittstelle zwischen Plugin und Server stellen Variablen dar, über die Informationen & Statistiken der jeweiligen Klangdateien ausgetauscht werden. Der Server übergibt zwei *Arrays*[9], die detailierte Informationen der aktuellen liFe.f0rm sowie der heruntergeladenen Datei enthalten, an das Plugin und erwartet einen Pfad auf die vom Plugin neu erzeugte Datei zurück, um mit dieser den Verarbeitungsprozess abzuschließen.

3.5 Reguläre Ausdrücke

Im Rahmen dieser Diplomarbeit habe ich mich das erste Mal mit „Regulären Ausdrücken“ mit Hilfe eines Standardwerkes [4] auseinandergesetzt. Nun frage ich mich, wie ich vorher ohne diese zurecht kam. Denn dieses mächtige Werkzeug ist unabdingbar, wenn es um die Arbeit mit Text, bzw. Zeichenketten (*Strings*) im weitesten Sinne geht. Wie beispielsweise in 4.1.6 beschrieben, habe ich Reguläre Ausdrücke dazu benutzt, um Statistiken der Klangdateien zu erstellen, die für die weitere Verarbeitung derer genutzt werden.

Desweiteren ist vorgesehen, Reguläre Ausdrücke zur Implementation von

[9] Die wichtigste Datenstruktur von PHP sind sogenannte „Arrays“. Diese stellen ein- bis mehrdimensionale Anordnungen, bzw. Felder dar, die eine den Dimensionen entsprechende Sammlung von Variablen unterschiedlichen Typs, mit zugehörigen Werten enthalten. (vgl. [1], [2], [3], [43], [50])

„Sinnesorganen“ zu nutzen, da das Internet, welches den *Lebensraum* für die liFe.f0rm darstellt, weitestgehend auf Text basiert. Somit kann diese textbasierte *Umwelt*, durch entsprechende Interpretation mit Hilfe von Regulären Ausdrücken, in Impulse umgesetzt werden, die die jeweilige liFe.f0rm in einer bestimmten Art und Weise beeinflussen (vgl. 6.2 auf Seite 66). Dies läßt sich auch auf den Rechner, auf dem eine liFe.f0rm beheimatet ist, ausdehnen, der ebenfalls Werte im Klartext liefern kann, die als Einflüsse weiterverwertet werden können. Dazu gehören z.B. die Auslastung der Netzwerkverbindung, der Festplatte, des Hauptspeichers oder des Prozessors, welche sich aus den im Verzeichnis `/proc/` befindlichen *Pseudodateien*[10] eines GNU/Linux-Systems auslesen lassen.

Somit stehen eine ganze Reihe von potentiellen Parametern zur Verfügung, die mit Regulären Ausdrücken in ein verwertbares Format umgewandelt und individuell auf eine liFe.f0rm einwirken können.

3.6 Stammbaum-Rekonstruktion

Jeder Schritt kann anhand der gesetzten *Timestamps*[11], der Bestandteil der Dateinamen für alte und heruntergeladene Dateien ist, zurückverfolgt werden. Somit lassen sich immer beide *Elternteile* eines bestimmten Stadiums der liFe.f0rm rekonstruieren. Allerdings nur theoretisch und nicht praktisch, da der Zufall eine wesentliche Rolle bei dem Vererbungsprozess spielt (vgl. 2.2 auf Seite 4).

Die Abbildung 1 enthält eine schematische Darstellung solcher Stammbäume. Hierbei habe ich den jeweiligen Timestamp durch ein „$“ gekennzeichnet sowie diesen auf eine einstellige, repräsentative Zahl reduziert,

[10] Keine tatsächlich physikalisch vorhandenen Dateien, die sich jedoch als solche behandeln lassen, also lesend und teilweise schreibend darauf zugegriffen werden kann.

[11] Zeit in Sekunden, vom 01.01.1970 00:00:00 UTC an gerechnet

gepaart mit einem Buchstaben, der die Zugehörigkeit zu einer bestimmten *Familie* darstellt[12].

Die umschließenden Kästen entsprechen, neben der Familienzugehörigkeit, dem Verzeichnis, in dem die Klangdateien gespeichert sind, wodurch die eigentliche liFe.f0rm, mittels des Server-Arguments `-liFedir`, definiert wird (vgl. 4.1.2 auf Seite 19).

Die neu zu verarbeitenden Dateien, die vom Client an den Server übergeben werden, habe ich mit „-down“ gekennzeichnet. Die Klangdateien, die als Initial-Version (vgl. 4.1.5 auf Seite 25), also erste liFe.f0rm eines Stammbaumes dienen, sind als „Init“ gekennzeichnet.

Die eingezeichneten Pfeile zeigen mögliche Verbindungen zwischen den unterschiedlichen liFe.f0rm-Familien. Dabei bedeutet solch ein Pfeil, daß eine bestimmte Klangdatei ihren Weg (unverändert) in eine andere Familie gefunden hat (`$a2.wav` ⟶ `$b1-down.wav`), also sich mit der vorhandenen liFe.f0rm der anderen Familie verbunden hat, oder gar der Ausgangspunkt, bzw. die Initial-Version für einen neuen liFe.f0rm-Stammbaum darstellt (`liFe-b.wav` ⟶ `$c1.wav`).

Mit der geplanten Verlagerung der Daten in eine MySQL-Datenbank wird es möglich, eine automatische, grafische Darstellung auf der Basis von Timestamps zu implementieren. Ein Feld, eines Datenbank-Eintrages zu einer Datei, enthält dann den Timestamp, sodaß problemlos und schnell die Verwandtschafts-Verhältnisse, durch eine Such-Abfrage an die Datenbank, ausgelesen und dargestellt werden können.

[12] Ein Beispiel dazu:
`/dir-a/1097721187.wav` wird reduziert auf `$a1.wav`
`/dir-a/download/1097721187.ogg` wird reduziert auf `$a1-down.ogg`

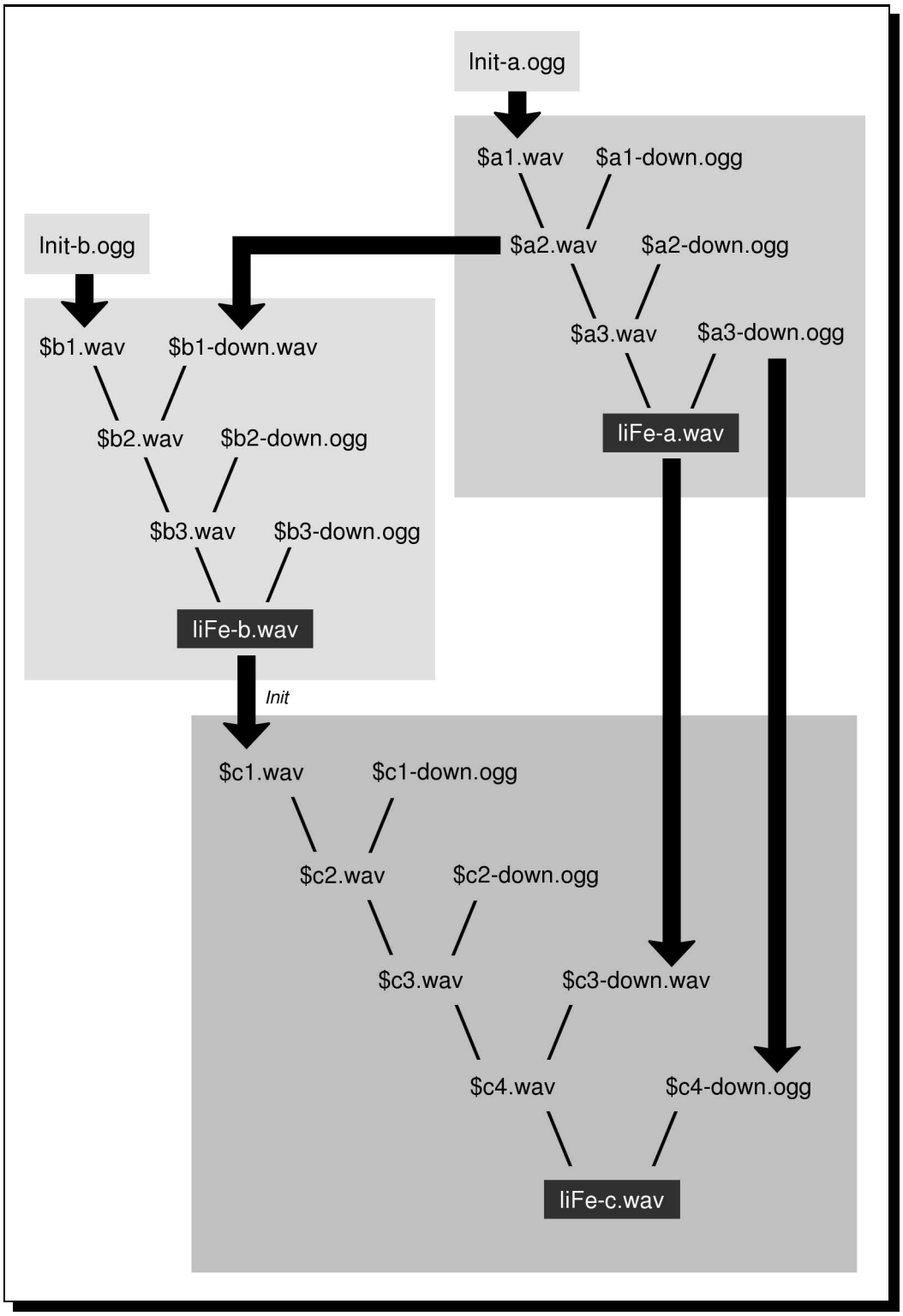

Abbildung 1: *Rekonstruktion des Stammbaumes anhand von „Timestamps“ in Dateinamen*