Entwicklung eines PHP-Web-Frontends zur Neurosimulation auf einem Compute Server

Diplomarbeit

zur Erlangung des Grades eines Diplom-Ökonomen der
Wirtschaftswissenschaftlichen Fakultät der Universität Hannover

vorgelegt von

Name: König Vorname: Simon
[redacted]

Erstprüfer: Prof. Dr. Michael H. Breitner

Hannover, den 08.11.2004

Inhaltsverzeichnis

Abbildungsverzeichnis **iii**

Abkürzungsverzeichnis **iv**

1 Einleitung **5**

2 Analyse **6**

2.1 Faun 6

2.2 Grafische Benutzerschnittstelle 8

3 Sollkonzept **14**

3.1 Datenorientierte Sicht 14

3.2 Funktionsorientierte Sicht 14

4 Entwurf **17**

4.1 Programmiersprachen 17

4.1.1 Delphi/Kylix 17

4.1.2 Java Servlets und Java Server Pages 18

4.1.3 PHP: Hypertext Preprocessor 19

4.1.4 Fazit 19

4.2 Bildformate 20

4.3 Architektur 21

4.3.1 Trennung von Programmlogik und Darstellung 21

4.3.2 Trennung von Inhalt und Layout 22

4.3.3 Datenmodell 23

4.3.3.1 `Parameter` 24

4.3.3.2 `Inifile` 25

4.3.3.3 `Controlfile` 26

4.3.3.4 `Outputfile2` 28

4.3.3.5 `Pattern` 30

4.3.3.6 `Faun` 30

4.3.3.7 `Project` 33

4.3.3.8 `User` 34

4.3.3.9 `Error` 37

4.3.4 Erzeugung der HTML-Seiten . . . 38
4.3.4.1 `TemplateController` . . . 39
4.3.4.2 `MasterPage` . . . 39
4.3.4.3 `Factory` . . . 42
4.3.4.4 `Button` . . . 42
4.3.4.5 `Page` . . . 43
4.3.4.6 `formate.css` . . . 44
4.3.4.7 Hinzufügen neuer Seiten . . . 46
4.3.5 Erzeugung der Grafiken . . . 47
4.4 Benutzung des Web-Frontends . . . 49
4.4.1 Anmeldung . . . 49
4.4.2 Administratorbereich . . . 49
4.4.3 FAUN-Bedienoberfläche . . . 51
4.4.3.1 Faun . . . 51
4.4.3.2 Dateien . . . 53
4.4.3.3 Muster bearbeiten . . . 53
4.4.3.4 Beispiele laden . . . 55
4.4.3.5 Parameter . . . 56
4.4.3.6 Auswertung . . . 57

5 Realisierung **59**
5.1 Test . . . 59
5.2 Einrichtung der Umgebung . . . 60
5.3 Softwarequalität . . . 61

6 Ausblick **63**

Literatur **65**

Anhang **67**

1 Einleitung

Die Idee der Neurosimulation orientiert sich am Vorbild der Natur: Das Nervensystem des Menschen besteht aus Milliarden von Nervenzellen (Neuronen), die jeweils mit tausenden anderer Nervenzellen verbunden sind. Die Ausprägung einer dieser Verbindungen und damit der Einfluss auf andere Neuronen spielt beispielsweise bei Lernprozessen eine wesentliche Rolle. Das Gehirn erhält durch das gleichzeitige Zusammenwirken vieler vergleichsweise einfacher Einheiten seine Leistungsfähigkeit.[1]

Dieses Prinzip wird als Rechenmodell verwendet, um künstliche neuronale Netze zu trainieren. Dabei spricht man ebenfalls von Neuronen, die über Gewichte miteinander verbunden sind. Wie beim natürlichen Vorbild können diese Gewichte das Signal des Neurons verstärken oder abschwächen. Auch hier wird das Ergebnis durch die Anzahl und Stärke der Verbindungen zwischen den Neuronen bestimmt.[2]

Neuronale Netze werden häufig für Probleme eingesetzt, bei denen exakte Lösungen nicht möglich oder zu aufwändig sind. So werden neurale Netze beispielsweise zur Schrifterkennung eingesetzt. Durch geeignetes Training können mit neuronalen Netzen sogar bei der Verarbeitung „verrauschter“ Daten, z. B. durch Flecken auf einem eingescannten Blatt, hohe Trefferquoten erreicht werden.

Im Rahmen dieser Diplomarbeit wird ein Web-Frontend für den Neurosimulator FAUN entwickelt. Die bereits vorhandene Oberfläche (siehe Abschnitt 2.2 auf Seite 8) bietet bereits hohen Komfort zur Bedienung, so dass sich die Frage stellt, warum eine weitere Benutzerschnittstelle benötigt wird. Der Rechenaufwand zum Training neuronaler Netze ist hoch. Abhängig von der Topologie und der Menge an Eingabedaten kann das Training Minuten, Stunden oder sogar Tage dauern. Für diejenigen Benutzer, die nicht über Hochleistungsrechner mit grafischer Bedienoberfläche verfügen können, bedeutet dies eine echte Einschränkung. Falls sie Zugriff auf Hochleistungsrechner haben, erfolgt die Steuerung möglicherweise textbasiert, so dass die bisherige Oberfläche nicht eingesetzt werden kann. Bleibt also noch der Einsatz auf Desktop-PCs oder mobilen Rechnern, die aber in der Regel nicht zum Dauereinsatz konzipiert sind. Darüber hinaus wäre der Rechner dann für andere prozessorintensive Vorgänge regelrecht blockiert.

Diesem Problem will das Web-Frontend begegnen. Die Installation und der Betrieb erfolgt vorzugsweise auf einem leistungsstarken Rechner, der seine Prozessorleistung zur

[1] vgl. [Pet04], Absatz 29
[2] vgl. [Pet04], Absätze 28, 30-34

Verfügung stellt, einem sogenannten Compute Server. Der Benutzer kann dann sowohl durch den Einsatz von Hochleistungsrechnern als auch von einer ansprechenden und funktionalen Bedienoberfläche profitieren.

Das hier verwendete Vorgehensmodell lehnt sich stark an das von Peter Stahlknecht und Ulrich Hasenkamp vorgeschlagene, allgemeine Vorgehensmodell [3] an. In der Analysephase wird zunächst die derzeitige Situation analysiert (Abschnitt 2), bevor mit der Feststellung der Anforderungen im Sollkonzept (Abschnitt 3 auf Seite 14) diese Phase abgeschlossen wird. In der Entwurfsphase (Abschnitt 4 auf Seite 17) werden schließlich neben der Architektur des Systems und seinen Schnittstellen die zu verwendenden Mittel diskutiert und beschrieben. In der Realisierungsphase (Abschnitt 5 auf Seite 59) werden die durchgeführten Tests sowie die Installation des Web-Frontends dokumentiert und Aspekte der Softwarequalität besprochen.

2 Analyse

2.1 Faun

FAUN wird durch eine Reihe von Parametern gesteuert, die in zwei Textdateien mit den Namen `control_1_1_0` und `control_2_1_0` aufgeteilt sind. Letztere beeinflusst das Optimierungsverfahren und die Menge an Informationen, die das Optimierungsverfahren ausgibt. In den meisten Fällen wird die Datei folgenden Inhalt besitzen:[4]

```
Begin
  Print file           46
  Verify level         -1
  Derivative level      1
  Major print level     0
End
```

Die Datei `control_1_1_0` dient der Steuerung von FAUN und enthält insgesamt 13 Parameter. Jeder Parameter beginnt auf einer neuen Zeile, jedoch sind mehrere Parameter in Teilparameter unterteilt, so dass eine Zeile auch mehrere Werte enthalten kann. Kommentarzeilen werden durch „#“ eingeleitet.[5]

[3] vgl. [SH02], S. 221
[4] vgl. [Bre03], S. 187 f.
[5] vgl. [Bre03], S. 172-187

Darüber hinaus benötigt FAUN Trainings- und Validierungsmuster. Die Trainingsmuster liegen in Textdateien mit Namen `training_unscaled` bzw. `training_scaled` vor, abhängig davon, ob die Muster bereits skaliert („scaled“) sind oder durch FAUN skaliert werden („unscaled“). Für die Validierungsmusterdatei gilt die Benennung analog. Die Anzahl der Muster ist jeweils beliebig. Jedes Muster besteht aus einem Wert für jedes Eingabeneuron gefolgt von einem Wert für jedes Outputneuron. Die Werte können auf mehrere Zeilen verteilt werden, jedoch dürfen Werte für Input- und Outputneuronen nicht in einer Zeile stehen. In diesen Dateien werden Kommentarzeilen ebenfalls durch „#“ eingeleitet.[6]

Sind alle notwendigen Dateien vorhanden, generiert FAUN nach Programmstart, neben den Ausgaben des Programms selbst bezüglich der Steuerungsparameter und der Anzahl der eingelesenen Muster, Informationen in bis zu fünf verschiedenen Dateien[7]:

`output_1` In diese Datei schreibt FAUN die Verläufe von Trainings- und Validierungsfehlern sämtlicher Netze, d. h. auch die von nicht erfolgreich trainierten Netzen.

`output_2` Die Gewichte aller erfolgreich trainierten Netze sowie deren Trainings- und Validierungsfehler werden in diese Datei geschrieben.

`output_3` Ausgaben des Optimierungsverfahrens, abhängig von Einstellungen in der Datei `control_2_1_0`.

`output_4` Diese Datei wird nur verwendet, wenn für Parameter 9 der Wert 1 gewählt wird und enthält den Eingabewert sowie die Ist- und Sollausgabe des Netzes.

`output_5` Wird der Wert für Parameter 3 größer als 1 gewählt (Training mehrerer Netze gleichzeitig), werden die Fehlerverläufe der Einzelnetze in diese Datei geschrieben.

FAUN erwartet sämtliche Eingabedateien in festgelegten Verzeichnissen, relativ zum Programmverzeichnis. Die Pfade zu den Ausgabedateien sind ebenfalls festgelegt: Das Verzeichnis `data/control_and_output_files` enthält die Steuerungs- und Ausgabedateien, das Verzeichnis `data/data_files` die Trainings- und Validierungsdaten. Den Bezug zum Programmverzeichnis veranschaulicht Abbildung 1 auf der nächsten Seite. Der Name des Programmverzeichnisses ist beliebig.

[6] vgl. [Bre03], S. 172 f.
[7] vgl. [Bre03], S. 196-201

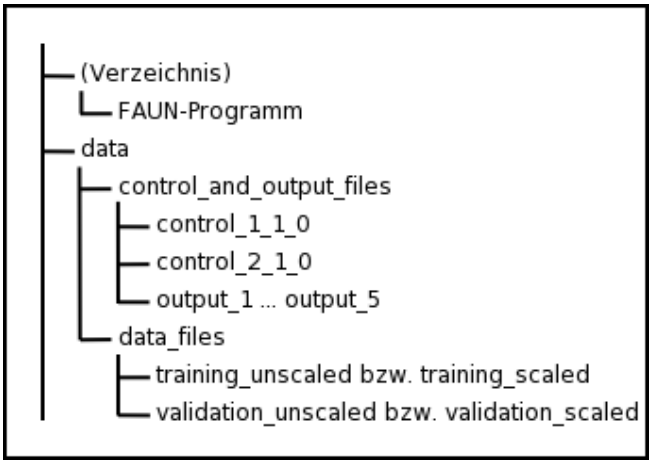

Abbildung 1: Erforderliche Verzeichnisstruktur für FAUN

2.2 Grafische Benutzerschnittstelle

Mittels Delphi[8] wurde bereits eine Oberfläche geschaffen, die Benutzern die Bedienung von FAUN erleichtern soll (siehe Abbildung 2 auf der nächsten Seite). Die darin implementierte Funktionalität ist in Abbildung 3 auf der nächsten Seite dargestellt.

Die vorbereitenden Tätigkeiten umfassen die Einstellung der Parameter durch Laden vorhandener Steuerungsdateien bzw. durch Anpassung in der Oberfläche, sowie das Bereitstellen von Trainings- und Validierungsmustern. Letztere können über die Oberfläche nicht verändert werden. Dies muss weiterhin durch einen externen Editor geschehen. Alternativ kann eine Beispielkonfiguration, bestehend aus Steuerungsdateien und Trainings- und Validierungsmustern, geladen und deren Parameter gegebenenfalls variiert werden. Die Dateien `DefControl_1_1_0.ini` sowie `DefControl_2_1_0.ini` enthalten die Parameter für die jeweilige Steuerungsdatei. Derzeit existieren zu den Beispielen „Space shuttle guidance“ und „4-bit analog/digital converter“ verschiedene Unterbeispiele mit unterschiedlichen Parameterwerten. Diese Werte sind in Dateien mit dem Präfix „Shuttle“ bzw. „Converter“, d.h. `ShuttleDefControl_1_1_0.ini` usw., enthalten.

Das Format dieser Dateien entspricht den unter dem Betriebssystem Windows von Microsoft verbreiteten Inidateien. Diese Dateien enthalten benannte Abschnitte der Art „[Name]“. Jeder Abschnitt enthält Zeilen der Art „Schlüssel=Wert“, wobei der zugeordnete Wert über den Schlüssel eindeutig identifiziert werden kann:

[8] `http://www.borland.de/delphi/index.html`

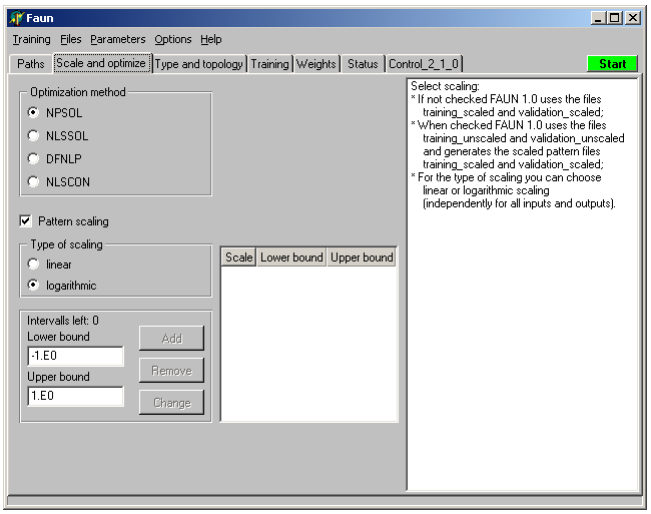

Abbildung 2: FAUN-GUI unter Windows

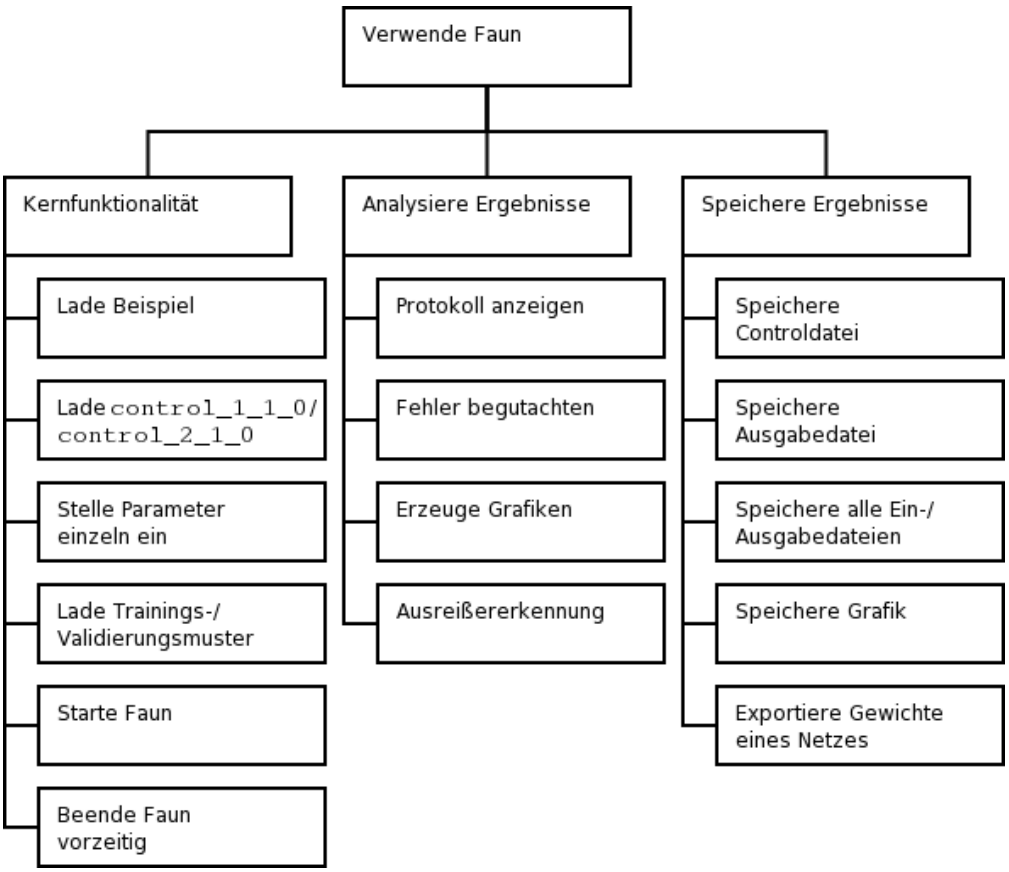

Abbildung 3: Funktionsbaum für FAUN-GUI

```
[Abschnitt1]
Schlüssel1=Wert1
Schlüssel2=Wert2
⋮
[Abschnitt2]
Schlüssel1=Wert3
⋮
```

Hier entspricht der Abschnittsname dem Beispielnamen; der Schlüssel entspricht für `control_1_1_0` der Nummerierung der Parameter[9], für `control_2_1_0` jeweils dem ersten Wort des vollständigen Parameternamens.

Die Dateinamen der Trainings- und Validierungsdaten zu den Beispielen enden auf „.txt“ und beginnen mit einem Präfix, das aus der Inidatei ermittelt werden kann, indem der Wert zum Schlüssel „prefix“ des betreffenden Abschnitts ausgelesen wird. Eine Ausnahme bilden die Beispiele zum „4-bit analog/digital converter“, denn hier dient der Abschnittsname gleichzeitig als Dateinamenspräfix. Die Dateien können unverändert in das Verzeichnis `data_files` kopiert werden, allerdings müssen Präfix und Suffix aus dem Dateinamen entfernt werden.

Zur Nachbereitung eines FAUN-Laufs gehört die Analyse. Die Ausgabe von FAUN wird in der Oberfläche dargestellt und erleichtert so die Erkennung z. B. fehlerhafter Parameter. Aus den Ausgabedateien können mittels Gnuplot[10] Grafiken im PNG-Format erzeugt und angezeigt werden. Einzige Ausnahme bildet die Ausgabedatei `output_3`. Sie enthält Informationen über das verwendete Optimierungsverfahren[11] und kann über die Oberfläche nicht ausgewertet werden. Zusätzlich werden die Netze mit den geringsten Trainingsfehlern ausgegeben und können zur Erkennung von Ausreißern ausgewählt werden. Der Zusammenhang zwischen den einzelnen Funktionen sowie der typische Arbeitsablauf mit der Oberfläche wird als Aktivitätsdiagramm in Abbildung 4 auf der nächsten Seite dargestellt.

Die Steuerung von Gnuplot erfolgt mit Hilfe von Textdateien, die Gnuplot-Befehle enthalten und abgearbeitet werden. Zu jeder erzeugbaren Grafik existiert eine Vorlage, die durch die Oberfläche ausgefüllt, gespeichert und anschließend mit Gnuplot verarbeitet wird. Der Aufbau dieser Vorlagen ist jeweils identisch und wird am Beispiel der

[9] vgl. die ursprüngliche Datei in [Bre03], S. 172-178
[10] `http://www.gnuplot.info/`
[11] vgl. [Bre03], S. 199

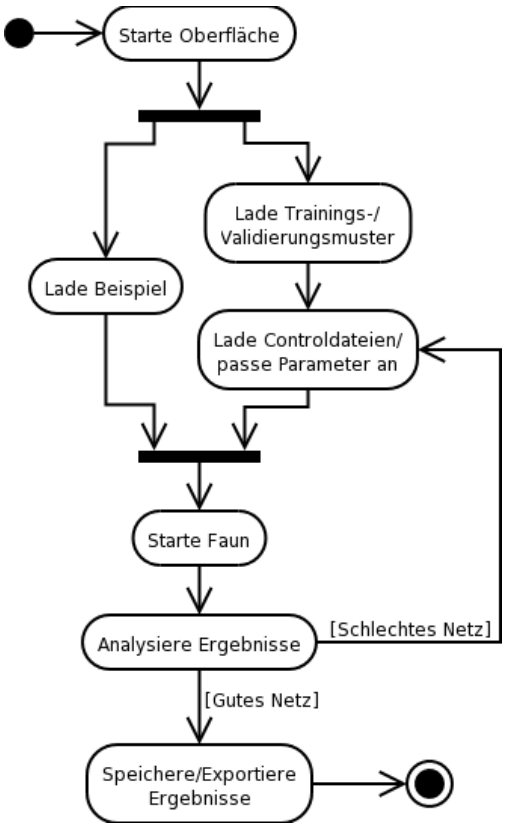

Abbildung 4: Typischer Arbeitsablauf mit FAUN-GUI

Befehlsdatei zur Darstellung der Fehler trainierter Netze[12] beschrieben:

```
set title "Faun 1.0: Error of successfully trained networks" 0,0
set xlabel 'Rank of the successfully trained network'
set ylabel 'Training and validation error'
set terminal png small color
set output 'errormod_1.png'
set size <#xscale>,<#yscale>
plot [<#xmin>:<#xmax>] [<#ymin>:<#ymax>] '<#FaunOutputFile>' \
    u :1 title 'training error' w l lt <#lt1> lw <#lw1>, \
  ⋮
```

Die ersten drei Zeilen legen Titel und Achsenbeschriftung fest. Mit `terminal` wird die Art der Ausgabe festgelegt, `output` hingegen bestimmt den Dateinamen. Ohne Angabe

[12] In der GUI ist diese Grafik als Offline-Grafik „Errors (sorted)“ abrufbar.

eines Pfades wird die Grafik im aktuellen Verzeichnis erzeugt, so dass der Aufrufer in das Zielverzeichnis wechseln muss, bevor er Gnuplot startet. Die letzte Einstellung betrifft die Größe der Grafik. Sie wird gesetzt, um die Grafik auch bei Änderungen der Fenstergröße angemessen anzeigen zu können. Da das Bildformat PNG kein Vektorformat[13] ist, kann sie nicht beliebig skaliert werden. Bei Änderungen der Fenstergröße wird daher die Grafik mit der neuen Größe erneut erzeugt, indem die Platzhalter `<#xscale>` und `<#yscale>` durch die gewünschte Breite und Höhe ersetzt werden.

Das Zeichnen der Grafik erfolgt durch den `plot`-Befehl. In eckigen Klammern kann zunächst ein Intervall spezifiziert werden, um Ausschnitte zu betrachten. Bleiben die Klammern leer, wird eine Grafik aus allen Eingabedaten erzeugt. Anschließend werden in diesem Beispiel mehrere durch Kommata getrennte Linien definiert, deren Daten Gnuplot aus `<#FaunOutputFile>` extrahiert. Da der Befehl eigentlich in einer einzigen Zeile stehen muss, kann Gnuplot durch „\“ mitgeteilt werden, dass der Befehl in der nächsten Zeile fortgesetzt wird. Die Ersetzung aller Platzhalter kann beispielsweise zu folgendem `plot`-Befehl führen:

```
plot [:] [:] '../data/control_and_output_files/output_2_mod' \
    u :1 title 'training error' w l lt 1 lw 3, \
  ⋮
```

Die Datei `output_2_mod` wird nicht direkt von FAUN erzeugt. Sie wird aus der Datei `output_2` durch die Oberfläche erzeugt und enthält die Fehler der trainierten Netze in aufsteigender Reihenfolge.

Aus der Oberfläche heraus können sämtliche Ein- und Ausgabedateien von FAUN sowie die erzeugten Grafiken gespeichert werden. Ein einzelnes erfolgreich trainiertes Netz kann in einem Textformat exportiert werden, das zur Weiterverarbeitung mit Maple[14] geeignet ist[15].

Unterstützend werden zu jedem Parameter sowie einigen anderen Komponenten Hilfetexte zur Erläuterung der Auswirkungen sowie möglichen Einstellmöglichkeiten geboten.

Die Datenflüsse zeigt Abbildung 5 auf der nächsten Seite.

[13] `http://de.wikipedia.org/wiki/Vektorgrafik`
[14] `http://www.maplesoft.com/products/maple/`
[15] vgl. zur Weiterverarbeitung [Bre03], S. 228-240

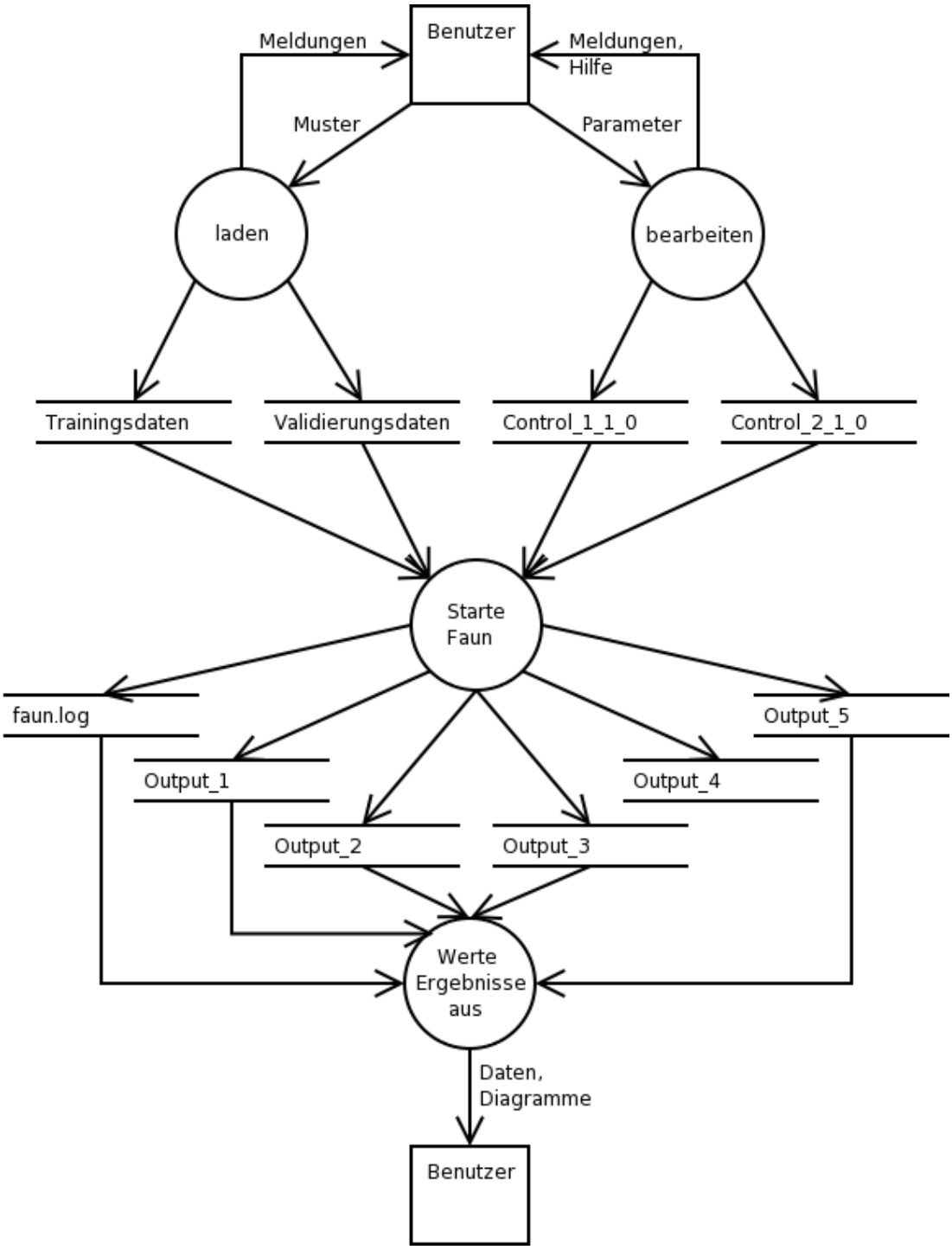

Abbildung 5: Datenflussdiagramm für FAUN-GUI

Betrieb ist kein weiterer Dienst erforderlich. Erst ein Seitenaufruf führt dazu, dass der Webserver die entsprechenden PHP-Dateien an den Interpreter schickt, um die Seite zu erzeugen. Die Aufteilung in Klassen vermindert den zu verarbeitenden Code. Darüber hinaus sind die einzelnen HTML-Seiten relativ klein, so dass nur wenig Daten übertragen werden müssen. Die erzeugten Grafiken sind ebenfalls nur wenige Kilobyte groß, so dass nicht die Seitenerzeugung, sondern die Ausführungsgeschwindigkeit von FAUN bzw. Gnuplot einen Engpaß darstellt.

Änderbarkeit Dieses Merkmal ist für die Weiterentwicklung des Web-Frontends von großer Bedeutung. Wie in Abschnitt 4.3.4.7 auf Seite 46 erläutert wurde, kann das Web-Frontend mit geringem Aufwand um weitere Seiten oder weitere Grafiken ergänzt werden, ohne die Funktion anderer Seiten zu beeinflussen oder überhaupt zu kennen. Anpassungen bestehender Seiten sind ohne Seiteneffekte möglich und deren Darstellung ist durch Umgestaltung des HTML-Codes sogar überwiegend ohne PHP-Kenntnisse möglich. Änderungen an Klassen des Datenmodells können leicht überprüft werden, indem die Test-Skripte (siehe Abschnitt 5.1 auf Seite 59) im Browser aufgerufen werden. Auf diese Weise kann die Wahrscheinlichkeit unerwarteter Auswirkungen der vorgenommenen Änderungen minimiert werden.

Übertragbarkeit Obwohl die Unabhängigkeit von der zugrundeliegenden Software keine Anforderung war, konnte sie mit geringen Mitteln erreicht werden. Das Web-Frontend funktioniert grundsätzlich mit jedem Webserver, der Unterstützung von PHP 5 bietet, unabhängig vom Betriebssystem, mit einer Ausnahme: Der vorzeitige Abbruch von FAUN war mit PHP nicht zu lösen, so dass hier betriebssystemspezifische Aufrufe benötigt wurden. Bei einem Wechsel des Betriebssystems muss daher ein Ersatz gefunden werden.

Die Installation selbst ist trivial, da lediglich das Verzeichnis in einen dem Webserver zugänglichen Ordner kopiert und der Pfad dahin in die Datei `site-values.php` eingetragen werden muss.

6 Ausblick

Das in dieser Diplomarbeit entwickelte PHP-Web-Frontend ermöglicht wie die in Delphi entwickelte Oberfläche eine übersichtliche und komfortable Bedienung von FAUN. Das Web-Frontend befreit den Benutzer allerdings noch weitgehender von der Notwendigkeit,

die einzelnen Ein- und Ausgabedateien von FAUN kennen zu müssen. Möglichkeiten der Bearbeitung sowohl der Steuerungsparameter als auch der Trainings- und Validierungsmuster sind nun unter einer einheitlichen Oberfläche vereint. Trotzdem besteht jederzeit die Möglichkeit, vorhandene Dateien einzubinden.

Aus Entwicklersicht bietet das Web-Frontend ebenfalls Vorteile. Borland scheint Kylix zu vernachlässigen und stattdessen auf .NET zu setzen (vgl. Abschnitt 4.1.1 auf Seite 17), so dass hier möglicherweise Anpassungen im Quellcode notwendig werden, um die Oberfläche für zukünftige Windows-Plattformen bereitstellen zu können. Die Entwicklung mit Kylix scheint wenig erfolgversprechend.

Mit PHP hingegen steht eine Programmiersprache zur Verfügung, die frei verfügbar ist und aktiv weiterentwickelt wird. Darüber hinaus ist die zur Verfügung stehende Funktionalität mittels PEAR stark erweiterbar, so dass zukünftigen Anforderungen leichter begegnet werden kann.

Die weitgehende Unabhängigkeit der einzelnen Klassen des Web-Frontends ermöglicht eine relativ schnelle und unkomplizierte Erweiterung des Systems. Dies ist besonders vor dem Hintergrund wichtig, dass eine FAUN-Version für Parallel- und Vektorrechner entwickelt wird. Die zusätzlichen Möglichkeiten können bequem auf einer zusätzlichen Seite untergebracht werden, ohne das vorhandene System zu beeinflussen.

Im Rahmen der Weiterentwicklung des Web-Frontends sollte außerdem darüber nachgedacht werden, die bisherige vollständig serverseitige Steuerung teilweise auf den Client zu verlagern oder clientseitig zu ergänzen. Beispielsweise können Parameter bereits bei der Eingabe auf offensichtliche Fehler überprüft werden, ohne die Daten der vollständigen Seite dem Webserver zur Validierung schicken zu müssen. Das vermeidet unnötige Datenübertragung und ermöglicht eine schnellere Rückmeldung dem Benutzer gegenüber.

Zusammenfassend kann festgestellt werden, dass das hier entwickelte Web-Frontend, aufgrund voraussichtlich längerfristig verfügbarer Technologien und Komponenten sowie guter Wartbarkeit, ein großes Potential birgt, dass es sowohl aus Benutzer- wie auch aus Entwicklersicht zu nutzen gilt.