Konzeption und Realisierung eines Online-Experiments mit Zeiterfassung

Phillip Merensky <phillip.sky@gmx.de>
26. Oktober 2005

Bericht über ein Teilprojekt

Kognitive Systeme
SS 2005
Otto-Friedrich-Universität Bamberg

Zusammenfassung

Der vorliegende Bericht umfasst Konzeption und Realisierung eines Online-Experiments mit Zeiterfassung. Besonderes Augenmerk liegt hierbei auf den speziellen Herausforderungen und am aktuellen Beispiel nachvollziehbaren Lösungen, die bei einer derartigen Implementierung zu berücksichtigen sind. In diesem Bericht soll aufgezeigt werden, wie es mit vertretbarem Aufwand möglich ist ein den Anforderungen entsprechend sicheres Experiment mit klarem Software-Design und konsistentem Datenbestand zu entwickeln. Die Datenmodellierung bei Experimenten dieses Umfangs ist leicht zu bewältigen und genau wie die Erstellung der HTML-Seiten deshalb bewusst nicht Bestandteil dieser Ausführungen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ausgangspunkt des Projekts war der inkrementelle Entscheidungsbaum-Algorithmus CAL2 (Unger & Wysotzki 1981), der den induktiven Erwerb von Konzepten modelliert. Die Reihenfolge der Trainingsbeispiele hat dabei erheblichen Einfluss auf die Komplexität der gelernten Regel. CAL2 findet für die Trainingsmenge zwar immer eine Regel, die es ermöglicht die Trainingsbeispiele richtig zu klassifizieren; dass es die einfachste und kürzeste Regel ist, wird dabei aber nicht sichergestellt. Unter der Annahme, dass auch der Mensch durch die Reihenfolge der Trainingsbeispiele in der Wahl seiner Regel beeinflusst werden kann, sollte mit Hilfe des Experiments nachgewiesen werden, dass die Probanden, die für ihre Entscheidung eine komplexere Regel verwenden, länger für eine Klassifkation von ungesehenen Beispielen brauchen, als diejenigen mit einer Regel aus weniger Konjunktionen.

Der Aufbau sollte als Online-Experiment realisiert werden. Unserem kleinen Projektteam war es so möglich zahlreiche Probanden zu verwalten ohne auf zusätzliches Personal oder zusätzliche Laboratorien zurückgreifen zu müssen. Das WWW ermöglicht außerdem ein sehr heterogenes Probandenfeld, da das Medium von Repräsentanten nahezu jeder Bevölkerungsschicht genutzt wird. Vorteilhaft war auch die einfache Möglichkeit neue Probanden zu gewinnen. Der Verweis auf das Experiment kann für einen größeren Teilnehmerkreis einfach auf einschlägigen Seiten publiziert werden (Birnbaum 2000).

Kapitel 2

Problemstellung

Im folgenden soll eine implementierungsbezogene Sicht der Anforderungen erstellt werden. Diese Anforderungen bilden das Gerüst und dienen als Anhaltspunkt für die Implementierung, sollen sie aber noch nicht detailliert festlegen.

2.1 Anforderungen

Das Experiment ließ sich schon im Vorfeld in vier gut separierbare Phasen einteilen, die auch aus Sicht der Implementierung sinnvoll erschienen. Um typische Fehler beim Planen und bei der späteren Realisierung auszuschließen, wurde die Vorlage von Reips (2002) zur Erstellung von Online-Experimenten als Anhaltspunkt verwendet.

2.1.1 Einführung in das Experiment

Die Startseite der Homepage des Experiments soll Informationen über die Motivation und einen kurzen Überblick über die Thematik des Experiments bieten. Im Vorfeld sollte außerdem sichergestellt werden, dass die nötigen softwaretechnischen Voraussetzungen (Cookies, JavaScript) für eine fehlerfreie Durchführung des Experiments erfüllt sind. Der Proband kann von dort mit Hilfe eines Links zu den Ablaufinformationen voranschreiten. Ziel hierbei ist es ihm kurz und prägnant näherzubringen, wie er das Experiment zu bewältigen hat. Nachdem alle nötigen Hinweise aufgenommen wurden, soll der Benutzer dann mit Hilfe eines weiteren Links in die Lernphase gelangen.

2.1.2 Lernphase

Der Erwerb von Konzepten gliedert sich in eine Lern- und eine Anwendungsphase. Die Lernphase sollte aus acht Beispielen bestehen, die durch Buchstabenkombinationen jeweils gleicher Buchstaben dargestellt werden. Die unterschiedlichen Merkmale anhand derer sich die Beispiele klassifizieren lassen sollen, sind Buchstabe, Anzahl, Größe und Farbe. Jedes dieser Merkmale soll in drei Ausprägungen vorliegen. Ein Durchgang in einer Lernphase soll immer aus drei Schritten bestehen:

1. Anzeige des Beispiels und der Frage, ob es zum Konzept gehört. Die Antwortmöglichkeiten sollen sich auf *Ja*, *Nein* und *Keine Ahnung* beschränken. Nach dem Drücken des entsprechenden Antwort-Buttons geht es weiter zur Rückmeldung.

2. Rückmeldung auf die vorangegangene Klassifizierung. Die Möglichkeiten setzen sich folgendermaßen zusammen:

   {Beispiel gehörte zum Konzept, Beispiel gehörte nicht zum Konzept} $\times$
   {Klassifizierung korrekt, Klassifizierung falsch, Klassifizierung nicht eindeutig}

   Mit einem Link soll der Proband dann weiter zum nächsten Schritt gelangen.

3. Angabe der Entscheidung zugrundeliegenden Regel. Zur Vereinfachung soll auch das Beispiel, welches als letztes bewertet wurde, angezeigt werden. Erst wenn eine komplette Regel angegeben wurde, kann der Benutzer nach einem Klick auf den Absende-Button zum nächsten Schritt übergehen. Hat der Proband an diesem Punkt bereits acht aufeinanderfolgende Beispiele richtig klassifiziert, ist die Lernphase beendet und es kann mit der Anwendungsphase begonnen werden. Ist dies noch nicht der Fall, so wird ein neues Beispiel angezeigt und die Abfolge beginnt wieder bei der Beispielanzeige.

2.1.3 Anwendungsphase

Die Anwendungsphase ist der Lernphase in 2.1.2 sehr ähnlich. Es sollen auf dieselbe Art und Weise Beispiele angezeigt werden. An diesem Punkt hat der Benutzer die Regel aber bereits gelernt. Somit können die Rückmeldung über die Korrektheit und die Seite zur Angabe der Regel entfallen. Im Unterschied zur Lernphase wird jetzt allerdings die Zeit gemessen, die der Benutzer benötigt um ein Beispiel zu klassifizieren. Anhand dieser Zeit soll später entschieden werden, ob Probanden mit einer komplexeren Regel länger für die Klassifizierung benötigen. Die drei Beispiele der Anwendungsphase sollen also mit einer kurzen Zwischenseite, die es dem Benutzer ermöglicht, kontrolliert mit dem nächsten Beispiel zu beginnen, aufeinanderfolgen.

2.1.4 Ende des Experiments

Nach dem Ende der Anwendungsphase soll der Teilnehmer über das erfolgreiche Ende seiner Eingaben informiert werden. Anders als in Reips (2002) soll er erst an dieser Stelle auf einer weiteren Seite die Möglichkeit bekommen, statistische Daten zu seiner Person anzugeben. Die Daten sollen

später am Ende angegeben werden, um zu vermeiden, dass potentielle Probanden durch die Angabe der Informationen am Anfang abgeschreckt werden und das Experiment womöglich gar nicht starten. Bei der Angabe am Ende hat der Benutzer das Experiment vollständig abgeschlossen und der Verlust der statistischen Daten an dieser Stelle ist verschmerzbar. Nach der Seite mit den statistischen Daten soll es außerdem möglich sein die Email-Adresse unabhängig von den Versuchsdaten zu hinterlassen, damit über den Ausgang des Experiments informiert werden kann.

2.2 Wahl der Realisierungsmittel

Für die Realisierung sollte, um später eine einfachere Auswertung der Ergebnisse mit Statistikprogrammen (wie z.B. SPSS[1]) zu ermöglichen, auf eine SQL-Datenbank zurückgegriffen werden.

Die Auswahl dieser und die Auswahl der Programmiersprache war durch die gegebene Infrastruktur beschränkt. Da die bestehende Serverlandschaft wenn möglich genutzt werden sollte, fiel die Wahl auf die im Internet häufig eingesetzte Kombination aus PHP, MySQL und Apache. Weitere Gründe für dieses Gespann waren die starken zeitlichen Einschränkungen für die Entwicklung und die schon bestehenden Erfahrungen mit den beiden Produkten. Die leichte Erlernbarkeit und die gute Dokumentation von PHP und MySQL würden auch weniger erfahrenen Programmierern einen schnellen Einstieg ermöglichen.

[1] http://www.spss.com/spss/

Kapitel 3

Realisierung

Die Realisierung ist in drei Teile untergliedert. In den *Herausforderungen* werden allgemein Probleme erläutert, die bei der Realisierung eines Online-Experiments auftreten können. Unter *struktureller Aufbau der Anwendung* wird anschließend allgemein in die aktuelle Implementierung eingeführt. Da durch diesen Aufbau einige Herausforderungen schon bewältigt werden und für die weiteren Realisierungsdetails ein Grundverständnis des Aufbaus unabdingbar ist, werden erst danach die *Lösungsmöglichkeiten* der *Herausforderungen* angesprochen. Jede Lösungsmöglichkeit ist dabei in einen allgemeinen Teil und die Realisierung am Beispiel der aktuellen Implementierung separiert.

3.1 Herausforderungen

3.1.1 Sichere Implementierung

Wie bei jeder Webanwendung, die einem großen Publikum zur Verfügung gestellt wird, ist auch bei einem Online-Experiment die Sicherheit der Anwendung ein zentraler Punkt der Entwicklung. Bei Online-Experimenten werden zwar selten hochgradig sensitive Informationen verwaltet, in der heutigen Flut von Spam-Mails reichen aber oft schon die zahlreichen gesammelten E-Mail-Adressen um einen Einbruch in die Datenbank rentabel erscheinen zu lassen.

Im vorliegenden Online-Experiment gibt es keine Möglichkeit sich zu registrieren oder einen geschützten Bereich mit für die Öffentlichkeit interessanten Informationen zu betreten. Angriffe, bei denen z.B. versucht wird, Cookies von eingeloggten Benutzern zu stehlen um dann ihre Session übernehmen zu können (z.B. mit Hilfe von Cross-Site-Scripting), fallen also weg. Einen derartigen Angriff vorzubereiten nur um die Experiment-Session eines Probanden zu manipulieren, ist hochgradig abwegig, weil der nötige Aufwand in keinerlei Relation zu dem Gewinn einer solchen Aktion steht.

Sinnvoller erscheint da schon, sich die Mühe zu machen mit Hilfe von SQL-Injection[1] vorhandene SQL-Statements so zu manipulieren, dass die Datenbank nicht für die Öffentlichkeit bestimmte Informationen preisgibt. Denkbar wäre hier beispielsweise, wie schon erwähnt, eine Liste aller E-Mail-Adressen. Noch interessanter werden derartige Manipulationen, wenn andere wichtige Applikationen mit der gleichen Datenbank arbeiten und diese nicht durch strikte Zugriffsrechte voneinander getrennt sind. Die Brisanz der Daten eines Online-Experiments mag noch harmlos erscheinen. Gibt ein derartiger Angriff aber dann zum Beispiel einem Angreifer auch die Möglichkeit auf den Datenbestand des E-Learning-Management-Systems des Lehrstuhls zuzugreifen und dort detaillierte Informationen über die registrierten Benutzer herauszufinden, ist die Tragweite schon eine ganz andere.

Ziel eines Online-Experiments sollte es deswegen sein Angriffe auf die Informationen der Probanden auszuschließen. Die Teilnahme an Online-Experimenten wird zumeist nicht entlohnt und baut auf das Vertrauen und die Reputation der jeweiligen Institution auf. Wird das Vertrauen durch mangelnde Sicherheitsstandards erschüttert, wirkt sich das auch auf spätere Experimente aus.

3.1.2 Konsistenz der Ergebnisse

Online-Experimente entziehen sich der direkten Kontrolle des Experimentleiters. Anders als bei einem persönlich beaufsichtigten Versuchsablauf, gibt es keine Möglichkeit später regulierend einzugreifen. Fehler im Versuchsablauf können nach der Freigabe nahezu gar nicht mehr aufgedeckt werden. Für die Leitung des Experiments bedeutet das, dass die Anwendung vorher noch ausgiebiger getestet werden muss. Hierbei muss aber nicht nur auf den möglichen korrekten Ablauf geachtet werden, sondern auch die Möglichkeit in Betracht gezogen werden, dass der Benutzer bewusst versucht den Ablauf zu stören und zu verändern. Mit Hilfe des Browsers und seinen Vor- und Zurück-Tasten beispielsweise ist ein Eingreifen von Benutzerseite schnell und unkompliziert möglich. Eine denkbare Motivation des Probanden wäre z.B. ein durch Mogeln schnelleres Durchführen eines Experiments, in dem - eventuell auch nur für den Teilnehmer selbst - die schnelle Bearbeitung im Vordergrund steht und sein Erfolg und seine Intelligenz für ihn eben daran gemessen werden. Sind Manipulationen am Ablauf des Experiments möglich, leidet natürlich auch ein zentrales Ziel - die möglichst hohe Zuverlässigkeit der Daten. Manipulationen sollten deswegen strikt ausgeschlossen werden. Im vorliegenden Experiment muss zusätzlich zu den Benutzereingaben die Zeit ausgewertet werden, die der Benutzer benötigt um die ihm vorgelegten Beispiele wie in 2.1.3 beschrieben zu klassifizieren, was zu einem weiteren nicht trivialen Problem führt.

3.1.3 Zeiterfassung

Um die Schwierigkeiten der Zeiterfassung in diesem Projekt zu verstehen, soll hier ein kurzer Einblick in das Client-Server-Modell gewählt werden. Für tiefere Einblicke sei auf Ferstl und Sinz (2001) verwiesen.

Wird eine Internetadresse aufgerufen, sendet der Client eine Anfrage an die Adresse des Servers. Je nach Art der Anfrage sendet der Server zum Beispiel eine Internetseite, die aus Text und aus Formatierungsangaben in Form von HTML-Tags besteht. Diese Seite wird vom Client empfangen,

[1] Eine gute Einführung in dieses Thema findet sich auf http://www.unixwiz.net/techtips/sql-injection.html

zwischengespeichert und mit Hilfe des Browsers interpretiert. Die HTML-Tags werden dabei in die durch sie beschriebene Formatierung umgewandelt. Zentraler Punkt für die Zeiterfassung an diesem System ist die Tatsache, dass nur auf Anfrage des Clients eine HTML-Seite vom Server gesendet werden kann. Die umgekehrte Form - also vom Server zum Client - der Kommunikation ist nicht möglich. Der kleinstmögliche Zeitabschnitt der auf dem Server gemessen werden kann, ist somit die Zeit zwischen zwei Anforderungen des Clients. Es ist zwar möglich, dem Client die angeforderte Internetseite für die Klassifizierung zu schicken, und die Zeit zu messen, bis sie wieder mittels eines Formulars an den Server geschickt wird. Diese Lösung kann aber nicht für eine Zeitmessung im Millisekundenbereich verwendet werden, da in dieser Größenordnung bereits die Geschwindigkeit der Internetanbindung des Clients Ergebnisse verfälschen kann. Probanden mit einer langsamen Internetanbindung wären bei der Antwortzeit Probanden mit einer schnelleren Internetanbindung unterlegen, weil ihre Antwort den Server langsamer erreicht. Für ein Experiment bedeutet das eine Unvergleichbarkeit der Daten, die die Ergebnisse beeinflussen kann und deswegen ausgeschlossen werden muss. Eine Herausforderung sind die nur vom Client ausgehenden Anfragen auch im Bereich der Interaktivität.

3.1.4 Hohe Interaktivität

Interaktivität, das heißt Reaktion der Webanwendung auf Eingaben des Benutzers und umgekehrt, stellt auch eine große Herausforderung an den Ersteller eines Online-Experiments. HTML, die Darstellungssprache[2] des World Wide Web, ist ein Standard um Text und Grafiken anzuordnen und zu formatieren. Reaktion auf Benutzereingaben auf der Serverseite ist hierbei nicht vorgesehen und wird erst durch eine Kombination aus HTML und einer Programmiersprache wie z.B. PHP, Java oder Perl möglich. Nur durch ein erneutes Laden der kompletten Internetseite, kann eine Reaktion auf Benutzereingaben sichtbar gemacht werden. Im vorliegenden Experiment ist der Interaktionsgrad mit dem Benutzer wie in 2.1.2 beschrieben sehr hoch. Auf nahezu jeder Seite sollen vom Probanden Eingaben getätigt werden, die sich auf den weiteren Verlauf des Experiments auswirken. All diese Daten müssen im Sinne der vorher betrachteten Konsistenz und unter dem Sicherheitsaspekt ausgewertet werden. Für die Entwicklung bedeutet das einen Anstieg von Komplexität und Umfang. Je komplexer und umfangreicher ein Programm wird, desto größer ist auch die Wahrscheinlichkeit Fehler zu produzieren, wenn nicht für zusätzliche Transparenz durch die Software-Architektur (siehe 3.2.5) gesorgt wird.

[2] Wie in http://de.wikipedia.org/wiki/Programmiersprache beschrieben, teilt der Autor die Auffassung, dass mit einer Programmiersprache zumindest die grundlegende mathematische Arithmetik ausgedrückt werden können sollte. HTML ist demnach keine Programmiersprache und wird in diesem Bericht als eine Darstellungssprache bezeichnet.

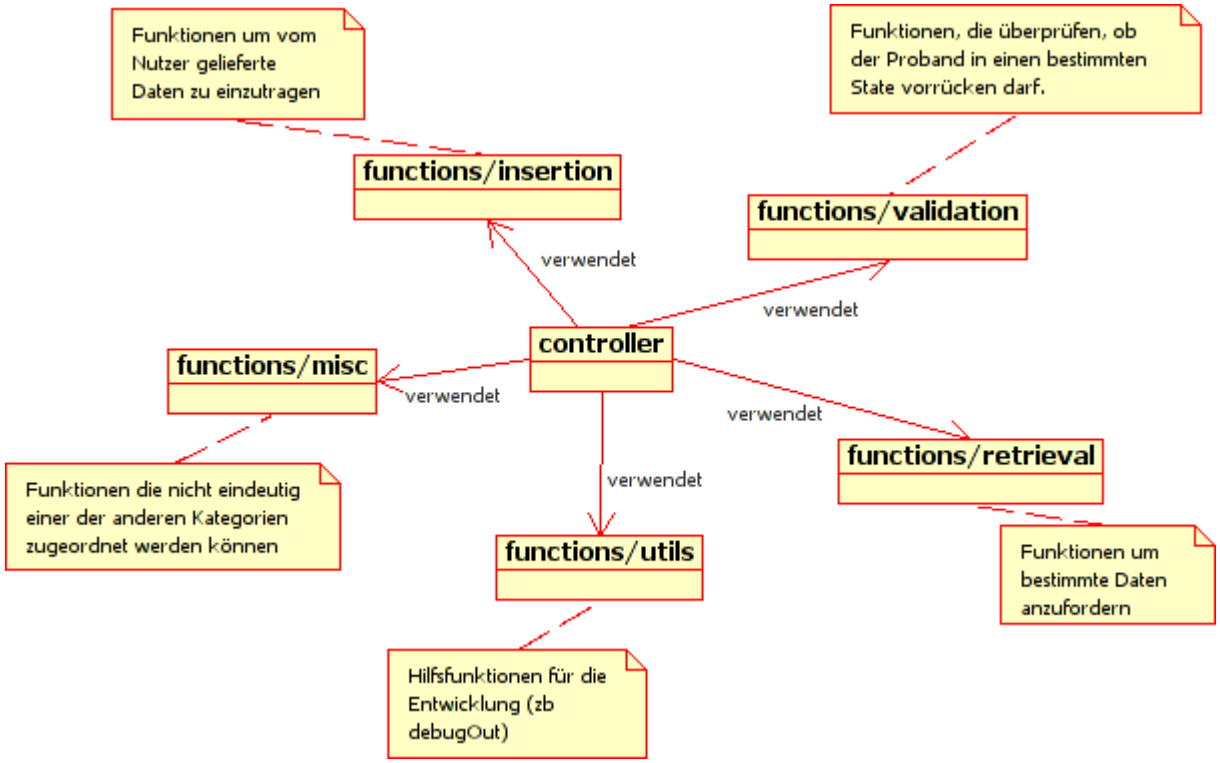

Abbildung 3.1: Funktionsverteilung

3.2 Struktureller Aufbau der Anwendung

3.2.1 Dateistruktur

Für eine bessere Orientierung im Quellcode sind in Tabelle 3.1 wichtige Strukturelemente kurz erklärt und in Abbildung 3.1 illustriert.

3.2.2 Standardkonformität

In der Einleitung wurde erwähnt, dass ein großer Vorteil von Online-Experimenten die leichte Verbreitung und der nahezu uneingeschränkt große Probandenkreis ist. Im Gegensatz zu stationären Experimenten, die oft auch noch beaufsichtigt werden müssen, fällt während des Experimentverlaufs keinerlei Arbeit an. Um dieses große Publikum nicht unnötig einzuschränken, sollte das Ex-

Datei/Ordner	Beschreibung
README	Installationsanleitung
beispiel.html	Beispiele von Buchstabenkombinationen (Einleitung des Experiments)
controller.php	Vom Benutzer aufgerufen und für die komplette Ablaufsteuerung während des Experiments verantwortlich
css	Ordner für Stylesheets
expstart.html	Kurze Anweisungen unmittelbar vor dem Experiment (Einleitung des Experiments)
functions	thematisch geordnete Anwendungsfunktionen
functions.php	Inkludiert alle weiteren Dateien in functions/. So muss nur diese Datei im Header inkludiert werden um alle Funktionen benutzen zu können => Alle anderen Dateien werden in der Grafik 'Funktionsverteilung' beschrieben
images	Ordner für Bilder
impressum.htm	Impressum
includes	HTML-Fragmente, die vom controller je nach State des Benutzers eingefügt werden. Namen sind selbsterklärend
index.htm	Nach der Eingabe der Experiment-URL, wird dem Probanden als erstes diese Seite angezeigt
javascript	JavaScript-Dateien

Tabelle 3.1: Dateistruktur

periment mit jedem handelsüblichen Browser der neuesten Generation durchführbar sein. Eine Möglichkeit das zu garantieren, ist die Anpassung an einen Standard. Im aktuellen Beispiel ist das die XML-konforme Implementierung von HTML, 'XHTML 1.0', die vom W3C (World Wide Web Konsortium[3] entworfen wurde. Mit Hilfe des Markup Validation Service[4] ist es bequem möglich die eigenen HTML-Seiten auf Konformitätsfehler zu überprüfen. Sind die HTML-Seiten dann an diesen Standard angepasst, sollten mit keinem XHTML unterstützenden Browser Darstellungsprobleme auftreten. Tests des Experiments mit allen aktuellen Browsern sollten aber trotzdem durchgeführt werden um Layoutfeinheiten überprüfen zu können.

3.2.3 Implementierungshilfen

Je umfangreicher eine Anwendungssoftware wird, desto größer ist oft auch die Wahrscheinlichkeit, dass sich Fehler einschleichen. Ein klar strukturierter Quellcode (siehe 3.2.5) mit übersichtlichem Aufbau sorgt in solchen Fällen dafür, dass die Fehlerwahrschlichkeit auf einem niedrigen Niveau gehalten werden kann. Vollkommen vermeiden lassen sich so genannte Bugs aber nahezu nie. Problematisch wird es dann, wenn Fehler durch schlechte Software-Architektur nicht entdeckt oder nicht zurückverfolgt werden können. Eine Beseitigung ist dann oft nur unter erheblichem Aufwand möglich. Ein ausgefeiltes Exception-Handling und Optionen um beim Ablauf zusätzliche, für den Fehler suchenden Programmierer sinnvolle Informationen einzublenden, ermöglichen es auftretende Fehler schnell zu lokalisieren und mit vertretbarem Aufwand zu beseitigen.

In diesem Online-Experiment wurde dafür eine einfache, dem Umfang des Projekts angemessene Lösung gewählt. Die beiden Funktionen `errorOut($output)` und `debugOut($output)` aus `functions/utils.php` ermöglichen es aus dem Quellcode auf sehr einfache Art und Weise Fehlermeldungen und Informationen in der Ansichtsebene darzustellen. `errorOut` ist dabei für die Ausgabe von Fehlern, die vom Benutzer produziert wurden, zuständig und wird als roter Balken am oberen Ende der HTML-Seite angezeigt. `debugOut` hingegen dient zur zusätzliche Information des Entwicklers. Wenn in der `includes/config.php` die statische Variable `DEBUG` auf `TRUE` gesetzt wurde, erscheinen diese Informationen als grüner Balken am oberen Ende der HTML-Seite.

Innerhalb von `config.php` lassen sich noch weitere Einstellungen an zentraler Stelle vornehmen. Alle diese Einstellungen hängen aber unmittelbar mit der Installation der Scripten zusammen und eine Erklärung ihrer Funktion ist deshalb direkt dem Quellcode als `README`-Datei beigefügt. Erwähnenswert ist lediglich die Möglichkeit in dieser Webanwendung sämtliche Dateisystempfade und Ordnernamen an einer Stelle konfigurieren zu können. Die sonst fehlerträchtige Anpassung an jeder Stelle, an der diese Angaben verwendet werden, entfällt somit.

3.2.4 Session-Management

HTTP (Hyper Text Transport Protocol[5]) ist ein zustandsloses Übertragungsprotokoll um vom Server an Client (Browser) angeforderte Daten über eine Verbindung zu schicken. Die Verbindung zwischen Client und Server besteht nur so lange, wie die Übertragung andauert. Danach wird die

[3] Mehr Informationen unter http://www.w3.org/

[4] http://validator.w3.org/

[5] Mehr Informationen unter http://www.answers.com/topic/http

Verbindung sofort wieder geschlossen. Der Server weiß also nur für die Zeit einer Datenanforderung, an wen er die Daten schicken muss. Danach 'vergisst' er diese Information wieder. Mit Hilfe von HTTP ist es deswegen nicht möglich Clients über mehrere Anfragen (Session genannt) hinweg zu identifizieren.

Auf anderem Wege ist das auf zweierlei Arten möglich. Einerseits mit Hilfe einer so genannten SessionId, die an jede aufgerufene URL angehängt wird und so den Benutzer auf Serverseite über mehrere Anfragen hinweg eindeutig identifizieren kann. Da die Adresse in der Adressleiste des Browsers aber sehr leicht vom Benutzer verändert werden kann und kopierte Links der aktuellen Seite, auf der der Client sich befindet, auch die SessionId enthalten, ist es nicht unbedingt ratsam diese Methode zu wählen.

Die zweite Möglichkeit verwendet Cookies. Das sind kleine in ihrer Größe sehr beschränkte Textdateien auf dem Client, die vom Server aus geschrieben werden können. In ihnen kann, die vorher schon erwähnte SessionId des jeweiligen Benutzers gespeichert und vom Server bei jeder Anfrage dieses Clients wieder ausgelesen werden. Diese kleinen Textdateien können natürlich auch vom Benutzer manipuliert werden. Jedoch ist dies wesentlich umständlicher.

Implementierung am aktuellen Beispiel Das aktuelle Projekt verwendet aus diesen Gründen Cookies um den jeweiligen Benutzer während des Experiments eindeutig zu identifizieren. Dazu wird mit Hilfe der folgenden Methode in der Datei `includes/header.php` überprüft, ob schon ein Cookie vorhanden ist. Ist dies der Fall, wird die ausgelesene SessionId unter der Variablen `$session_id` abgelegt. Ansonsten wird ein neuer Cookie mit einer über einen md5-String aus einer Zufallszahl generierten SessionId angelegt.

```
if (!isset ($_COOKIE['session_id'])) {

 $session_id = md5(uniqid(rand()));

 setcookie("session_id", $session_id, time() + 5184000);

} else {

 $session_id = $_COOKIE["session_id"];

}
```

Über den Zeitraum des Experiments hinweg kann der Proband jetzt identifiziert werden. Cookies haben aber noch den weiteren Vorteil, dass sie nicht unmittelbar nach dem Beenden des Browsers gelöscht werden müssen. Durch ein Verfallsdatum ist es möglich diesen Zeitpunkt beliebig zu beeinflussen. In der oben genannten Funktion wird das Verfallsdatum auf 60 Tage (5184000s) in die Zukunft verlegt. Wenn der Cookie nicht vom Benutzer manuell gelöscht wird, kann also bis zu 60 Tage nach dem Experiment festgestellt werden, dass der Proband das Experiment im Vorfeld schon einmal beendet hatte. Er wird dann nicht erneut die Möglichkeit bekommen am Experiment teilzunehmen, weil sich seine Session schon im Zustand 'Experiment beendet' befindet.

3.2.5 Architektur

Die Software-Architektur des Experiments musste sich stark an den besonderen Gegebenheiten orientieren. An nahezu jeder Station und Zwischenstation des Experiments war es nötig Daten, die vorher durch Eingaben des Probanden generiert worden waren, zu verarbeiten. Um bei den insgesamt neun Stationen (siehe 2.1) und den dazugehörigen Daten den Überblick nicht zu verlieren wurde das Model-View-Controller[6] Pattern gewählt. Dieses Pattern setzt sich eigentlich aus mehreren Patterns zusammen. Die View und Controller-Komponente ähneln sehr dem Front-Controller Pattern. Eine ausführliche Beschreibung dafür ist in Alur, Crupi und Malks (2002) zu finden. Beim MVC-Pattern werden Komponenten für die Benutzerinteraktion (Controller), die Anzeige (View) und die Operation mit der Datenschicht (Model) möglichst stark voneinander getrennt. Wird in der Anzeige eine Funktion aufgerufen, geht dieser Aufruf an den Controller, der dann je nach Zustand der aktuellen Anwendung entscheidet, welche Methoden des Models verwendet werden müssen. In Abhängigkeit davon, wird schließlich die Anzeigekomponente aktualisiert. Diese Architektur lässt sich am besten mit einer objektorientierten Sprache verwirklichen. Auch in nicht-objektorientiertem PHP können die Ansätze aber genutzt werden um eine Anwendung übersichtlich zu gestalten.
Die Hauptvorteile sind:

- Anwendungslogik und Anzeige sind - wo möglich - voneinander getrennt. Bei Webapplikationen können Layoutverantwortliche so weitgehend unabhängig von Programmierern arbeiten.
- Die Fehleranfälligkeit sinkt, da Code im Controller zentralisiert und wiederverwendet werden kann und nicht fehlerträchtig in jeden Teil der Ansicht kopiert werden muss.

Implementierung am aktuellen Beispiel In der vorliegenden Anwendung übernimmt die Kontrolle der Anzeigekomponenten das Script `controller.php`. Alle Anfragen des Benutzers werden an dieses Script gerichtet. Je nach Bearbeitungszustand des Benutzers werden dann mit Hilfe des `include_once()`-Befehls die für den Zustand nötigen HTML-Fragmente aus `/includes/` geladen. Um zu verhindern, dass ein findiger Benutzer diese Seiten direkt anspricht, ist das `include`-Verzeichnis über eine Apache-`.htaccess`-Datei vor direktem Zugriff geschützt. Jede HTML-Seite des Experiments setzt sich aus drei Teilen zusammen. Im controller wird als erstes `includes/header.php` und danach die jeweilige HTML-Teilseite im switch-Block am Ende eingefügt. Der dritte Teil kann im controller entweder am Ende aus einem Footer bestehen, oder einfach die in den beiden vorher eingefügten Seiten geöffneten HTML-Tags schließen. In diesem Fall werden `</body>` und `</html>` geschlossen, so dass am Ende eine valide HTML-Seite entsteht.

3.3 Lösungsmöglichkeiten

3.3.1 Sichere Implementierung

Um mit einer datenbankbasierten Webanwendung nicht unwissentlich auch andere Bereiche (siehe 3.1.1) der Öffentlichkeit Preis zu geben, ist es ratsam die verschiedenen Applikationen auf Datenbankebene sorgfältig voneinander zu trennen. Das ist in diesem Fall bei MySQL durch strenge

[6] Mehr Informationen unter http://ootips.org/mvc-pattern.html

Rechtevergabe möglich. Die Anwendung läuft auf Datenbankebene in einem eigenen Datenbankschema unter einem neu angelegten Benutzer, der nur auf dieses Schema Zugriff hat. Im Falle eines Einbruchs wäre es dem Einbrecher nur möglich in diesem Bereich Veränderungen vorzunehmen oder Daten auszulesen.

Nachdem die Gefahr so auf die aktuelle Anwendung beschränkt wurde, gilt es andere Methoden der Datenkompromittierung zu verhindern. Weit verbreitet und ohne große Hilfsmittel möglich ist die schon erwähnte SQL-Injection. Die Methode soll hier kurz an einem einfachen Beispiel demonstriert werden. In dem vorliegenden Experiment soll es wie in 2.1.4 beschrieben später für den Benutzer auch die Möglichkeit geben seine Email-Adresse anzugeben um später über den Ausgang des Experiments informiert zu werden. Das SQL-Statement, das später an das Datenbankmanagementsystem übermittelt wird, könnte nun folgendermaßen aussehen:

```
INSERT INTO email (email) VALUES ('$_POST[email]');
```

`$_POST[email]` steht hierbei für die Daten die aus dem Eingabefeld für die Email-Adresse ausgelesen wurden. Der Einfachheit halber soll bei diesem Beispiel auf eine Validierung der Email-Adresse verzichtet werden. Statt seiner Email-Adresse kann ein potentieller Eindringling nun versuchen, das Formular anderweitig auszunutzen. Man nehme an, der Benutzer gibt folgendes in das Formularfeld ein. `email'); DROP TABLE users;`. Zusammengesetzt ergibt sich dann:

```
INSERT INTO email (email) VALUES ('email'); DROP TABLE users;);
```

So wäre die Tabelle mit den mühsam gesammelten Benutzerdaten gelöscht. Natürlich kann man auf diese Art und Weise, je nach Art der Eingabemöglichkeit auch SELECT-Statements und andere einschleusen. Vom Prinzip ausgehend gleichen sich jedoch alle Möglichkeiten.

Ein einfach zu realisierender Ansatz um derartige Einbrüche zu verhindern oder zumindest zu erschweren, ist nur Zeichen einer vorgegebenen Menge zuzulassen. Sobald andere Zeichen vom Eingabefeld übermittelt werden, wird das SQL-Statement nicht ausgeführt und eine Fehlermeldung ausgegeben. Hochkommata und Klammern wären in dieser Menge nicht erlaubt. Diese kleine Begrenzung würde das oben genannte Beispiel schon unmöglich machen. Die Methode nur bestimmte Zeichen zuzulassen wurde auch bei diesem Experiment gewählt. Potentielle Einbrecher sollten nicht auf einfache Art und Weise die Daten manipulieren können. Im Hinblick auf die kommerziell relativ uninteressanten Daten dieses Experiments, sollte dieser Schritt genügen.

Implementierung am aktuellen Beispiel Im vorliegenden Experiment ist die Realisierung einer Whitelist der möglichen Zeichen relativ einfach umzusetzen. Gespeichert werden in PHP mit Formularen mit der Methode 'POST' übertragene Daten im globalen Array `$_POST['key']` des im Formular im Action-Feld erwähnten Scripts. 'key' spezifiziert hierbei den Namen des HTML-Input-Felds, was ausgelesen werden soll. Bevor eine HTML-Seite angezeigt wird, wird nun in `includes/header.php` jeder Eintrag dieses Arrays auf unerlaubte Zeichen mittels eines regulären Ausdrucks überprüft. Werden unerlaubte Zeichen gefunden, bricht die Ausführung des Scripts mit einer Fehlermeldung ab. Wie der Name schon sagt, werden in der header-Datei die Kopfdaten der HTML-Seite hinzugefügt. Jede Experimentseite, die Daten empfängt, enthält den Header und somit auch die Sicherungsfunktion.

Erschweren lässt sich SQL-Injection auch durch das Unterdrücken der Standardfehlermeldungen der Datenbank. Diese geben oft aufschlussreiche Informationen über die Tabellenstruktur preis. Da

der Angreifer beim Einschleusen von Code für seine SQL-Statements diese Struktur in Erfahrung bringen muss, wird ihm das bei nichtssagenden Fehlermeldungen erschwert. In PHP lassen sich Datenbankfehlermeldungen durch ein vorangestelltes `@` vor den MySQL-Funktionen unterdrücken (z.B. `@mysql_query`). Im DEBUG-Mode (siehe 3.2.3) dieses Experiments werden die Informationen für Entwickler jedoch angezeigt.

3.3.2 Konsistenz der Ergebnisse

Manipulationen gibt es natürlich auch noch auf anderer Ebene. Bei Experimenten ist es oft gar nicht nötig den Datenbestand direkt auf der Datenschicht zu manipulieren. In zahlreichen Fällen ist es schon ausreichend den Ablauf durch Benutzen von Standardfunktionen durcheinander zu bringen. Wie in 2.1.2 beschrieben besteht die Lernphase aus einer Iteration folgender Zwischenstationen:

1. Präsentation des Beispiels und Klassifizierung
2. Rückmeldung
3. Regelangabe

Enthält die Rückmeldung beispielsweise, das Ergebnis, dass der Benutzer mit seiner Klassifizierung falsch lag, könnte er auf den Gedanken kommen, einfach den Zurück-Button des Browsers zu benutzen um das zuerst falsch klassifizierte Beispiel mit dem neuen Wissen nun richtig zu klassifizieren. Je nach Art der Speicherung der Experimentdaten kann ein solches Eingreifen mehrfache Datensätze zu einem Beispiel zur Folge haben, wenn nach einer Klassifizierung das Ergebnis ohne Überprüfung in den Datenbestand übernommen wird. Diese Art der Manipulation kann natürlich in anderen Phasen des Experiments gleichermaßen genutzt werden. Um das zu verhindern, ist es nötig vor jeder Änderung des Datenbestands zu überprüfen, ob die Änderung im aktuellen Zustand des Experiments möglich sein darf.

Implementierung am aktuellen Beispiel Diese Überprüfung übernehmen im vorliegenden Fall Validierungsmethoden in `functions/validation.php`. Sie überprüfen vor jedem Setzen des Benutzers in einen neuen `state`, ob alle Voraussetzungen dafür erfüllt sind. Die komplexesten sind hierbei `validateLearningUpdate` und `validateExerciseUpdate`. Innerhalb dieser Methoden muss überprüft werden, ob das Beispiel, zu dem der Proband eine Klassifizierung abgibt, auch das Beispiel ist, was nach der Reihenfolge an der Reihe ist. Somit muss zum einen überprüft werden, ob die Beispiele aufeinanderfolgen, zum anderen ob zu dem Beispiel schon eine Klassifizierung existiert. Nicht so kompliziert gestaltet sich das im Falle Validierung der Angabe der Regel. Hier muss nur überprüft werden, dass alle Buttons gesetzt sind. Für das Experiment ist es nicht weiter relevant, ob der Proband die schon gesetzte Regel durch den Zurückschritt im Browser noch einmal überschreibt, da das jeweils nur für das aktuelle Beispiel möglich ist. Unmöglich ist also zum Beispiel das Regelfeedback des vorletzten Beispiels im Nachhinein zu verändern.

3.3.3 Zeiterfassung

In 3.1.3 wurde darauf hingewiesen, dass eine akkurate Zeiterfassung von Serverseite aus nicht möglich ist, da der kleinste messbare Zeitabschnitt sich zwischen zwei Client-Server-Anfragen befindet.

Durch die zusätzliche Verzögerung die durch den Versand von Client zu Server entsteht und die unterschiedlichen Internetanbindungen der Nutzer ist eine genaue Messung so nicht möglich. Die in 2.1.3 erwähnten Zeiten für die Klassifizierung müssen aber sehr genau gemessen werden, um Unterschiede deutlich herausstellen zu können. Im vorliegenden Experiment wurde deshalb ein Ansatz gewählt, der es ermöglicht sowohl auf Client- als auch auf Serverseite die Zeiterfassung vorzunehmen. Auf Clientseite wird dies mit Hilfe von JavaScript[7] realisiert. JavaScript wird direkt mit Hilfe des Browsers ausgeführt. Ein Server oder eine Internetanbindung sind dafür nicht erforderlich, was sowohl Vorteile wie auch Nachteile mit sich bringt. Die für uns hier wichtigen Vorteile sind die clientseitige Ausführung des Scripts, die eine sehr akkurate Zeitmessung ermöglicht. Die Zeit startet so erst, wenn die Internetseite vollständig beim Client geladen ist. Gestoppt wird die Zeit auch auf Clientseite, sobald einer der Buttons für die Klassifizierung gedrückt wurde. Die Übertragungszeit zum Server geht so nicht mit in die Zeitmessung ein. Problematisch an der clientseitigen Ausrichtung ist allerdings die Tatsache, dass der Proband volle Kontrolle über seinen Browser und die Ausführung von JavaScript hat. Manipulativ in die Zeitmessung kann hier durch das Drücken der Neu-Laden-Taste im Browser sehr leicht eingegriffen werden. Die Zeit startet durch das erneute Laden der Beispielanzeige wieder von vorne und der Benutzer kann sich so künstlich gute Klassifizierungszeiten verschaffen. Um eine derartige Manipulation erkennen zu können, wird zusätzlich die Zeit gespeichert zu der die Seite mit dem Anwendungsbeispiel an den Benutzer gesendet wurde und die Zeit zu der die Klassifizierung vom Server empfangen wurde. Weichen die serverseitigen Zeiten erheblich von den clientseitigen ab, können diese Ergebnisse später aussortiert werden.

Implementierung am aktuellen Beispiel Im vorliegenden Fall wird beim Laden der Seite mit dem Anwendungsbeispiel die Funktion `setStartTime()` aus `javascript/js.js` durch die Zeile `<body onload="setStartTime()">` in `includes/question.php` aufgerufen. Beim Drücken eines der Klassifikations-Buttons wird daraufhin mit Hilfe der Funktion `setPassedTime()` aus der JavaScript-Datei die Differenz aus dem Startwert und dem Endwert berechnet und mit Hilfe von `document.forms["abfrage"].elements["passedTime"].setAttribute("value", passedTime);` in das Formularfeld `passedTime` aus der Datei `includes/question.php` geschrieben und später mit komplettem HTML-Formular durch eine der `voteXX()` Funktionen an den Server geschickt und dort ausgewertet. JavaScript wird von verschiedenen Browsern unterschiedlich unterstützt. Manche Funktionen sind nicht bei allen Browsern verfügbar. Deswegen ist es normalerweise ratsam JavaScript für Internetseiten, die möglichst vielen Benutzern vollständig zur Verfügung stehen sollen, nicht zu verwenden. In diesem Experiment wurde aus diesem Grund darauf geachtet browserübergreifende Funktionen aus JavaScript zu verwenden. Zusätzlich wurde die Implementierung noch mit den bekanntesten Browsern (Opera, Internet Explorer, Firefox, Safari und Konqueror) auf ihre Funktionsfähigkeit überprüft. Probanden, die JavaScript komplett deaktiviert haben, werden beim Aufrufen der Startseite des Experiments (`index.htm`) darauf hingewiesen JavaScript zu aktivieren. Da das weitere Experiment in einem durch JavaScript realisierten Popup geöffnet wird, können Teilnehmer ohne aktiviertem JavaScript auch nicht ohne Weiteres mit dem Experiment beginnen.

Die serverseitige Zeitmessung konnte auch auf relativ einfache Weise implementiert werden. Hierzu wird unmittelbar nach dem Einfügen von `question.php` in der `switch`-Anweisung in `controller.php` die Funktion `insertStartingTimestamp` aus `functions/insertion.php` aufgerufen. Sie schreibt in die Datenbanktabelle `starting_timestamp` ein Tupel aus `id`, `user_id`,

[7] Mehr über JavaScript unter http://de.selfhtml.org/javascript/index.htm

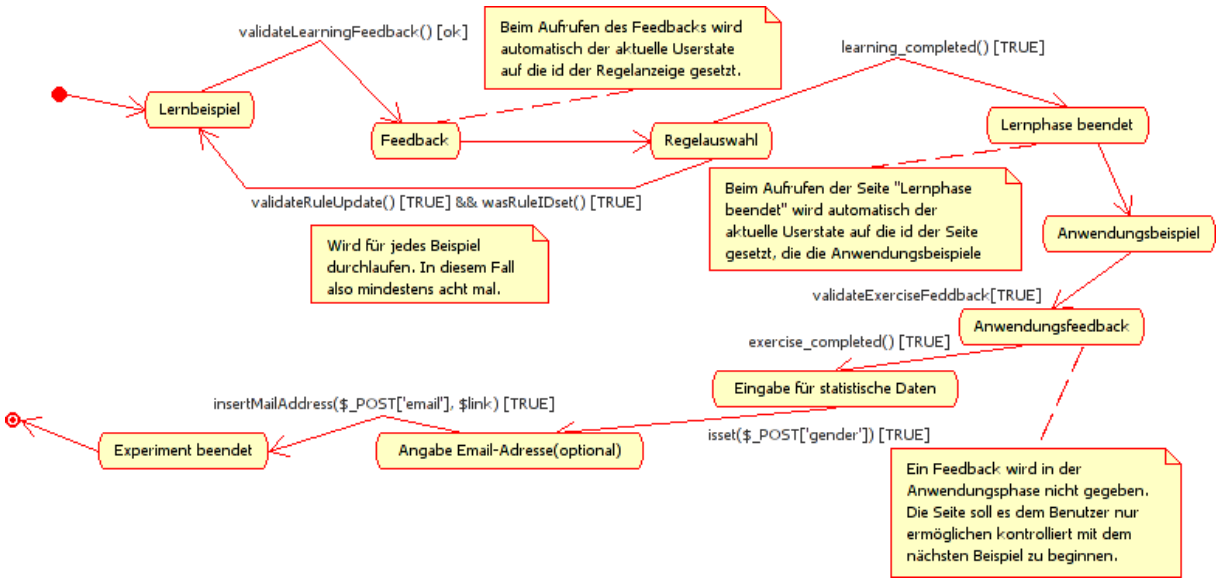

Abbildung 3.2: Zustandsdiagramm

`example_id` und `start_timestamp`, das später über die `user_id` und die Beispielnummer mit dem Beispieltupel aus der Tabelle `learning` verknüpft werden kann. Ein Beispieltupel in der Tabelle `learning`, wird immer dann geschrieben, wenn eine Klassifizierung abgegeben wurde. Der dabei gespeicherte Zeitstempel ist also der Endzeitstempel. Später kann die Differenz dieser beiden Werte dann genutzt werden um den auch in der Tabelle `learning` gespeicherten JavaScript-timestamp in `learning_timestamp` auf Validität zu überprüfen.

3.3.4 Hohe Interaktivität

Um der wie in 3.1.4 beschriebenen, zusätzlichen Komplexität Rechnung zu tragen, war ein ausgeklügeltes Konzept notwendig. Grundstein dafür legte das in 3.2.5 näher beschriebene Model-View-Controller-Pattern in dem alle Webanforderungen zentral gesteuert und verwaltet werden.

Implementierung am aktuellen Beispiel Diese Aufgabe übernimmt in dieser Webanwendung ausschließlich `controller.php`. Beim Start des Experiments ruft der Benutzer `controller.php` auf. In Abbildung 3.2 ist dann zu sehen, welche Funktionen erfolgreich durchlaufen werden müssen, um in den nächsten Zustand vorzurücken. Nach dem Aufruf von `controller.php` werden jeweils `updateUserState` und `$temp_user_state_id` mit der id des neuen Zustands aufgerufen bzw. auf die id des neuen Zustands gesetzt. `updateUserState` trägt die Änderung in das Feld `user_state_id` des jeweiligen Benutzers in der Datenbank ein. Anhand von `$temp_user_state_id` wird am Ende von `controller.php` mit einer `switch`-Anweisung die einzufügende Datei bestimmt. Normalerweise wird `$temp_user_state_id` für das Script `controller.php` mit Hilfe der Funktion

`getUserStateID` zu Beginn des Scripts gesetzt. Durch Daten, die mit Hilfe eines Formulars an `controller.php` geschickt wurden (z.B. durch die Regelfeedback-Seite), ist es allerdings möglich, dass sich der Zustand des Benutzers während der Ausführung des Scripts ändert. Wäre die state-ID nicht zwischengespeichert, müsste der Wert mit `updateUserState` in der Datenbank gesichert und am Ende von `controller.php` wieder über die Datenbank mit Hilfe von `getUserStateID` ausgelesen werden, was die Performance durch eine zusätzliche Datenbankabfrage nur unnötig verschlechtern würde. Wurde der entsprechende state des Benutzers ermittelt, können die nötigen HTML-Fragmente aus `includes` eingebettet werden.

Kapitel 4

Perspektiven

Nach mehr als 1500 Codezeilen und zahlreichen Arbeitsstunden stellt sich immer auch die Frage, ob ein derartiges Projekt nicht auch mit weniger Aufwand hätte realisiert werden können. Es wäre auch denkbar gewesen eine Lösung mit einem Window-Toolkit wie Java-Swing, GTK oder Windows-Forms zu realisieren. Bei Java mit Swing hätte nicht einmal auf die Distribution über das Internet verzichtet werden müssen, da diese Lösung auch innerhalb des Browsers als Java-Applet gestartet werden kann. Java braucht zur Ausführung auf dem Client-PC allerdings eine Virtual Machine, die erst installiert werden muss. Der mögliche Benutzerkreis einer solchen Lösung würde also wieder unnötig eingeschränkt werden. Lösungen mit anderen Programmiersprachen und Window-Toolkits erfordern eine Installation von Seiten des Benutzers. Für ein freiwilliges, unentgeltliches Experiment eine Hürde die sicher nicht jeder bereit ist zu nehmen. Außerdem müssten auch in diesem Fall die Experimentdaten möglichst an einem zentralen Ort gesichert werden. Die Kommunikation der Anwendung mit der Datenbank könnte in derartigen Fällen auf dem Client-PC durch eine Personal Firewall eingeschränkt werden, was den Kreis an möglichen Probanden wiederum einschränken würde. Natürlich hätte das Experiment auch auf Rechnern vorinstalliert werden können und dann unter Beaufsichtigung der Ersteller des Experiments durchgeführt werden, was mit erheblichen größeren Arbeitsaufwand für die Aufsicht und das Finden von Teilnehmern verbunden gewesen wäre. Unter dieser Durchführungsart würde an der Universität wahrscheinlich auch die Repräsentativität leiden, da erfahrungsgemäß an solchen Experimenten nahezu ausschließlich Studenten, die oft noch aus dem gleichen Fachbereich stammen, teilnehmen.

Auch im Nachhinein überwiegen also positive Argumente wie die potentiell größere Menge von Probanden, der geringe Betreuungsaufwand während des Experiments und die somit kleinen personellen Anforderungen gegenüber dem Zeitaufwand einer komplexen Implementierung. Möglicherweise können Codebruchstücke sogar direkt für andere Experimente wiederverwendet werden. Anpassungen an der aktuellen Implementierung sind leicht möglich, da durch die Architektur zum Beispiel relativ einfach neue User-States hinzugefügt oder entfernt werden können (siehe 3.3.4). Ein mögliches neues Experiment könnte beispielsweise versuchen zu prüfen, ob unterschiedliche Abfolgen von

Trainingsbeispielen sich auch auf die Anzahl der Beispiele auswirken, die der Benutzer benötigt um das Konzept zu erlernen. Ohne große Änderungen können in diesem Experiment die Abfolge und Anzahl der Trainingsbeispiele in der Datenbank (Tabelle `learning_sets`) und die Anzahl und Art der Regelattribute und Regeln (Tabelle `rules`) verändert werden. Für ein neues Online-Experiment können Aufbau und Implementierung in jedem Fall ein guter Anhaltspunkt sein.

Literaturverzeichnis

Alur, D., Crupi, J. Malks, D. (2002). Core J2EE Patterns. In (Kap. 7.2). München: Markt+Technik Verlag.

Birnbaum, M. H. (2000). *Psychological experiments on the internet.* San Diego: Academic Press.

Ferstl, O. K. Sinz, E. J. (2001). Grundlagen der Wirtschaftsinformatik. In (Kap. 8.2.7). München, Wien: R. Oldenburg Verlag.

Reips, U.-D. (2002). Five Dos and Five Don'ts. *Social Science Computerreview*, *20*(3), 241–249.

Unger, S. Wysotzki, F. (1981). *Lernfähige Klassifizierungssysteme.* Berlin: Akademie Verlag.