Programmierparadigmen mit Scheme

Christian Wagenknecht

Fachbereich Informatik
Hochschule Zittau/Görlitz (FH)
Brückenstr. 1
02826 Görlitz
c.wagenknecht@hs-zigr.de

Veit Berger

Fachschaft Informatik
Geschwister-Scholl-Gymnasium Löbau
Pestalozzistr. 21
02708 Löbau
v.berger@gmx.net

Abstract: Der Beitrag diskutiert Erfahrungen aus einem Kurs, in dem zahlreiche Inhalte des Informatiklehrplanes der Klassenstufen 11 und 12 mit funktionsorientierter Programmierung handlungsorientiert vermittelt werden. Das betrachtete Themenspektrum geht dabei über das hinaus, was üblicherweise unter Programmierparadigmen subsumiert wird. Innovativ ist die Behandlung dieser Themenvielfalt ausschließlich mit Scheme als sprachlichen Träger. Dies erlaubt, die charakteristischen Eigenschaften alternativer Programmier- und Denkstile sowie tragende Konzepte der Informatik in ein und derselben Programmierumgebung herauszuarbeiten. Es ist vorgesehen, die hier diskutierte fachdidaktische Konzeption durch eine speziell entwickelte Lernumgebung mediendidaktisch zu begleiten.

1 Einführung

Informatische Bildung ist Allgemeinbildung, die für die komplexen Anforderungen im beruflichen und gesellschaftlichen Alltag unerlässlich geworden ist. Mit der Reform der sächsischen Lehrpläne ([LP04]) wurde dieser Aspekt besonders berücksichtigt. Im Eckwertepapier zur informatischen Bildung ([EW02]) wird deshalb für Schüler der gymnasialen Oberstufe formuliert: „ ... Während der Bearbeitung größerer Projekte lernen sie, in der Fachsprache zu argumentieren, grundlegende Konzepte in der Informatik zu erläutern, Gestaltungsaufgaben zu beschreiben und komplexe Anwendungen und Aufgaben zu analysieren. Die Anwendung der Fachsprache zielt insbesondere auf das in dieser Altersstufe vorhandene Abstraktionsvermögen und erwartet von den Schülern Interpretationen und Begründungen im fachlichen Kontext. ... “

Insbesondere *funktionsorientierte* Programmiersprachen bieten mit ihrem *deskriptiven* Konzept geeignete didaktische Ansätze, um den o. g. Anforderungen in einem Grund- bzw. Wahlgrundkurs der Jahrgangsstufen 11/12 nachzukommen. Auf der Basis eines Lehrbuches [CW04] für Informatik-Studierende im 1. Semester entstand angepasstes Unterrichtsmaterial [CL04]. Das entwickelte Material wurde in den Schuljahren 2003/04 und 2004/05 in je einem Wahlgrundkurs der 11. Klasse erprobt.

Die Verwendung der (Turing-vollständigen) Programmiersprache Scheme für ausgewählte didaktische Zwecke ist vor allem an Hochschulen weit verbreitet. Es gibt eine Reihe von Erprobungen in Schulen.

Die in [CW04] erstmals in dieser Breite publizierte *Verwendung von Scheme zur Einführung einer breiten Palette informatischer Grundkonzepte* steht nicht in Konkurrenz zu didaktisch spezialisierten *Modellwelten*, wie etwa der turtle geometry, BlueJ, PAGE, Karel, DrPython, KARA, Hamster u. v. m.

Nachfolgend werden die Schwerpunkte und einige ausgewählte Inhalte dieses Kurses vorgestellt und im Hinblick auf deren didaktisch-methodische Umsetzung reflektiert. Beispiele und weiterführende Übungsaufgaben zeigen, dass der Zugang über die funktionsorientierte Programmierung zum Studium fundamentaler Wirk- und Funktionsprinzipien komplexerer Informatiksysteme, wie beispielsweise der Client-Server-Architektur, ausgeweitet werden kann.

2 Fachdidaktische Konzeption

Zur Repräsentation funktionsorientierter Programmierung verwenden wir Scheme. Diese Sprache ist vorrangig für Lehrzwecke entwickelt worden und gestattet die Formulierung semantisch anspruchsvoller Ausdrücke bei äußerst knapper Syntax. Dies versetzt uns in die Lage, leistungsstarke Programmierkonzepte, wie die *Rekursion*, *Funktionen höherer Ordnung* und *verzögerte Evaluation* im Unterricht zu thematisieren. Diese sind in anderen Programmiersystemen nur ansatzweise vorhanden, erfordern starke Zugeständnisse an den Compiler und erfüllen eine unbefriedigende Lernkurve.

Scheme liegt in Gestalt von DrScheme als *kostenlose didaktische Entwicklungsumgebung* für die Betriebssysteme Windows und Linux vor. Damit ist DrScheme eines der ganz wenigen Programmiersysteme, die einen didaktischen Zuschnitt haben und permanent gepflegt und erweitert werden. DrScheme ist intuitiv zu bedienen und ermöglicht interaktive Entwurfs- und modulare Testzyklen.

In Tab. 1 sind die Schwerpunkte und Lehrplaninhalte zusammengestellt, mit denen wir uns hier besonders beschäftigt haben. Man kann erkennen, dass der betrachtete Kurs nicht verengt auf einen Lehrgang „funktionales Programmieren“ angelegt wurde.

Vielmehr geht es um grundlegender Datenstrukturen und Wirkprinzipien, die basierend auf dem Paradigma der funktionsorientierten Programmierung vermittelt werden. Dementsprechend wird der Schwerpunkt auf die *Entwicklung von Mentaltechniken*, wie das Abstrahieren und Generalisieren sowie das Beschreiben von Problemlösungen, gelegt.

Diese sehr anspruchsvolle Zielsetzung erfordert *motivierende Aufgaben* mit *praxisrelevantem Sachkontext*. Problemstellungen dieser Art finden sich dann auch in den Projektarbeiten wieder, s. Tab. 1.

LB	Thema	Schwerpunkte	Inhalte	Stunden
1	Einführung in die funktionsorientierte Programmierung	Read-Eval-Print-Loop (REPL) Datentypen - Zahlen - Zeichen / Zeichenketten - Wahrheitswerte - Datenstruktur Liste Bedingte Ausdrücke Prozeduren - benannt / unbenannt Rekursionen - echte Rekursionen - endständige Rekursionen - Mehrfachrekursionen	Kommunikation im Direktmodus einstellige / mehrstellige Prozeduren: - Summe, Fakultät, Fibonacci-Zahlen Prozeduren mit Listen - Minimum - Loeschen, Superloeschen - Element?, Superelement?	20
2	Prozeduren höherer Ordnung	Prozedur höherer Ordnung Anwendung von Prozeduren auf Listen komplexe Anwendungen	Numerische Ableitung beliebiger Funktionen Wertetabellen Nullstellensuche	10
3	verzögerte Evaluation	Evaluationstechniken Streams	Projektarbeit: Zahlenfolgen - rekursive, explizite Bildungsvorschrift - Grenzwerte von Zahlenfolgen - Partialsummenfolgen	10
4	λ - Kalkül	theoretische Grundlagen α-Konvention, β-Reduktion, η-Vereinfachung Symbolverarbeitung	einfache Anwendungen der Rechenregeln Projektarbeit: Symbolisches u. numerisches Differenzieren	10
5	Ausgewählte Algorithmen und ihre Effizienz	Intuitiver Algorithmusbegriff - Sequentielle / binäre Suche - Minsort / Quicksort Aufwandsbetrachtungen: - Zeitkomplexität und O-Notation Grenzen der Berechenbarkeit komplexe Anwendungen aus-gewählter Algorithmen in einem Mini-Datenbanksystem	Implementation ausgewählter Algorithmen empirische Effizienzuntersuchungen Projektarbeit: Mini-Datenbanksystem - Ein- / Ausgabe / Speichern von Datensätzen - Löschen von Datensätzen - Such- und Sortierfunktionen	15

Tabelle 1: Schwerpunkte und Inhalte

3 Zur Einführung der funktionsorientierten Programmierung

Die erste Kommunikation mit Scheme führt die Schüler zur *Präfixnotation*. Diese ist zwar zunächst gewöhnungsbedürftig, bereitet jedoch erfahrungsgemäß kaum Schwierigkeiten. Denn sehr häufig benutzen wir in der Mathematik bei der verbalen Beschreibung von Termstrukturen eine „Präfixsprechweise“, z. B.: „Der Quotient aus der Summe der Zahlen 3 und 5 und der Differenz der Zahlen 6 und 2“. Scheme folgt exakt dieser Deskription: **`(/ (+ 3 5) (- 6 2))`**. Die Scheme-Repräsentation wird also direkt durch *Versprachlichung* mathematischer Ausdrücke gewonnen. Durch die Nutzung *generischer Operatoren* entfallen die Grenzen typischer Binäroperationen: „Die Summe aller natürlichen Zahlen von 1 bis 10“ lautet in Scheme: **`(+ 1 2 3 4 5 6 7 8 9 10)`**.

Mit diesen einfachen Beispielen wird bereits die Bedeutung des *Beschreibungsaspekts* bei funktionsorientierter Programmierung andeutet: Wir abstrahieren vom konkreten Berechnungsprozess durch *Beschreibung des Resultat*, also des ‚Was’ und delegieren das ‚Wie’ an die Maschine, s. auch in [CW94].

Zwischen benutzerdefinierten und eingebauten Prozeduren besteht kein Gebrauchsunterschied. Hingegen ist die Unterscheidung „Term vs. Funktion/Prozedur“ fundamental. An dieser Stelle setzt begriffliches Arbeiten an, wobei die funktionsorientierte Programmierung nicht Selbstzweck ist, sondern Vermittler i. S. eines *didaktischen Transportmediums*. Die Bezeichnung **`lambda`** wird den Schülern kurz begründet.

```
(define volumen
  (lambda (radius hoehe)
    (* pi (sqr radius) hoehe)))

> (volumen 2.5 7.0)
137.44467859455347
```

Ein besonders leistungsfähiges Mittel zur Beschreibung von Resultaten ist die Rekursion. Damit wird von einem meist komplexen Berechnungsprozess abstrahiert. Dieser wird nach entsprechendem Aufruf vom Computer ausgeführt. Im Kopf des Programmierers darf er keinerlei Rolle spielen, wenn die Lösung erfolgreich entwickelt werden soll. *Das Denken in der Kontrolle ist schädlich für den Entwurf rekursiver Prozeduren!*

Den Schülern bereitet diese Denkungsart anfangs große Schwierigkeiten. Hier lässt sich nicht auf Alltagserfahrung zurückgreifen, denn rekursiv beschriebene Prozesse gibt es dort kaum. Deshalb sind intensive Übungen notwendig. Selbstähnliche geometrische Figuren helfen. Gründe, die einem grafischen Zugang den Vorzug geben, kennen wir bereits von LOGO’s turtle geometry. Allerdings eigenen sich auch Inhalte aus der Mathematik, wie (explizit und) rekursiv definierte Zahlenfolgen. Die Aufgabe, aus einer rekursiven Beschreibung einen iterativen Prozess abzuleiten, wird dem Programmiersystem übertragen.

Rekursives Denken kann trainiert werden! Die folgenden beiden Beispiele gehen aus Anregungen von Schülern hervor.

Beispiel 1: Minimumsuche in einer Zahlenliste

Beschreibung: *Das Minimum einer einelementigen Liste ist das Element selbst, anderenfalls bestimme das Minimum der Restliste (= Liste ohne erstes Element). Ist dieses kleiner als das erste Listenelement, so gib es als Minimum der gesamten Liste zurück, anderenfalls ist das erste Listenelement das Resultat.*

Implementation:
```
(define minimum
  (lambda (ls)
    (if (null? (cdr ls))
        (car ls)
        (let ([min (minimum (cdr ls))])
          (if (< min (car ls))
              min
              (car ls))))))

> (minimum '(23 43 4 5 6 7 898 7 65 43 2 1 5 5 56))
1
```

Beispiel 2: Schnapszahlenprüfer (keine Schnapsidee!)

Vorüberlegung: Eine mehrstellige natürliche Zahl heißt Schnapszahl, wenn alle ihre Ziffern untereinander gleich sind. Wir wollen diese Definition mit einem numerischen Algorithmus implementieren. Die leistungsstarke Ganzzahlarithmetik von DrScheme macht die Umsetzung besonders attraktiv.

Beschreibung: *Eine einstellige Zahl ist keine Schnapszahl. Eine zweistellige Zahl ist eine Schnapszahl, wenn beide Ziffern gleich sind. Eine mehrstellige Zahl ist eine Schnapszahl, wenn ihre letzten beiden Ziffern gleich sind und die Zahl ohne der letzten Ziffer eine Schnapszahl ist.*

Implementation:
```
(define schnapszahl?
  (lambda (n)
    (cond
      [(< n 10) #f]
      [(= (quotient n 10) (modulo n 10)) #t]
      [else
        (and (= (modulo n 10) (modulo (quotient n 10) 10))
             (schnapszahl? (quotient n 10)))])))

> (schnapszahl? 55555555555555555555555555555555)
#t
> (schnapszahl? 55555555555555555555655555555555)
#f
```

In Abgrenzung zu anders lautenden, auch aktuellen, aber keineswegs neuen didaktischen Diskussionen sind mathematische Einstiegsaufgaben hier keinesfalls kontraproduktiv. Mit der Auswahl einer Programmiersprache wird das dominierende Paradigma festgelegt. Das Paradigma bestimmt das Modellierungskonzept.

Im Falle der Objekt-orientierten Modellierung (OOA/OOM), die die aktuelle Programmierwelt prägen, stellt dies eine didaktische Herausforderung besonderer Komplexität dar: „OOM ist für Anfänger keineswegs intuitiv und einfach. …“ [SS04, S. 196]. Dagegen erweist sich dieser Modellierungsaspekt für funktional-applikative Sprachen durch deren *natürliche Nähe zur Mathematik* als wesentlich „anspruchsloser“. Insofern können funktional-applikative Sprachen sehr wohl Träger informatischer Konzepte sein, womit wir älteren Thesen (1996), wie „Funktionale Sprachen sind für die Schule ungeeignet, sie gehören auf die Universität.“ [RB96, S. 240] strikt entgegentreten.

Eine Folgerung aus diesen Überlegung ist die *Wahl mathematischer Anwendungen als Einstiegsaufgaben*. Obwohl wir mindestens seit 1980 von S. Papert wissen, dass grafisch repräsentierbare Sachkontexte zum Einstieg ins Programmieren algebraischen vorzuziehen sind, folgen wir der oben beschriebenen Modellierungscharakteristik und wählen deshalb Aufgabenstellungen aus der Mathematik. Außerdem werden nicht Grundschulkinder oder noch Jüngere angesprochen, sondern Schülerinnen und Schüler der gymnasialen Oberstufe. An dieser Mathematik-Anfangslastigkeit, die schon vor 10 Jahren in [CW94] bewusst praktiziert wurde, halten wir auf der Basis von mehr als 20-jähriger Lehrerfahrung mit funktional-applikativen Sprachen fest.

Für die Ausarbeitung von Themen, die sich mit der Effizienz von Algorithmen beschäftigen, bietet [CW03] eine Orientierung. Interessierte Leser finden hierin weitere Beispiele rekursiver Prozeduren, wie sie für typische algorithmische Entwurfsmuster durch *Prozessabstraktion* gewonnen werden können.

4 Ausgewählte Inhalte und fachdidaktisch-methodische Analysen

4.1 Raten einer Zahl – ein Einstiegsbeispiel für Effizienzuntersuchungen

Mit der Implementation des Ratens einer (vom Computer „erdachten“) Zahl ergibt sich sofort die Frage nach einem effizienten Algorithmus. Oft wird von den Schülern das Intervallhalbierungsverfahren als einfachen Vertreter der *Teile-und-Herrsche-Algorithmen* vorgeschlagen.

```
(define computerraten-anzahl
  (lambda (zahl max)
    (letrec
        ([suche
          (lambda (li z re)
            (let ([mi (quotient (+ li re) 2)])
              (cond
                [(= z mi) 1]
                [(< z mi) (+ 1 (suche li z mi))]
                [else
                 (+ 1 (suche mi z re))])))])
      (suche 1 zahl (+ max 1)))))

> (computerraten-anzahl (+ 1 (random 1000000)) 1000000)
19
```

Erwartungsgemäß liefert die Verwendung von Zufallszahlen recht unterschiedliche Ergebnisse, so dass sich nun empirische Untersuchungen anschließen müssen. Die Auswertung der entsprechenden Versuchsreihen kann dabei völlig offen gehalten werden. Denkbar sind hinreichend viele „Versuchswiederholungen“, die mit Standardwerkzeugen (Excel, gnuplot) ausgewertet werden. Zu Ausweitungen führen Prozeduren, wie beispielsweise **`(computerraten-wdhlg`** max wdhlg**`)`** und **`(grafik min max schritt wdhlg)`**. Der Aufruf **`(grafik 200 10000 200 1000)`** liefert das in Abb. 1 dargestellte qualitative Schaubild mit der Turtlegrafik von DrScheme.

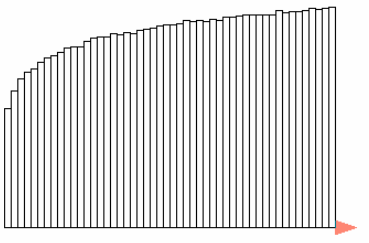

Abbildung 1: Schaubild zur Effizienzuntersuchung beim Raten einer Zahl

4.2 Dokumentbeschreibungssprachen und Generierung statischer Web-Seiten

Das DrScheme-System hält auch Module bereit, die es ermöglichen, einfache Modelle für die Kommunikation in Netzwerken zu entwickeln und zu erproben. Ohne in die bekannte „Technikfalle“ zu gehen, werden wesentliche Wirkprinzipien in vertrauter Arbeitsumgebung herausgearbeitet und projektbezogen angewandt.

Zwischen HTML/XML-Sprachen und Scheme gibt es eine enge strukturelle Verwandtschaft. Auf dieser Basis können bestimmte Scheme-Ausdrücke – quasi in natürlicher Weise – *statische* HTML-Dokumente *generieren*. Das kann den Autor stark entlasten.

```
(define wertetabelle
  (lambda (von bis step f)
    (letrec
      ([zeile
        (lambda (z)
          (if (> z bis)
            ()
            (cons
              `(tr (td ,(number->string z))
                (td ,(number->string (f z))))
              (zeile (+ z step)))))])
      (cons
        '(tr (th "x") (th "f(x)"))
        (zeile von)))))
```

```
(define website
  <>
<html>
<head>
<title>Wertetabelle</title>
</head>
<body bgcolor="white">
<h3>Fakult&auml;tsfunktion</h3>
<table border = "1">
  ,@
  (wertetabelle 0 10 1 fak)
</table>
</body>
</html>
)
```

Derart erzeugte Dokumente müssen vor der Betrachtung mit einem Webbrowser gespeichert werden.

Die Scheme-Elemente im HTML-Dokument werden evaluiert und an der vorgesehenen Stelle eingefügt. Im Beispiel wird die durch `(wertetabelle 0 10 1 fak)` *generierte* Tabelle (in HTML-Syntax) integriert. Die davon ausgehende Motivation für die Verwendung von „Produkt-Beschreibungen“ basiert darauf, dass Menschen nur widerwillig bereit sind, aufwändige und fehleranfällige Routinetätigkeiten, wie das Abtippen üppiger Tabellen mit separat berechneten Inhalten, auszuführen. Der Computer soll dies für uns auf der Basis einer geeigneten Beschreibung unseres Wunschobjekts erledigen.

4.3 Dynamische Web-Seiten – Client-Server-Architekur

Zur Kommunikation in Netzwerken stellt DrScheme einen eigenen Webserver bereit. Damit wird es möglich, *dynamische* Webseiten *serverseitig* zu *generieren*, um bestimmte Dienstleistungen über ein Web-Interface anzubieten. Abb. 2 zeigt einen „Funktionsanalysator“, der als Projekt unter Rückgriff auf Vorarbeiten zum Schwerpunkt „Prozeduren höherer Ordnung“ (s. Tab. 1) entwickelt wurde.

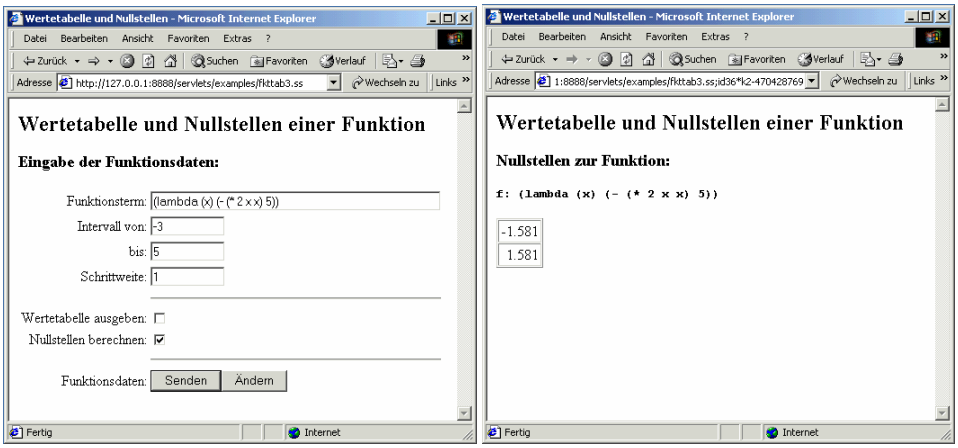

Abbildung 2: Nutzerformular des Clients / Antwortseite des Servers

Server und Client sollten auf getrennten Rechnern laufen, um den Schülern die Client-Server-Kommunikation erlebbar zu machen. Dann ist es nur noch ein kleiner Denkschritt zu weltweit verteilten Systemen. Dennoch sollte der Hinweis nicht fehlen, dass Client-Programm und Server-Programm auch auf ein und demselben PC laufen könnten (so wie es im Entwicklungsablauf zeitlich vorher auch stattfand). Hieraus erwächst die so wichtige Erkenntnis, dass Client und Server softwaremäßig bestimmte Begriffe sind. Redensarten, wie „Druckerserver“ und „Fileserver“, verwischen dieses Bild und assoziieren Hardwaregebundenheit.

4.4 Web-Datenbanken

Eine weitere Ausbaustufe stellt die Realisierung einer kleinen Webdatenbank dar.

Hier wird der bereits beschriebenen Client-Server-Kommunikation eine Datenbankkomponente hinzugefügt. Abb. 3 zeigt die Grundstruktur.

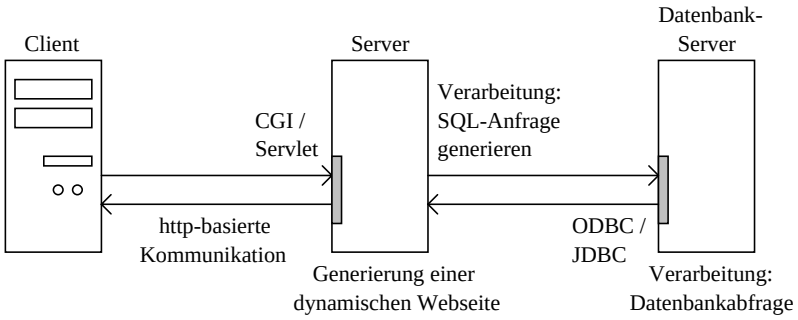

Abbildung 3: Client-Server-Kommunikation bei einer Webdatenbank

In Abb. 4 ist ein Kommunikationsbeispiel festgehalten worden.

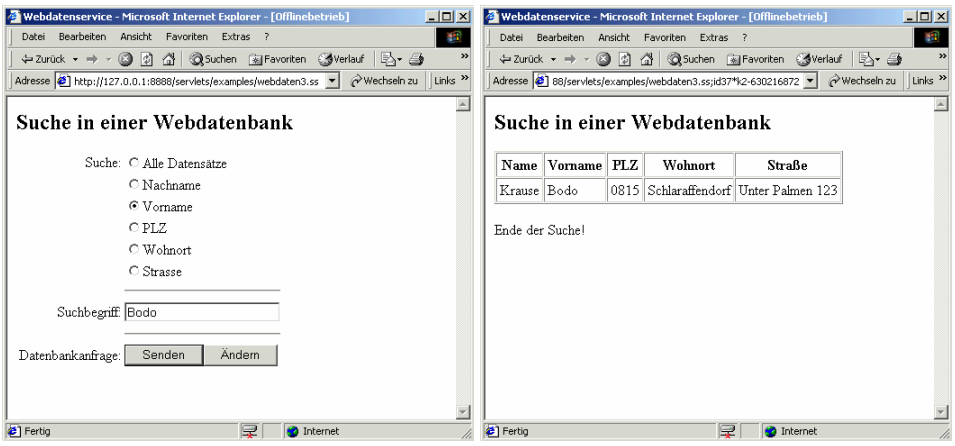

Abbildung 4: Nutzerformular und Antwortseite zum Webdatenservice

Serverseitig arbeitet Scheme als Interpreter für die SQL-Kommandos, deren Generierung im Interaktionsfenster des Scheme-Webservers mitverfolgt werden kann Die Zusammenarbeit mit dem bekannten relationalen Datenbankmanagementsystem MySQL via ODBC gelingt problemlos, s. [CW04]. Mit ein und derselben Umgebung werden damit Datenbank- und Netzwerkanwendungen für Schüler erleb- und nachvollziehbar.

5 Bisherige Erfahrungen

Die bisherige zweijährige Unterrichtspraxis lässt folgende Schlussfolgerungen zu:

- *Abstrahierendes Beschreiben* stellt hohe Anforderungen an die Schüler, sodass sich fachliche Schwächen nicht mit „schönen Programmoberflächen“ kaschierbar sind.

- Zunehmende Polarisierung der Schülerleistungen, verstärkte Leistungs- und Niveauunterschiede drücken sich auch in den Benotungen aus.
- Die überschaubare, einfache Syntax und die intuitive Bedienung von DrScheme setzen den Fokus stärker auf die Problemlösung. Auch das (gezielte) „Probieren durch Interagieren“ kann in schwierigen Phasen der Problemlösung weiterhelfen.
- Leistungsfähige Programmierkonzepte (verzögerte Evaluation / Prozeduren höherer Ordnung) gewährleisten einen ausgeprägten Praxisbezug.
- Die Software steht kostenlos zur Verfügung, was eine außerschulische Nutzung ermöglicht. Um die Unterrichtsziele zu erreichen, sind Fortführungen notwendig.
- Seiteneffektfreie funktionsorientierte Programmierung sowie die im Allg. zustandslosen Prozeduren gestatten nicht nur eine scharfe Modularisierung, sondern eröffnen auch verschiedene Möglichkeiten eines differenzierten Informatik-Unterrichts.
- Der didaktische Zuschnitt von DrScheme ermöglicht zahlreiche Bezüge zu den Wirkprinzipien verschiedener Informatiksysteme, die modellhaft nachgebildet werden können. Die betrachteten Zusammenhänge werden dadurch besser verstanden.
- Verschiedene Vertiefungsrichtungen (Datenbanksystem, Grafik) sowie unterschiedliche Programmierparadigmen (z. B. imperative-, objektorientierte-, logikbasierte-, parallele- und ereignisgesteuerte Programmierung) können ohne Wechsel der Programmierumgebung exploriert werden.

6 Fazit

Die Vermittlung von Grundkonzepten und Prinzipien der Informatik via funktionsorientierter Programmierung mit Scheme ist unter Verwendung der beschriebenen Mittel ein didaktisch tragfähiger Weg. Dies wurde bereits für ganz besonders abstrakte Inhalte der theoretischen Informatik nachgewiesen, vgl. [CW98].

Literaturverzeichnis

[LP04] http://www.sn.schule.de/~ci/download/lp_gy_informatik.pdf: Lehrplan für Informatik an Gymnasien im Freistaat Sachsen, 2004.

[EW02] Eckwerte zur informatischen Bildung an sächsischen Schulen, Comenius-Institut, 2002.

[CW04] Wagenknecht, Christian: Programmierparadigmen – eine Einführung auf der Grundlage von Scheme. Wiesbaden: Teubner-Verlag, 2004.

[CL04] Landfried, Carsten: Entwicklung und Erprobung lehrplangerechter Unterrichtsmaterialien für den Informatik-Unterricht in der Sekundarstufe II, Diplomarbeit 2004.

[CW94] Wagenknecht, Christian: Rekursion. Ein didaktischer Zugang mit Funktionen. Bonn: Dümmler, 1994.

[SS04] Schubert, Sigrid; Schill, Andreas: Didaktik der Informatik. Heidelberg, Berlin: Spektrum Akademischer Verlag, 2004.

[RB96] Didaktik der Informatik. Stuttgart: Klett-Verlag, 1996.

[CW03] Wagenknecht, Christian: Algorithmen und Komplexität. München: Hanser-Verlag, 2003.

[CW98] Wagenknecht, Christian; Friedman, Daniel P.: Teaching Nondeterministic and Universal Automata Using Scheme. In: Computer Science Education, Swets & Zeitlinger, vol. 8, no. 3, pp. 197-227, 1998.