



Institut für Verteilte Systeme
Modellbildung — Bewertung — Industrielle Anwendungen

FH Braunschweig/Wolfenbüttel
Hochschule für Technik, Sozial- und Wirtschaftswissenschaften

Aspekte der Auswahl von Programmiersprachen für die Lehre. Ein Diskussionsbeitrag

Argumente für Oberon als erste Sprache im Informatik-Studium

Technischer Bericht 08A/97

Roland Rüdiger¹

¹Postanschrift: Prof. Dr. Roland Rüdiger, Institut für Verteilte Systeme, Fachbereich Informatik, FH Braunschweig/Wolfenbüttel, Salzdahlumer Str. 46/48, D-38302 Wolfenbüttel

Zusammenfassung

In diesem Bericht werden die Argumente zusammengestellt, die zur Einführung von Oberon als erster Programmiersprache in den Studiengängen *Praktische* und *Technische Informatik* im Fachbereich Informatik der FH Braunschweig/Wolfenbüttel ab dem WS 94/95 geführt haben. Nach einer allgemeinen Einleitung werden die Argumente ausführlich kommentiert und Vor- und Nachteile gegenübergestellt. Es wird ferner über erste Erfahrungen mit Oberon in der Lehre berichtet. Im letzten Abschnitt wird als Fazit zusammengefaßt, daß Oberon wegen seiner extrem knappen, auf das Wesentliche reduzierten Sprachmittel sehr gut geeignet ist, in der zur Verfügung stehenden Zeit die wesentlichen Konzepte der Objektorientierten Programmierung bis zum Ende des 2. Studiensemesters zu vermitteln. Es wird darüber hinaus vorgeschlagen, bereits im 2. Semester rudimentär eine weitere Sprache einzuführen, um einer zu starken gedanklichen Assoziation zwischen allgemeinen Informatikkonzepten und den Konstrukten einer einzelnen, speziellen Programmiersprache zu begegnen.

Inhaltsverzeichnis

1	Einleitung	1
2	Allgemeiner Rahmen	1
3	Auswahl von Programmiersprachen für die Lehre	3
3.1	Allgemeine Aspekte	3
3.2	Argumente für Oberon	4
4	Erste Erfahrungen mit Oberon	6
5	Fazit und Perspektiven	7
A	Softwarebezogene Lehrveranstaltungen im Fb Informatik	9

1 Einleitung

Nach der Errichtung des Fachbereichs Informatik an der FH Braunschweig/Wolfenbüttel im Jahre 1993 wurde zum Wintersemester 1994/95 ein neuer Studiengang *Praktische Informatik* ins Leben gerufen und der bereits bestehende Studiengang *Technische Informatik* in neuer Form fortgeführt. Eines der Ziele, welches mit der Einführung des Studiengangs Praktische Informatik erreicht werden sollte, bestand darin, die Lehr- und Forschungsaktivitäten in der softwareorientierten Informatik zu intensivieren, da ein besonderer Bedarf an Hochschulabsolventen mit diesem Ausbildungsschwerpunkt besteht. Die Zahlen der Studienplatzbewerber in den vergangenen Jahren zeigen, daß dieses Konzept richtig war.

Zu den formalen Voraussetzungen, die für einen neuen Studiengang zu schaffen sind, muß natürlich hauptsächlich und in erster Linie die detaillierte Ausgestaltung der Lehrinhalte treten. Die Lehrveranstaltungen müssen aufeinander abgestimmt werden; es müssen Erweiterungen in den Fächern vorgesehen werden, deren Stundenvolumen erhöht worden ist, z.B. in den einführenden Lehrveranstaltungen *Informatik I, II und III (Theoretische Informatik)*. Unvermeidlich bedeutet das auch eine Kürzung in anderen Fächern, in denen die Lehrinhalte dementsprechend zu straffen und gleichzeitig den besonderen und spezifischen Bedürfnissen der Informatik anzupassen sind.

Einer der wichtigsten Aspekte in einer softwareorientierten Informatikausbildung ist die Wahl der ersten und der weiteren Programmiersprachen; insbesondere die erste Sprache wirkt prägend das gesamte Studium hindurch und darüber hinaus. Auch wenn das erklärte Ziel ist, primär Konzepte zu vermitteln, so wirkt doch die erlernte Programmiersprache auf das Denken zurück (vgl. auch [CS86]). Es wurde daher lange und sorgfältig diskutiert, welche Sprache als erste unter heutigen Aspekten geeignet ist. In diesem Bericht werden die Argumente zusammengestellt, die letztendlich zur Wahl von Oberon¹ als der ersten Sprache geführt haben.

Die Argumente können nicht losgelöst gesehen werden von sonstigen Überlegungen für die inhaltliche Ausgestaltung der Studiengänge, insbesondere von den Möglichkeiten, weitere Sprachen einzuführen. Deshalb werden auch einige Überlegungen dargestellt, die über die Einführung einer Pro-

grammiersprache hinausgehen. Es ist aber nicht annähernd das Ziel, mit diesem Bericht ein Gesamtkonzept für die Lehre in Informatik vorzulegen. Ein vergleichbarer Artikel, in dem für die Einführung von C und C++ plädoyiert wird, allerdings unter anderen Randbedingungen, findet sich in [LWW96]. Ein wesentlich allgemeineres Rahmenkonzept für die Lehre in Informatik an Fachhochschulen findet man in [Fre96]. Der vorliegende Bericht macht gewissermaßen einen Vorschlag zur detaillierten Ausgestaltung eines einzelnen Punktes aus [Fre96], nämlich der Wahl von Programmiersprachen, wobei z.T. auch spezielle Rahmenbedingungen des Fachbereichs Informatik der FH Braunschweig/Wolfenbüttel eingehen.

Die hier präsentierten Vorstellungen sind ganz entscheidend durch zahlreiche Diskussionen mit vielen Kollegen aus dem Fachbereich Informatik geprägt worden. Trotzdem bleiben sicherlich viele der Gedanken und Ideen, die hier dargestellt werden, in mancher Hinsicht subjektiv.

Der vorliegende Bericht ist primär geschrieben für diejenigen Studierenden, die sich darüber informieren möchten, mit welchen Argumenten und Überlegungen Lehrinhalte festgelegt worden sind und wie die weitere Fortentwicklung aussehen könnte, aber auch für die weitere fachbereichsinterne Diskussion. Für externe Leser könnten die in Abschnitt 4 beschriebenen Erfahrungen mit Oberon von Interesse sein.

2 Allgemeiner Rahmen

Jedes technisch orientierte Fach ist heute in einer sehr ausgeprägten Weise mit einer ganzen Reihe von Problemen konfrontiert. In der Informatik treten diese Probleme in einer besonders zugespitzten Weise auf. Dazu gehört insbesondere, daß

- die Zyklen der Innovation von Hard- und Software äußerst kurz sind,
- die Vielfalt an Softwaresystemen extrem groß ist, so daß eine Auswahl unter dem Aspekt hoher Effizienz für die Ausbildung sehr schwierig ist,
- die Anforderungen an die Fähigkeit zur Abstraktion rapide steigen und daß
- zunehmend nicht-technische Anforderungen gestellt werden, wie z.B. die Fähigkeit und

¹Im folgenden wird von *Oberon* gesprochen, wenn es auf eine Unterscheidung der beiden Sprachvarianten Oberon-1 und Oberon-2 nicht ankommt. Mit *Modula* ist im folgenden *Modula-2* [Wir85] gemeint.

Bereitschaft zur Zusammenarbeit, etwa bei der Erstellung großer Softwaresysteme.

Die kurzen Innovationszyklen bedeuten, daß jegliches konkrete Wissen schnell veraltet, mit der Folge, daß Detailwissen nicht ganz selten seine Aktualität mit Ende des Studiums eingebüßt hat. So muß man damit rechnen, daß z.B. Kenntnisse zu einer speziellen Compiler-Version oder Entwicklungsumgebung, die am Anfang des Studiums erworben wurden, am Ende des Studiums – im günstigen Fall(!) – bereits nicht mehr aktuell sind oder – im ungünstigen und leider inzwischen schon realistischeren Fall – am Ende des Semesters, in dem sie gelehrt wurden, eine offensichtlich absurde Situation; denn die Ausbildung hätte dann – in technischer Nomenklatur gesprochen – den Wirkungsgrad Null. Wie extrem die Situation hier wirklich ist, sieht man an der Tatsache, daß die Ausbildung, die in der Regel ca. 4 Jahre dauert, die Basis schaffen muß für eine folgende Berufstätigkeit von ca. 40 Jahren bei einer typischen Zeitskala für Versionswechsel von Softwareprodukten einer Größenordnung von ca. 1-2 Jahren.

Das Argument, die Hochschule sollte nur den *Einstieg* ins Berufsleben ermöglichen, alles weitere ergäbe sich von selbst, ist nicht überzeugend. Man denke hier an die Systemprogrammierer, die als gesuchte (und hochbezahlte) Kenner der IBM-370-Welt und hervorragende 370-Assembler-Spezialisten vor wenigen Jahren fast „über Nacht“ ihre Basis in der Berufswelt verloren, weil ihnen der Umstieg in die neue Welt der PCs und Netze nicht zugetraut wurde.

Die Vielfalt an Softwaresystemen und die raschen Versionswechsel – für jedermann ersichtlich in einer Kategorie wie z.B. der Textverarbeitung – erzeugt den permanenten Zwang, von Implementierungsdetails zu abstrahieren, weil diese zu schnell veralten. Andererseits sind reale Systeme nicht ohne konkretes Detailwissen zu handhaben. Es ist nicht immer einfach, eine sinnvolle Trennlinie zwischen Wichtigem und Unwichtigem zu finden und diese den Studierenden dann auch überzeugend zu vermitteln. Absolut nicht vernachlässigbar sind die finanziellen Probleme, die sich ergeben, wenn die Lehre stets anhand neuer Produkte bzw. Versionen erfolgen soll.

Die Anforderungen an die Fähigkeit, abstrakte Begriffsbildungen aufzunehmen und zu verstehen, werden nicht nur in theoretisch orientierten Ar-

beiten deutlich, sondern auch in modernen Normtexten. Man betrachte unter diesem Aspekt z.B. die DIN-Norm 66 256 für Pascal, oder die für C (ANSI X3.159-1989) oder Modula (ISO Modula, IS 10 154, April 1993). Auch bei rein technischen Normen, die nicht einmal Programmiersprachen direkt betreffen, ist das in keiner Weise anders, z.B. bei der Norm für die Schicht 7 des Feldbusses PROFIBUS (DIN-Norm 19 245, Teil 2). Hier werden z.B. gewisse Kenntnisse der Automatentheorie sowie der Syntaxspezifikationssprache ASN.1 der ISO von 1987 vorausgesetzt, obwohl es letztlich „nur“ um die industrielle Fertigung geht, also durchaus nicht um ein abgehobenes theoretisches Gebiet.

Ein ähnliches Bild bietet sich, wenn man moderne Entwurfstechniken betrachtet: die verschiedenen Ebenen, die hier üblicherweise als wesentlich genannt werden, sind Funktionale Abstraktion, Datenabstraktion, Abstrakte Datentypen sowie Objektorientierung (s. z.B. [Bal90], [Mö93]). Hierbei ist wichtig, daß diese Hierarchie von den Studierenden konzeptionell begriffen wird, also sprach- bzw. implementierungsunabhängig (vgl. dazu auch das Vorwort von N. Wirth in [Mö93, S. vi].).

Daß die Förderung der sozialen Kompetenz von Studierenden durch die Hochschulen nicht immer ausreichend ist wird von der Industrie gelegentlich beklagt. Natürlich bieten auch nicht alle Fächer entsprechende Möglichkeiten in gleicher Weise. Die Softwaretechnik scheint allerdings besonders dazu prädestiniert zu sein, hier Beiträge zu liefern, weil die Entwicklung großer Softwaresysteme nicht anders als durch koordinierte Zusammenarbeit vieler Personen möglich ist².

Ein weiteres Problem in einführenden Lehrveranstaltungen zur Informatik besteht darin, daß die Vorkenntnisse von Studierenden manchmal in einer sehr extremen Weise divergieren: es gibt gelegentlich in der gleichen Lehrveranstaltung Teilnehmer, die bereits Softwareentwicklung auf einer professionellen Basis betreiben neben anderen Teilnehmern, die kaum Erfahrung mit Rechnern haben oder vielleicht sogar überhaupt zum ersten Mal an einem Rechner sitzen. Allerdings trifft man auch nicht ganz selten auf die Situation, daß umfangreiche Vorkenntnisse auch hemmend sein können, eine Erfahrung, über die auch in [CS86] berichtet wird.

²Ein neueres Beispiel: in [SCG93] findet sich eine Notiz, daß das Softwarehaus PTT ein in Eiffel geschriebenes Programmpaket für den Marktforschungsbereich erstellt. Daran arbeiten 20 Programmierer mit einem geschätzten Aufwand von 100 Personenjahren.

3 Auswahl von Programmiersprachen für die Lehre

3.1 Allgemeine Aspekte

Natürlich kann und soll hier kein Patentrezept präsentiert werden, das alle oben genannten Probleme sozusagen im Handumdrehen löst – und möglicherweise die nicht genannten gleich noch dazu. Tatsächlich dürfte es eine solche Lösung auch kaum geben, da eine ganze Reihe konkurrierender Einflüsse auftreten, zwischen denen recht subtile Abwägungen vorzunehmen und Kompromisse zu schließen sind.

Wenn man Informatiker oder Praktiker aus dem Informatikbereich fragt, nach welchen Gesichtspunkten Programmiersprachen für die Lehre auszuwählen sind, so erhält man ein ganzes Spektrum verschiedenartiger Antworten. Einige typische Repräsentanten könnten etwa die folgenden sein:

- Die Ausbildung im Programmieren spielt überhaupt keine Rolle mehr; wir befinden uns heute in einer Ära der Sprachen der 4. Generation, die es jedem, auch dem nicht-gelernten Programmierer, gestatten, aufwendige und anspruchsvolle Programme zu entwickeln.
- Man wählt diejenige Sprache aus, die in der Industrie, also dem späteren Arbeitsplatz für die große Mehrzahl der Absolventen, am meisten verbreitet ist. Das bedeutet für die Studierenden bei Bewerbungen nach ihrem Studienabschluß einen außerordentlichen Vorteil. Es genügt, nur eine einzige Sprache zu lehren, die aber sehr gründlich.
- Man versucht, die Sprache so zu wählen, daß nach Möglichkeit langlebige Konzepte und Methoden vermittelt werden ohne Rücksicht auf die (derzeitige) Verbreitung der Sprache. Diese nimmt nur die Rolle eines untergeordneten Hilfsmittels ein.
- Man lehrt überhaupt nur oder überwiegend abstrakte Konzepte und die dann ohne Rechner, d.h., nur mit „Programmieren auf Papier“. Diese Konzepte sind naturgemäß weitaus langlebiger als alle konkreten Implementierungen.
- In einer Zeit sinkender Zahlen von Bewerbern um einen Studienplatz sollte die erste Sprache auch unter dem Aspekt ausgewählt

werden, „populär“ zu sein; das kann einen wichtigen Werbeeffect haben.

Keines dieser Argumente – im Zusammenhang mit anderen – sollte man als vollkommen falsch oder – für sich genommen – als vollkommen richtig ansehen. Es muß ein Kompromiß geschlossen werden zwischen einer ganzen Reihe von begrenzenden Einflußgrößen. Zunächst sollen hier die obigen Argumente kommentiert werden.

Die sogenannten Sprachen der 4. Generation werden in der Fachliteratur entweder nicht zur Kenntnis genommen oder sehr kritisch gesehen, s. dazu [Lud93]. Sie gelten als mehr oder weniger nützliche Produkte für die Spezialzwecke, für die sie ausgelegt sind. Der Anspruch, eine neue Generation von *universellen* Programmiersprachen zu bilden, ist wohl nicht gerechtfertigt. Man vergleiche die Situation bei der Verarbeitung (Formatierung) von Texten: ein Textverarbeitungsprogramm selbst neu zu schreiben, etwa in Pascal, C oder FORTRAN, wäre eine immens aufwendige Aufgabe: diese kann z.B. mit der Sprache(!) $\text{\TeX}$ (bzw. $\text{\LaTeX}$) wesentlich einfacher gelöst werden, denn sie wurde für diesen Zweck entwickelt. Trotzdem wurde noch nicht behauptet, $\text{\TeX}$ gehörte deswegen zu einer gänzlich neuen Generation von universellen Sprachen, die auch z.B. für die numerische Integration von partiellen Differentialgleichungssystemen bestens geeignet wären.

Das Argument, daß eine in der Industrie verbreitete Sprache gewählt werden sollte, scheint auf den ersten Blick sicherlich eine erhebliche Attraktivität zu besitzen. Das Argument wäre überzeugend, wenn es hier entweder eine große Einheitlichkeit gäbe oder wenn es einen (Groß-) Betrieb gäbe, der durchgehend nur eine Sprache verwendet und der praktisch alle Absolventen aufnimmt; das ist aber nicht der Fall. C ist zwar sehr verbreitet im industriellen Bereich, aber nicht in einem solchen ausschließlichen Ausmaß, welches es rechtfertigen würde, auf jegliche anderen Kenntnisse zu verzichten.

Die Konzentration auf das Lehren von Konzepten in den ersten Semestern ist der Weg, der hier propagiert wird mit den folgenden detaillierten Begründungen. Auch hier müssen allerdings die Grenzen aufgezeigt werden, innerhalb der dieses zu geschehen hat. Das Spektrum der Sprachen, die gelehrt werden, muß zeitlich ausgewogen sein: die Industrienähe nach heutigem Stand und Wissen muß in gleicher Weise zur Geltung kommen wie das von der praktischen Anwendung abstrahierte Lehren von Konzepten. Dabei ist es sicherlich rich-

tig, den Schwerpunkt auf eine Praxisorientierung verstärkt in die höheren Semester zu legen. Die oben genannten Zeitskalen machen deutlich, daß eine z.B. auf die Details einer speziellen Entwicklungsumgebung ausgelegte Ausbildung am Anfang des Studiums kaum sinnvoll sein kann.

Die ausschließliche Konzentration auf programmiersprachliche Konzepte scheint für Fachhochschulen kaum sinnvoll zu sein, für Universitäten heute allerdings wohl ebensowenig. Das Problem liegt darin, daß die vollständige Beschreibung der Wirkungsweise von Programmen in einer abstrahierten, etwa zu mathematischer Axiomatik analogen Weise, kaum möglich ist, jedenfalls nicht in einer für Anfänger akzeptablen Art. Das (berühmte) „Ausprobieren am Rechner“ hat zweifellos eine wichtige unverzichtbare didaktische Funktion; Probleme entstehen erst dann, wenn *nur noch* mit Ausprobieren programmiert wird.³

Der genannte Werbeeffect durch die Verwendung einer „populären“ Sprache ist natürlich nicht gänzlich von der Hand zu weisen. Aber die weit überwiegende Zahl aller Studierenden akzeptiert auch einen erheblichen Umfang an Lehre z.B. in Mathematik in den ersten Semestern, die wohl nur in Einzelfällen populär sein dürfte. Wenn geworben wird, so sollte wohl besser mit einem Gesamtkonzept aller Studieninhalte geworben werden, nicht mit dem Detail *Erste Programmiersprache*; auch hier muß deutlich gemacht werden, daß Programmiersprachen – bei aller Wichtigkeit – nicht der primäre Gegenstand der Lehre sind.

3.2 Argumente für Oberon

Die bis zum Sommersemester 1994 im ersten Studienabschnitt des Studiengangs *Technische Informatik* gelehrt Sprache war Pascal; verwendet wurde das Produkt Turbo-Pascal mit der bekannten Borland Entwicklungsumgebung. Zweifelloso war der vor einer Reihe von Jahren vollzogene Übergang von FORTRAN zu Pascal für die Softwaretechnik ein außerordentlich großer Fortschritt. Trotzdem gibt es gerade aus der Sicht der in den späteren Semestern nachfolgenden Ausbildung in Softwaretechnik gegenüber Pascal eine Reihe von Vorbehalten und Bedenken.

Diese Vorbehalte betreffen z.T. Pascal in der ursprünglichen Form, welches ja N. Wirth selbst weiterentwickelt hat zu Modula, z.T. spezielle Implementierungen wie z.B. Turbo-Pascal. Der Na-

me *Pascal* ist hier eigentlich nicht gerechtfertigt: Turbo-Pascal steht Modula näher als ISO-Pascal, was übrigens die gelegentlich geäußerte Aussage zurechtrückt, Modula hätte nie eine große Verbreitung gefunden. Um hier Mißverständnisse zu vermeiden: die Produkte Turbo/Borland-Pascal und nachfolgende Versionen sind nützliche Hilfsmittel, da ein großer Funktionsumfang zur Verfügung steht, mit dem man schnell und bequem Programme für diverse Zwecke entwickeln kann. Die Vorbehalte betreffen eher die Verwendung für Lehrzwecke mit dem Ziel, grundlegende Konzepte und Prinzipien zu vermitteln. Ein einziges Beispiel soll an dieser Stelle zur Verdeutlichung genügen. Bekanntlich ist **PROGRAM** das Startsymbol der Pascal-Grammatik. In ISO-Pascal folgt nach dem Programmnamen in Klammern eine Liste von Bezeichnern, die die Bindung herstellen zu externen Größen (in einer durch die Norm nicht festgelegten Weise). Diese Bindung wird in Turbo-Pascal anders festgelegt; es ist daher zulässig, den gesamten Programmkopf wegzulassen, so daß ein Turbo-Pascal-Programm z.B. mit dem Wortsymbol **USES** beginnen kann. Dem Anfänger wird dadurch suggeriert, daß die Sprachsyntax eher etwas vages ist: manche Dinge dürfen weggelassen werden, andere nicht; man muß sich das eben einprägen (oder noch einfacher: von Fall zu Fall am Rechner ausprobieren). Die Liste ließe sich fast beliebig verlängern. Daß hier allerdings auch eine andere Sicht vertreten werden kann und wird, zeigt die Informatik-Einführung von Gumm und Sommer ([GS94]), die als Sprache Borland-Pascal zugrundelegen.

Ebenso wie Pascal zu Modula, wurde Modula zu Oberon weiterentwickelt ([Rei91], [MW91], [RW92], [WG92], [Mö93]). Im folgenden sind nun Argumente zusammengestellt, die zur Wahl von Oberon als der einführenden Programmiersprache geführt haben. Wie bereits erwähnt, sind diese Argumente im Kontext sonstiger Lehrinhalte zu sehen.

- Oberon ist ein Mitglied der Algol/Pascal-Sprachfamilie. Pascal hat sich in den vergangenen zwei Jahrzehnten als eine für didaktische Zwecke hervorragend geeignete Sprache erwiesen ungeachtet der bekannten und oft genannten Defizite, vgl. [RW92]
- Es gibt gute Gründe, in dieser Sprachfamilie zu bleiben, da nicht nur Lehrbücher zur Soft-

³Wenn E. Dijkstra ausdrücklichen Wert auf die Feststellung legt, daß keines der Programme in einem seiner Bücher am Rechner ausprobiert („ausgetestet“) wurde, dann ist das sicherlich ein nachahmenswertes Vorbild für jeden Informatiker entsprechenden Ranges. Leider sind solche Informatiker aber nicht nur unter den Studierenden selten

waretechnik Pascal-artigen Pseudocode verwenden. Viele Sprachen lehnen sich an Pascal an; stellvertretend für zahlreiche Beispiele sei hier die Spezifikationssprache *Estelle* ([ISO89]) aus dem Bereich der formalen Beschreibungstechniken angeführt, die zur Spezifizierung von Protokollen für Kommunikationsnetze eingesetzt wird, einem Bereich, der gerade auch im industriellen Umfeld zunehmend wichtiger wird.

- Die Pascal-Sprachfamilie steht zu Recht in dem Ruf, daß der Übergang zu anderen Sprachen besonders einfach ist.
- Oberon ist eine extrem einfache Sprache. Sie enthält die wesentlichen Sprachelemente von Modula und damit auch die von Pascal. Eine Reihe von Sprachelementen, die nach dem Urteil von N. Wirth nicht unbedingt notwendig sind, da sie anderweitig nachgebildet werden können und der Typerweiterung im Wege stehen, sind weggelassen.
- Mit Oberon lernt man daher in gewisser Weise implizit automatisch Pascal und Modula mit. Insofern ist der Übergang zu dieser neueren Sprache konservativ und vorsichtig.
- Wegen der Einfachheit der Sprache sind die Compilationszeiten extrem kurz, eine nicht nur für Anfänger hervorragende Situation, da Tests ohne merkbare Leerlaufzeiten durchgeführt werden können. Man vergleiche hier etwa die Compilationszeiten von Eiffel/S in einer MS-DOS-Umgebung.
- Oberon erscheint als Sprache aus einem Guß ohne konzeptionelle Brüche. Sie unterscheidet sich in dieser Hinsicht von kommerziellen Pascal-Implementierungen, die Sprachkonstrukte aus Pascal, Modula und weiteren Sprachen mischen, wie oben bereits angedeutet.
- Wie in Modula wird in Oberon die modulare Programmierung unterstützt in einer gegenüber Modula noch weiter vereinfachten Weise. Es ergibt sich daher bereits sehr früh im Laufe der Ausbildung die Gelegenheit, die fundamentale Rolle des Begriffs *Schnittstelle* in einer prototypischen Form kennen zu lernen.

- Oberon ist eine hybride Sprache (wie C++ auch, anders als etwa Eiffel, Smalltalk oder Java), die auch die objektorientierte Programmierung (OOP) unterstützt, in der es aber keinen Zwang gibt, stets Klassen und Methoden im Sinne der OOP benutzen zu müssen. Daher läßt sich das Abstraktionsstufenmodell auf jeder Stufe im Detail mit Oberon-Sprachmitteln studieren.
- Die Einfachheit der Sprachkonstrukte, mit der eine Unterstützung der OOP erreicht wird, sucht ohne Zweifel ihresgleichen. Eine Folge ist, daß grundlegende Konzepte wie *Vererbung*, *Polymorphie* und *dynamisches Binden* leicht und schnell zu vermitteln sind.
- Die in Lehrveranstaltungen schwierige Situation, daß meistens einige der Studierenden bereits hervorragende Pascal-Kenntnisse mitbringen während andere keinerlei Programmiererfahrungen haben, wird durch die Einführung einer etwas „exotischen“ Sprache wie Oberon zumindestens entschärft. Allerdings gibt es auch vereinzelt bereits Studierende, die Oberon-Kenntnisse mitbringen.
- Das Oberon-System ist konzipiert als erweiterbares und anpaßbares System. Die typischen Fragen, nicht nur von Anfängern, „Kann Oberon das?“ oder „Geht das in Oberon auch?“ zeigen, daß das Verständnis des Konzepts der *Erweiterbarkeit*, wie es heute gesehen wird, noch nicht sehr verbreitet ist. Allerdings ist anzumerken, daß i.a. erhebliche Erfahrungen vonnöten sind, um Erweiterungen auch wirklich zu realisieren. Ein elementares Beispiel, welches aber dennoch am Anfang stets Probleme bereitet, bilden die Module *In* und *Out*. Es wird regelmäßig vermutet, die (sehr eingeschränkte) Funktionalität dieser Module wäre typisch für die Sprache(!) oder das System Oberon. Hier spielt offensichtlich die Vorerfahrung mit Pascal eine Rolle, in dem die Ein- und Ausgabeprozessen noch Teil der Sprache sind.
- Die Sprache Oberon hat sich bewährt in einem Projekt beträchtlicher Größe, nämlich in der Implementierung des Oberon-Systems selbst.⁴

⁴Wobei allerdings die beteiligten „Teilzeitprogrammierer“ ([WG92, S. 3]) wohl nicht sonderlich geeignet sind, den Durchschnittsmaßstab für Programmierer schlechthin festzulegen.

- Studierende der Informatik zeigen – wie auch die Lehrenden – in aller Regel eine große Begeisterung für neue technische Entwicklungen in der Informationstechnik. Eine ganze Reihe von Entwicklungen sollten jedoch auch kritisch gesehen werden. Das Oberon-Projekt ist aus einer derartigen kritischen Haltung gegenüber den heutigen aufgeblähten und extrem komplexen Systemen entstanden (s. dazu [Wir94]). Die Einführung in die Programmierung anhand von Oberon kann eine Gelegenheit sein, den Studierenden die kritische Haltung von N. Wirth als eine außerordentlich gewichtige und keinesfalls vereinzelte Meinung zu vermitteln.
- Und schließlich: Oberon ist kostenfrei erhältlich, nicht nur für Studierende mehr als nur ein angenehmer Seiteneffekt, es ist portiert auf eine große Anzahl von Plattformen; insbesondere gibt es eine Version, die unter Linux lauffähig ist.

Natürlich wurden auch solche Argumente betrachtet, die möglicherweise gegen die Verwendung von Oberon als der einführenden Sprache sprechen.

Der Haupteinwand ist natürlich die fehlende Verbreitung im industriellen Bereich. Ein Studiengang, der als „praxisorientiert“ konzipiert ist, sollte sich – so das Argument – auf Lehrinhalte beschränken, die auch im industriellen Umfeld sofort eingesetzt werden können. Dieses Argument träfe dann zu, wenn neben Oberon keine weitere Sprache gelehrt würde. Wie weiter unten ausgeführt, sind jedoch C und C++ die wesentlichen Sprachen im Hauptstudium und werden es auf absehbare Zeit auch bleiben. Darüber hinaus werden weitere Sprachen in einigen Wahlpflichtfächern verwendet.

Natürlich muß die Ausbildung generell so angelegt werden, daß den Studierenden gute Chancen auf dem Arbeitsmarkt eröffnet werden; dieser Aspekt ist jedoch in höheren Semestern wesentlich wichtiger als im Grundstudium. Einen guten Ausgleich zwischen der Vermittlung grundlegender Konzepte und der unmittelbaren Vorbereitung auf den Einstieg in eine praktische Berufstätigkeit erreicht man sicherlich, wenn man diese Schwerpunkte auf die beiden Studienabschnitte Grund- und Hauptstudium verteilt. Es ist zugegebenermaßen allerdings von Nachteil für diejenigen, die ihr Studium nicht regulär abschließen können.

Ob man die vollkommen neu konzipierte Handhabung des – verglichen mit herkömmlichen Fenstersystemen wie z.B. MS-Windows – gänzlich anders-

artigen Oberon-Systems als Nachteil bewerten soll, ist allerdings höchst fragwürdig; sie bedeutet vielmehr eine Schulung in Flexibilität, die für Informatiker in jedem Bereich, insbesondere auch in einem praktisch-technisch orientierten, vollkommen unverzichtbar ist.

4 Erste Erfahrungen mit Oberon

Die wesentliche positive Erfahrung mit Oberon als der einführenden Programmiersprache besteht darin, daß ohne Zeitprobleme bis zum Ende des 2. Semesters die grundlegenden Konzepte der OOP eingeführt werden können. Der Aufbau der Vorlesung folgt dem Abstraktionsstufenmodell; mit den einzelnen Ebenen dieses Modells können – nach einer allgemeinen einführenden Übersicht – entsprechende Sprachelemente eingeführt werden, also z.B. Prozeduren zusammen mit der prozeduralen Abstraktion. Es soll nun hier nicht behauptet werden, daß man mit einer anderen Sprache nicht ebenfalls diese Konzepte einführen könnte; es wird aber im Verlaufe eines Einführungskurses schnell deutlich, daß die konsequente Vereinfachung der Sprache gegenüber Pascal und Modula für die Lehre auch eine erhebliche Zeitersparnis bedeutet, die der Einführung neuerer Konzepte zugute kommt.

Von großem didaktischen Nutzen sind darüber hinaus die folgenden zwei Punkte. Die in Oberon eingeführte Nomenklatur reduziert die OOP auf das, was sie eigentlich ist, nämlich eine *Programmiertechnik*. Z.B. wird durch die Einführung des Begriffes *Typerweiterung* nicht eine Metapher wie Vererbung bzw. Spezialisierung in den Vordergrund gestellt. Selbst, wenn damit vielleicht noch nicht das gesamte Spektrum aller Möglichkeiten abgedeckt ist, welches das Konzept der Vererbung im allgemeinen eröffnet, so dürfte die Typerweiterung von Oberon doch wohl den Kern treffen. Zweitens wird durch die Betonung des Konzeptes der Erweiterbarkeit deutlich, daß OOP nicht einen Wert an sich darstellt, sondern eine Technik, erweiterbare Systeme entsprechend der heutigen Vorstellung von diesem Begriff zu erstellen. Oberon bietet an dieser Stelle besondere Möglichkeiten, die allerdings nur im Zusammenspiel mit dem Oberon-System voll ausgeschöpft werden können, in diesem Sinne also nicht eigentlich eine Spracheigenschaft darstellen.

Ein weiterer positiver Punkt ist die umfangreiche Dokumentation, die neben der zitierten

Lehrbuchliteratur on-line im Oberon-System zur Verfügung steht, darüber hinaus auch die zahlreichen Artikel und Diskussionsbeiträge im Internet. Allerdings sind hier Anfänger gelegentlich auch überfordert; elementare Lehrbuchliteratur zu Oberon, die der zu Pascal entspricht, fehlt weitgehend.

Als problematisch hat sich die relativ frühe Einführung von Oberon im Wintersemester 1994/95 herausgestellt. Auch wenn System und Sprache wesentlich älter sind, so waren die Portierungen auf Windows, die schon wegen der zur Verfügung stehenden Hardware benutzt werden mußten, noch recht neu und dementsprechend wenig stabil. Außerdem waren einige wesentlichen Komponenten nicht ausgearbeitet. So war der recht rudimentäre Editor eine permanente Quelle der Ärgernis. Für den Anfänger ist es schwierig zu unterscheiden, ob hier ein Detail noch nicht zuende bearbeitet worden ist oder ob nicht ein Fehler im Gesamtkonzept vorliegt. Er vermutet eher Letzteres. Ein spezielles Problem, welches inzwischen behoben ist, war die unzureichende personelle Unterstützung des Übungsbetriebes.

Die wesentliche Schwierigkeit, die mit der Verwendung von Oberon kaum etwas zu tun hat, liegt allerdings in dem Ansatz, in erster Linie Konzepte und nicht Sprachdetails zu vermitteln. Hier bestätigen sich weitgehend Erfahrungen, die in [CS86] wiedergegeben sind. Die Erwartungshaltung von Studierenden ist in einigen Fällen die, daß eine oder zwei Sprachen „die richtigen“ sind, die man braucht, um im Beruf bestehen zu können. Manche haben fixierte Vorlieben für die eine oder andere Sprache, häufig auch für die Möglichkeiten spezieller Rechnerhardware und vermuten daher, daß ihnen nun eine neue Vorliebe aufoktroziert werden soll, nämlich die für Oberon. Es entsteht in diesen Fällen wohl ein Gefühl wie beim Verbot, weiterhin seine Muttersprache sprechen zu dürfen.

Hinter der Vorstellung, daß das Gelernte, nicht nur eine Programmiersprache, auch sofort, direkt und im Detail im Beruf umgesetzt werden kann, steckt wohl das klassische Bild des Handwerkers, der nach der Lehre und praktischer Arbeit seine Meisterprüfung ablegt und dann ein Berufsleben lang das Gelernte anwendet. Wenn das heute in handwerklichen Berufen so wohl kaum noch zutrifft, dann erst recht nicht in Informatik-nahen Tätigkeiten. Es hat genaugenommen keinerlei Bezug zur Wirklichkeit. Alle, die sich zu didaktischen Fragen der

Informatik äußern, sind sich mehr oder weniger darin einig, daß das primäre Ziel der Ausbildung darin zu bestehen hat, auf der Basis langlebiger Konzepte das Erlernen neuer Dinge zu trainieren; mit dem Schlagwort „Das Lernen lernen“ wird dieses verdeutlicht. Es ist vielleicht aber ein gewisser Trost, daß selbst eine Hochschule wie die ETH Zürich anscheinend mit entsprechenden Problemen zu kämpfen hat.⁵

Man stellt auch fest, daß bei dem Versuch, allgemeine Konzepte durch die exemplarische Darstellung anhand der Sprache Oberon zu vermitteln, in den ersten zwei Semestern durchaus an die Grenze der Abstraktionsfähigkeiten vieler Studierender in der Phase stößt. Gerade hier zeigt sich aber die Flexibilität einer hybriden Sprache: die wesentlichen Sprachelemente von Pascal bzw. Modula werden ja auch vermittelt; und der Punkt, bis zu dem man darüber hinaus weitergeht, wird sich in jedem Semester je nach Möglichkeiten ergeben. Allerdings sollte nicht unerwähnt bleiben, daß auch die Lehrenden die Vermittlung von Konzepten mit einem neuen Hilfsmittel wie Oberon erst einüben müssen und dazu Zeit und Erfahrung benötigen.

5 Fazit und Perspektiven

Man kann die gewonnenen Erfahrungen aus der Einführung von Oberon als der ersten Programmiersprache und die sich daraus ergebenden Perspektiven wie folgt zusammenfassen.

- Für die Vermittlung neuer Konzepte hat sich Oberon bestens bewährt; die Sprache sollte als primäre einführende Sprache beibehalten werden. Ein erneuter Wechsel der Sprache zum jetzigen Zeitpunkt wäre kaum ein Fortschritt, da erfahrungsgemäß ca. 4 Semester vergehen, bis sich ein stabiler Lehrbetrieb eingependelt hat (Installation im Netz, personelle Ressourcen, Finanzierung und Beschaffung von Lehrbuchliteratur). Durch häufigeres Wechseln würde man mit großer Wahrscheinlichkeit doch nur der Entwicklung nachlaufen, denn in 2-3 Jahren wird möglicherweise auch eine Sprache wie z.B. Java, die heute sicherlich eine der sinnvollen Alternativen zu Oberon darstellt, ihr Aussehen wesentlich verändert haben.
- Die oben angesprochenen Probleme sollten

⁵Zitat von N. Wirth aus dem Vorwort von [Wir96]: „Da heutzutage alles, was nicht unmittelbar Profit abwirft, einer Rechtfertigung bedarf, will ich versuchen darzulegen, weshalb mir dieses Fach (Anm.: *Compilerbau*) trotzdem wichtig erscheint.“

natürlich nach Möglichkeit gelöst werden. Die Defizite der ersten Oberon-Portierungen sind inzwischen behoben: sowohl die ursprüngliche ETH-Version (jetzt an der Universität Linz weiterentwickelt) als auch das System 3 der ETH sind ganz wesentlich weiter ausgearbeitet.

- In Zukunft sollte als Basis das System 3 verwendet werden, weiterhin mit der Sprache Oberon-2. Oberon-2 ist ohnehin strikt aufwärtskompatibel zu dem ursprünglichen Oberon („Oberon-1“), hat aber mit seinen spezifischen Eigenschaften den didaktischen Vorteil, anderen konventionellen OO-Sprachen näher zu stehen, so daß ein späterer Übergang leichtfällt.
- Wichtig ist, daß das System vollständig mit allen seinen Möglichkeiten installiert wird, insbesondere also auch einem WWW-Zugriff und entsprechend aktiviertem Hypertext. Dieses würde die Motivation der Studierenden sicherlich wesentlich steigern.
- In der Lehre muß deutlich werden, daß in den ersten Semestern Informatik-Konzepte im Vordergrund stehen, nicht die Konstrukte einer speziellen Sprache. Eine Durchsicht neuerer Lehrbuchliteratur wie z.B. der Informatik-Einführung von Goos ([Goo96a], [Goo96b]) zeigt, daß durchaus auch formale Gegenstände bereits in die ersten zwei Semester integriert werden können, nicht erst ins 3. Semester, auch wenn eine vollständige Übernahme der in diesem Buch dargestellten Inhalte in einem Fachhochschulstudiengang wohl nicht möglich und sinnvoll wäre.
- Die Darstellung von Informatikkonzepten bedarf einer Notation; eine Möglichkeit, die oben als Problem geschilderte zu enge gedankliche Assoziation zwischen Konzepten und einer speziellen Notation, also einer Programmiersprache, zu vermeiden, besteht darin, in einem sehr frühen Stadium bereits eine zweite Sprache einzuführen. Es wird daher vorgeschlagen, bereits im 2. Semester zu Vergleichszwecken eine weitere Sprache neben Oberon einzuführen, wobei unterstellt wird, daß sich die Studierenden – mit entsprechender Unterstützung im Labor – Kenntnisse zu elementaren Sprachkonstrukten wie

z.B. Kontrollstrukturen selbst aneignen. Es kommen eine Reihe von Sprachen in Frage; dazu gehören Java und Smalltalk. Dieses wäre noch zu diskutieren. Man könnte auf diese Weise mit relativ großer Leichtigkeit flexibel auf „modische Entwicklungen“ reagieren, ohne gleich ein Gesamtkonzept umstoßen zu müssen. Das Grundproblem jeder Informatikausbildung, nicht immer genügend flexibel auf Entwicklungen reagieren zu können, würde dadurch zumindestens deutlich entschärft. Ein Blick auf die [html-Seite Oberon in Education](#) zeigt, daß auch andere Hochschulen einem ähnlichen Konzept folgen. Ohne, daß hier damit eine Prognose über aktuelle Sprachentwicklungen abgegeben werden soll, sei in diesem Zusammenhang daran erinnert, daß noch vor wenigen Jahren die Programmierung von Transputern, insbesondere auch mit der Sprache Occam, ein beherrschendes Thema war, das heute seine Aktualität vollständig eingebüßt hat. Es werden auch in Zukunft manche aktuellen Themen heutiger Zeit ähnlich schnell wieder aus dem Blickfeld verschwinden.

- Auch für den Wettbewerb um Studienplatzbewerber hätte dieses Konzept die positive Wirkung, daß stets auf eine außerordentlich aktuelle Komponente in der Lehre hingewiesen werden kann. Nicht alle Studierenden lassen sich schon am Anfang des Studiums von der Wichtigkeit allgemeiner Informatikkonzepte überzeugen; diesen würde so der (emotionale) Einstieg erleichtert, auch wenn die konkreten Anforderungen steigen.
- Die Befürchtung, die Anforderungen könnten auf diese Weise insgesamt zu hoch werden, ist wohl nicht gerechtfertigt; es würde sich eher um eine Anpassung von Anforderungen in der Informatik-Einführung an die anderer Fächer handeln.
- An dem Konzept, welches zur Zeit dem Hauptstudium zugrundeliegt, nämlich der Verwendung von C und C++ in den Lehrveranstaltungen *Programm- und Datenstrukturen I, II* und *III* sowie der Verwendung weiterer Sprachen in verschiedenen Wahlpflichtfächern, sollte aus meiner Sicht zur Zeit nichts geändert werden.

A Softwarebezogene Lehrveranstaltungen im Fb Informatik

Es folgt eine Zusammenstellung aller Lehrveranstaltungen der Studiengänge *Praktische Informatik* und *Technische Informatik* im Fachbereich Informatik der FH Braunschweig/Wolfenbüttel, die einen direkten Bezug zur Softwaretechnik haben. Eine Erklärung der verwendeten Abkürzungen findet man im Studienführer.

Grundstudium					
	Wochenstunden			Prüfungen	
	1. Sem.	2. Sem.	3. Sem.	Art	Gewicht
Studiengänge Technische und Praktische Informatik:					
Informatik I	4	–	–	FP K2	1/3
Labor Informatik I	2	–	–	SL ED	B
Informatik II	–	4	–	FP K2	1/3
Labor Informatik II	–	2	–	SL ED	B

Hauptstudium					
	Wochenstunden			Prüfungen	
	4. Sem.	6. Sem.	7. Sem.	Art	Gewicht
Pflichtfächer					
Studiengang Technische Informatik:					
Programm- und Datenstrukturen I	2	–	–	FP K2	1/3
Softwarelabor I	2	–	–	SL ED	B
Programm- und Datenstrukturen II	–	2	–	FP K2	1/3
Softwarelabor II	–	2	–	SL ED	B
Programm- und Datenstrukturen III	–	–	2	FP K2	1/3
Softwarelabor III	–	–	2	SL ED	B
Studiengang Praktische Informatik:					
Programm- und Datenstrukturen I	2	–	–	FP K2	1/3
Softwarelabor I	2	–	–	SL ED	B
Programm- und Datenstrukturen II	–	2	–	FP K2	1/3
Softwarelabor II	–	2	–	SL ED	B
Programm- und Datenstrukturen III	–	–	2	FP K2	1/3
Softwarelabor III	–	–	2	SL ED	B
Softwaretechnik I	2	–	–	FP K2	1/3
Softwaretechnik II	–	4	–	FP K2	1/3
Softwaretechnik III	–	–	4	FP K2	1/3
CASE-Labor	–	2	–	SL ED	B

Die Wahlpflichtfächer sind den Gruppen von Pflichtfächern zugeordnet.

Hauptstudium					
			Wochenstunden		Prüfungen
			4. Sem.	6. Sem.	7. Sem.
					Art Gewicht
Wahlpflichtfächer					
Studiengang Technische Informatik (zu Pflichtfach Programm- und Datenstrukturen):					
Systemmodellierung			–	4	–
Systemmodellierung-Labor			–	2	–
Studiengang Praktische Informatik (zu Pflichtfach Programm- und Datenstrukturen):					
Systemmodellierung			–	4	–
Systemmodellierung-Labor			–	2	–
Studiengang Praktische Informatik (zu Pflichtfach Softwaretechnik):					
Techniken der Computersimulation			2	–	–
Computersimulation-Labor			2	–	–
Formale Grundlagen der Softwaretechnik			–	4	–

Literatur

- [Bal90] H. Balzert, Hrsg. *CASE. Systeme und Werkzeuge*. BI Wissenschaftsverlag, 1990.
- [CS86] V. Claus und A. Schill. Informatikkenntnisse von Jugendlichen, untersucht am Beispiel der drei Bundeswettbewerbe Informatik. *Informatik Spektrum*, 9(5):270–279, 1986.
- [Fre96] J. Freytag. Empfehlungen der Gesellschaft für Informatik für das Informatikstudium an Fachhochschulen. *Informatik Spektrum*, 19(1):20–32, 1996.
- [Goo96a] Gerhard Goos. *Vorlesungen über Informatik. Grundlagen und funktionales Programmieren*, Band 1. Springer Verlag, 1996.
- [Goo96b] Gerhard Goos. *Vorlesungen über Informatik. Objektorientiertes Programmieren und Algorithmen*, Band 2. Springer Verlag, 1996.
- [GS94] Heinz-Peter Gumm und Manfred Sommer. *Einführung in die Informatik*. Addison-Wesley, 1994.
- [ISO89] *ISO: ESTELLE - A Formal Description Technique Based on an Extended State Transition Model, IS 9074*, 1989.
- [Lud93] J. Ludewig. Sprachen für das Software-Engineering. *Informatik Spektrum*, 16(5):286–294, 1993.
- [LWW96] Annegret Liebers, Dorothea Wagner und Karsten Weihe. C++ im Nebenfachstudium: Konzepte und Erfahrungen. *Informatik Spektrum*, 19(5):262–265, 1996.
- [Mö93] H. Mössenböck. *Object-Oriented Programming in Oberon-2*. Springer, 1993.
- [MW91] H. Mössenböck und N. Wirth. The programming language Oberon-2. *Structured Programming*, 12:179–195, 1991.
- [Rei91] M. Reiser. *The Oberon System. User Guide and Programmer's Manual*. Addison-Wesley, 1991.
- [RW92] M Reiser und N. Wirth. *Programming in Oberon. Steps beyond Pascal and Modula-2*. Addison-Wesley, 1992.
- [SCG93] SIG-Computer-GmbH. Mars wird in Eiffel programmiert. Eiffelpost 6, August 1993.
- [WG92] N. Wirth und J. Gutknecht. *Project Oberon. The Design of an Operating System and Compiler*. Addison-Wesley, 1992.
- [Wir85] N. Wirth. *Programmieren in Modula-2*. Springer-Verlag, 1985.
- [Wir94] N. Wirth. Gedanken zur Software-Explosion. *Informatik-Spektrum*, 17(1):5–10, 1994.
- [Wir96] N. Wirth. *Grundlagen und Techniken des Compilerbaus*. Addison-Wesley, 1996.