

Harro von Lavergne

Thesen zur aktuellen Orientierungslosigkeit

Welche Schwerpunkte kann die Informatik-Ausbildung künftig haben?

Die nahezu beliebige Vielfalt der seit einiger Zeit vorgeschlagenen Themen für "zeitgemäße" Vorhaben im Informatikunterricht läßt darauf schließen, daß nach dem Ende der kleinen Programme auch die großen Projekte im legendären Turbo-PASCAL nicht länger aktuell sind (wer möchte zum x-ten Male den Versuch machen, ein Schulverwaltungsprogramm auf der DOS-Ebene zu realisieren?).

Woran liegt das?

Nehmen wir an, das Schulverwaltungsprogramm sei erfolgreich implementiert (gab es so etwas ein einziges Mal?). Aus der Sicht eines Rechnerbenutzers wird das Programm wahrscheinlich schwerfällig zu bedienen sein - die auch bei kommerziellen Programmen teilweise heute noch übliche Gängelung wirkt lästig und ist fehleranfällig. Man denke als Beispiel an die Gewöhnungsbedürftigkeit von Geldautomaten mit ihrer schrittweisen starren Logik - eine einzige falsche Reaktion des Benutzers, und der ganze Vorgang muß abgebrochen und wiederholt werden.

Aus der Sicht der Programmierstellung zeigt das Beispiel die Notwendigkeit, jahrzehntealte Vorstellungen vom ablaufgesteuerten Programm (in PASCAL das "eigentliche" Programm, das Hauptprogramm) aufzugeben; die Mensch-Rechner-Kommunikation entspricht augenscheinlich nicht der Logik einer Turing-Maschine.

An die Stelle des ablaufgesteuerten Programms tritt das ereignisgesteuerte Programm, bei dem der Benutzer (in möglichst weiten Grenzen) die Kontrolle über den Programmablauf behält. Eine Geldabhebung am Automaten könnte auch so verlaufen, daß der Kunde ein elektronisches Formular ausfüllt, wobei Irrtümer uneingeschränkt korrigierbar sind, und anschließend die Auszahlung veranlaßt.

Auf der Ebene der Programmiersprachen geht mit der Akzentverschiebung vom ablaufgesteuerten zum ereignisgesteuerten Programm die Weiterentwicklung von der prozeduralen zur objektorientierten Programmierung einher, bei der nicht mehr das Wie (die Prozedur) sondern das Was (die Daten) im Vordergrund steht.

Die damit notwendig verbundenen Änderungen ganzer Programminhalte und -stilarten sind eine Chance und eine Herausforderung für den Informatikunterricht. In Zukunft werden Algorithmen und Programmentwurf wesentlich andere und wesentlich neue Inhalte des Unterrichts darstellen.

Das führt zu folgenden Thesen:

- Algorithmen und Programmkonstruktion stehen (zumindest in Abiturstunden) weiter im Zentrum
- Die Inhalte sind neu zu definieren
- Verbunden damit stellt sich die Frage nach geeigneten Werkzeugen (Programmiersprachen)

Was gibt es bisher?

PASCAL ist, trotz ihrem Alter, zur Zeit wahrscheinlich (noch) die im Anfängerunterricht der Schule hauptsächlich verwendete Programmiersprache.

Die weite Verbreitung von PASCAL liegt sicher daran, daß diese Sprache zu ihrer Zeit ein gelungener Entwurf war, der mit einer klaren Syntax und einem geringen Umfang an Sprachkonstrukten (bisher) alle wesentlichen Konzepte des Software Engineering vermitteln ließ:

- Skalare (compilerinterne) Datentypen
- Datenkapselung durch strukturierte (benutzerdefinierte) Datentypen
- Strukturiertes (prozedurales) Programmieren
- Dynamische Datenstrukturen über Zeiger (ohne direkte Speicheradressierung, eine der Hauptfehlerquellen von C-Programmen)

Mängel (der ursprünglichen Definition) von PASCAL:

- Keine Möglichkeit der Modularisierung, d. h. keine weitergehende Datenkapselung (information hiding)
- Keine Möglichkeit der verteilten Programmerstellung in Arbeitsgruppen (distributed programming)

Diese Defizite wurden gemildert mit der Weiterentwicklung von PASCAL zu MODULA, bzw. notdürftig mit der (hauptsächlich von Borland geschaffenen) PASCAL-Spracherweiterung durch die Einführung von Modulen (Units). Diese boten, wenn auch (bezogen auf Borlands PASCAL) in unvollkommener Form, folgende Programmiermöglichkeiten:

- Abstrakte Datenstrukturen, abstrakte Datentypen
- Prozedurtypen und offene (dynamische) Prozedurparameter (ab Turbo-PASCAL Version 7)
- Separate Kompilierung mit Schnittstellenüberprüfung

Gegenüber PASCAL haben die meisten anderen Sprachen (bis auf MODULA-2) im Schulunterricht keine wesentliche Rolle spielen können, insbesondere sind alle Versuche, C++ (als die Industriesprache) zu einer Ausbildungssprache in der Schule zu machen, wegen der Komplexität von C++ erfolglos gewesen.

Welche Veränderungen sind nötig?

Die vergangenen 10 bis 15 Jahre haben deutlich gemacht, daß der Ausweg aus der sogenannten software crisis sowie die (globale) Vernetzung (WWW) neue Formen von Anwendungsprogrammen und der Programmerstellung erfordern. Das Schlüsselwort in diesem Zusammenhang heißt Komponenten.

Komponenten erfordern eine über das Bisherige hinausgehende Kapselung von Daten; Komponenten sind autonome Programmeinheiten, die absichtlich unabhängig von einer speziellen Hardware erstellt werden, da sie (im Prinzip) auf allen Plattformen einsetzbar sein sollen. Komponenten greifen auf Komponenten zu und bedienen Komponenten. Jede Komponente kann in einer beliebigen Programmiersprache erstellt worden sein; eine Komponente kennt von anderen Komponenten nur deren öffentlichen Teil die Schnittstelle, nicht aber die Implementierung (programming by contract).

Ein wesentlicher Ansatz zur Erstellung derartiger Komponenten ist die objektorientierte Programmierung (OOP). In Konsequenz dieser Erkenntnis sind praktisch alle gängigen Programmiersprachen inzwischen mit objektorientierten Fähigkeiten nachgerüstet worden, selten wirklich geglückt, manchmal auch, wie im Fall Objekt PASCAL, (terminologisch) falsch und in vielen Fällen mühsam zu handhaben. Außerdem sind neue, mehr oder minder genuin objektorientierte Programmiersprachen entstanden; als erste (und terminologiedefinierende) Smalltalk; außerdem, neben etlichen anderen, JAVA und OBERON.

Was ist neu an objektorientierten Sprachen?

Objektorientierte Sprachen verstärken die Daten-Kapselung dadurch, daß sie nicht nur Daten im engeren Sinn in einem abstrakten Datentyp zusammenfassen, sondern auch die zu den Daten gehörenden Prozeduren mit den Daten bündeln. Solche Bündel können bereits in statischen Sprachen, wie PASCAL und MODULA, durch ein RECORD mit Daten- und Prozedurfeldern realisiert werden. Die mit einem derartigen Datentyp verbundenen Möglichkeiten lassen sich allerdings erst vollständig nutzen, wenn der Datentyp dynamisiert wird.

Dynamische (objektorientierte) Sprachen ergänzen die Datenkapselung durch ein wesentlich neues Konzept, das Vererbung genannt wird und in statischen Sprachen wie PASCAL unbekannt ist. Vererbung bedeutet, daß ein abstrakter Datentyp erweitert werden kann, wobei es möglich ist, die Erweiterung unabhängig von der Erstellung des Basistyps an anderer Stelle und zu anderer Zeit durchzuführen, ohne daß der Basistyp davon berührt wird. Die Zusammenfassung von Daten, Prozeduren und Vererbung in einem Datentyp wird Klasse genannt. Prozeduren solcher Klassen heißen Methoden, Variablen einer Klasse werden als Objekte oder auch Instanzen der Klasse bezeichnet.

Unnötigerweise führen die meisten Sprachen Klassenkonstrukte entweder als einzige oder als gesonderte, zusätzliche Datentypen ein; OBERON ist eine der wenigen Ausnahmen, jedes RECORD ist dort eine (potentielle) Klasse, da das Prinzip der Vererbung (Typ-Erweiterbarkeit) mit dem bereits in den statischen Vorgängersprachen bekannten RECORD-Typ verschmolzen ist.

Damit Klassen und Objekte zusammenwirken können, ist bei allen OO-Sprachen eine Systemumgebung erforderlich, die Koexistenz und Datenaustausch der Objekte ermöglicht. SMALLTALK, DELPHI, JAVA oder OBERON erzeugen keine .EXE-Dateien, vielmehr werden die (kompilierten oder interpretierten) Datenbündel erst zur Laufzeit verknüpft (late binding), statt (wie in PASCAL oder MODULA) zur Kompilationszeit (early binding). Im kommerziellen Bereich stellt Microsofts Windows mit OLE (object linking and embedding), das auf dem Microsoft component object model (COM) basiert, ansatzweise eine solche Systemumgebung zur Verfügung, die Windows-registry enthält sämtliche Schnittstellenbeschreibungen der installierten Komponenten, die Datenbündel heißen .DLL-Dateien.

Eine derartige Systemumgebung kann (prinzipiell) zu jeder Zeit durch zusätzliche Komponenten erweitert werden, sie ist also immer offen. Als Konsequenz ist eine manuelle Speicherverwaltung (Freigabe nicht mehr benötigten Speichers - in MODULA beispielsweise mittels der Prozedur DISPOSE - nicht möglich, OO-Systeme arbeiten mit automatischer Speicherbereinigung (garbage collection). COM enthält keinen garbage collector, die meisten von uns werden die daraus entstandenen Programmfehler in Form eines hängenden oder abgestürzten Systems kennen; als Konsequenz hat Microsoft garbage collection in der vor kurzem veröffentlichten Spezifikation von COM+ eingefügt.

Welche Sprache eignet sich am ehesten für den (Anfänger-)Unterricht?

Abgesehen von einigen Exoten (z.B. EIFFEL) kommen in Zukunft nach meiner Ansicht zwei Sprachen für den Unterricht in Frage: JAVA und OBERON.

Konzeptionell sind JAVA und OBERON in vieler Hinsicht ähnlich, beide sind im Hinblick auf die Programmierung von Komponenten konzipiert worden, beide daher prinzipiell für den Einsatz in der Schule geeignet.

Bei der Entscheidung für die eine oder andere Sprache sollten zwei Kriterien entscheidend sein:

- Leichte Erlernbarkeit
- Inkrementelle Erlernbarkeit

Leichte Erlernbarkeit setzt primär voraus, daß die Sprache gut strukturiert und von allem unnötigen Ballast frei ist. Inkrementelle Erlernbarkeit setzt voraus, daß die einzelnen Sprachkonstrukte soweit möglich abgeschlossen sind, also höchstens linear aufeinander bauen. Leichte Erlernbarkeit und inkrementelle Erlernbarkeit erfordern insbesondere eine möglichst geringe, wenn auch in der praktischen Programmierarbeit nicht völlig vermeidbare Verknüpfung von Sprache und Systemumgebung. Berücksichtigt man die genannten Kriterien, sprechen mehrere Aspekte für den Einsatz von OBERON und gegen JAVA.

OBERON basiert auf jahrzehntelanger Erfahrung mit den Sprachen PASCAL und MODULA und ist wie diese eindeutig (aber nicht ausschließlich) für Lehre und Forschung konzipiert worden. Bei der Entwicklung der Sprache und des Systems standen Verständlichkeit und leichte Erlernbarkeit im Vordergrund. Daher ist die Sprachdefinition von OBERON durch das Weglassen aller unwesentlichen Anteile der Vorgängersprachen vom Umfang her eine der kleinsten zur Zeit existierenden Programmiersprachen überhaupt, dennoch lassen sich in OBERON alle wesentlichen Konzepte der OOP realisieren. Ebenso ist die Struktur der OBERON-Systembibliotheken klar gegliedert und (relativ) einfach zu verstehen.

JAVA basiert syntaktisch (nicht von der Funktionalität her) auf C++. Das ist eine absichtsvolle Design-Entscheidung, die C++-Programmierern den Umstieg auf JAVA erleichtern soll, aus der Sicht (Gewohnheit) der Schule könnte dies eher gegen JAVA sprechen. Umgekehrt erspart die PASCAL-ähnliche Syntax von OBERON müden Lehrern das Erlernen einer völlig neuen Programmiersprache, schon das kann ein Argument für den Einsatz dieser Sprache im Schulunterricht sein.

JAVA ist erheblich jünger als OBERON und eindeutig für den kommerziellen Einsatz konzipiert. Die Systemumgebung wird laufend neu auftauchenden Anforderungen angepaßt, Bibliotheken werden ergänzt und erweitert, die Sprache ist (zumindest derzeit noch) nicht stabil.

Die Sprachdefinition wirkt im ersten Augenblick ebenfalls klein, ist aber groß, wenn man berücksichtigt, daß JAVA-Klassen intensiv ineinander und teilweise auch mit der Sprache verwoben sind, einige Sprachkonstrukte lassen sich ohne Bezug auf JAVA-Klassen weder verstehen noch anwenden. Bereits die JAVA-Sprachdefinition setzt die Existenz von vier JAVA-packages in der Laufzeitumgebung voraus: `java.lang`, `java.util`, `java.io`, `java.applet`. Insbesondere gibt es eine so enge Verzahnung zwischen der Sprache JAVA und dem package `java.lang` (Klassen: `Object`, `String`, `Throwable`), daß man dieses package in der Praxis als Bestandteil der Sprachdefinition ansehen muß.

Das package `java.lang` enthält im JAVA Development Kit 1.0 bereits 21 Klassen und 2 Schnittstellen (interfaces), die insgesamt 254 Methoden einführen (das JDK 1.1 hat diese Zahlen noch einmal erheblich erhöht). Auch ein Anfänger benötigt bei der Programmierarbeit eine gute Kenntnis des package, der Umgang mit JAVA ist schwieriger, als er im ersten Augenblick scheint und für alle Gelegenheitsprogrammierer zeitaufwendig.

Generell erfordert JAVA-Programmierung eine weitgehende Kenntnis der JAVA-Klassenbibliotheken, da die Klassen stark aufeinander bezogen sind. Dies mag im Hinblick auf kommerzielle Anwender problemlos sein, nach einer Eingewöhnungszeit sind wahrscheinlich auch extreme Anforderungen an JAVA-Applikationen programmierbar, für den Unterricht erscheint JAVA jedoch weniger geeignet.

Dagegen läßt sich der Unterricht in OBERON fast völlig unabhängig von den System-Bibliotheken gestalten, ein von mir durchgeführtes Konzept, das sämtliche Datenstrukturen bis hin zu linearen Listen und Binärbäumen vermittelt, benötigt lediglich zwei Standardmodule (`In` und `Out`). Bezieht man die Manipulation der Daten auf dem Bildschirm (Schriftattribute, Fenster, Menüleistenstellung etc.) und auf externen Speichermedien ein, werden zusätzlich Verfahren aus maximal acht weiteren Bibliotheksmodulen benötigt.

Weiterhin ist gerade für Schüler, die naturgemäß auch in einfachen Fällen bei der Erstellung eines lauffähigen Programms zahlreiche Fehler machen, eine Entwicklungsumgebung sinnvoll, in der das Erstellen und Austesten eines Programmentwurfs möglichst einfach zu handhaben sind.

Alle OBERON-Systeme stellen integrierte Entwicklungsumgebungen dar, in denen der Quelltext eines Moduls im Systemeditor erstellt und ohne Zwischenschritte kompiliert werden kann. Eventuell auftretende Programmfehler werden vom Compiler im Quelltext markiert, wobei die Marken jeweils Fehlerbeschreibungen enthalten; bei einem erneuten Kompilationsversuch werden die Fehlermarkierungen automatisch entfernt. Korrekt kompilierte Programme können unmittelbar im Anschluß von beliebiger Stelle des Systems durch Aktivierung eines der im Programm vorhandenen Kommandos gestartet werden. Die Ausgabe des Programms erscheint in einem Textfenster und ist dort beliebig bearbeitbar. Bei Laufzeitfehlern des Programms wird automatisch ein kommentierter Speicherauszug erstellt, der sich beim Auffinden des jeweiligen Fehlers als sehr hilfreich erweist.

Zur Erstellung von JAVA-Quelltexten benötigt der Programmierer einen eigenen Editor (was sicher in den meisten Fällen kein Problem darstellt), muß den erstellten Text jedoch vor jedem Kompilationsversuch als (ASCII-) Datei abspeichern und anschließend auf der Kommandozeilenebene des jeweiligen Betriebssystems den Compiler mit dem vollständigen Pfadnamen der Quelltextdatei als Parameter starten. Auftretende Fehler werden in der nicht

edierbaren Bildschirmausgabe eines Quelltextauschnitts markiert. Zur Korrektur des Fehlers muß wiederum der Editor gestartet, der Quelltext geändert, gespeichert und erneut auf Kommandozeilenebene kompiliert werden.

Ist das Programm korrekt kompiliert, wird eine JAVA-Klassendatei erstellt. Zur Ausführung muß wiederum auf Kommandozeilenebene der JAVA-Interpreter mit dem vollständigen Pfadnamen der Klassendatei aufgerufen werden. Die Ausgabe des Programms erfolgt in nicht edierbarer Form ebenfalls auf dem Kommandozeilen-Bildschirm, gegebenenfalls über Umleitung in eine Datei.

Alternativ können in prinzipiell gleicher Weise wie beschrieben JAVA-applet-Klassen erstellt werden, zur Darstellung der Programmausgabe muß in diesem Fall in einer Skriptsprache - beispielsweise HTML - eine Datei erstellt werden, die die Programmausgabe in einem geeigneten Editor - bei HTML in einem Browser - durchführt. Wie praktikabel solche Verfahren im Unterricht sind, mag der Leser selbst beurteilen.

Um einen Eindruck der beiden Programmiersprachen zu geben, sei in beiden Sprachen das bekannte Hello World-Programm vorgestellt (siehe Kasten *Vergleich*). Beide Beispiele sind den Dokumentationen der jeweiligen Systeme entnommen, das gleiche gilt für die jeweiligen Erläuterungen.

Vergleich	
OBERON	JAVA
<pre> MODULE Hello; IMPORT Out; PROCEDURE World*; BEGIN Out.String("Hello World!"); Out.Ln; END World; END Hello. </pre> <p>Kompilationseinheit ist das Modul (OBERON ist eine Hybrid-sprache, die Konzepte der objektorientierten und der klassischen prozeduralen Programmierung vereinigt).</p> <p>Benutzte Module, hier <i>Out</i> müssen explizit importiert werden, durch die IMPORT-Klausel erfahren Compiler und Programmierer, daß es sich bei <i>Out</i> um ein externes Modul handelt.</p> <p>(Die vollständige Schnittstelle des Moduls <i>Out</i> lautet</p> <pre> DEFINITION Out; PROCEDURE Char (ch: CHAR); PROCEDURE Int (i, n: INTEGER); PROCEDURE Ln; PROCEDURE Open; PROCEDURE Real (x: REAL; n: INTEGER); PROCEDURE String (str: ARRAY OF CHAR); END Out. </pre>	<pre> class HelloWorldApp { public static void main(String[] args) { System.out.println("Hello World!"); } } </pre> <p>Kompilationseinheit ist die Klasse (JAVA ist eine objektorientierte Sprache, die nur Klassen und Packages kennt).</p> <p>Benutzte Klassen müssen prinzipiell explizit importiert werden. Eine Ausnahme bilden alle Klassen aus dem package <i>java.lang</i>, das automatisch von allen JAVA-Programmen importiert wird.</p> <p>Programmblöcke werden durch Paare geschweifter Klammern gebildet. Einziger Bestandteil der Klasse <i>HelloWorldApp</i> ist die Methode <i>main</i>. Eine Methode dieses Namens muß in jeder JAVA-Applikation vorkommen, sie stellt das "Hauptprogramm" dar.</p> <p>Die Methode <i>main</i> wird exportiert durch den Bezeichner <i>public</i> (der Bezeichner <i>static</i> erklärt die Methode als klassengebunden, <i>void</i> deklariert sie als Prozedur [im Gegensatz zu einer Funktionsprozedur mit Wertrückgabe]). Die Parameterliste(<i>String[] args</i>) legt mit dem Bezeichner <i>String[]</i> den Datentyp Zeichenkettenarray für den formalen Parameter <i>args</i> fest; obwohl <i>args</i> in der Methode nicht aufgerufen wird, ist dies nötig, damit eine Zeichenkettenkonstante</p>

Bei den von Out exportierten Bezeichnern handelt es sich um übliche Prozeduren, nicht Methoden. Für die Datenverarbeitung gibt es in den OBERON-Systembibliotheken selbstverständlich ausgefeiltere Moduln, *Out* ist ebenso wie *In* ein absichtsvoll einfach gehaltenes Modul, das für den Unterricht ausreicht.)

Programmblöcke innerhalb eines Moduls sind Prozeduren.

Einzigster Bestandteil des Moduls *Hello* ist die Prozedur *World*.

World wird exportiert, dies geschieht durch Anfügen der Exportmarke * an den Prozedurnamen.

Der Prozedurrumpf verwendet zwei Bezeichner, *Out.String()* und *Out.Ln* die die Zeichenkettenkonstante "Hello world!" ausgeben bzw. einen Zeilenvorschub erzeugen. Alle importierten Bezeichner müssen mit dem Namen des exportierenden Moduls qualifiziert werden.

Exportierte parameterlose Prozeduren stellen die Endbenutzerschnittstellen dar, sie können innerhalb der jeweiligen Systemumgebung durch Anklicken (oder äquivalente Aufrufe) aktiviert werden. Die Aktivierung des Bezeichners *Hello.World* erzeugt die Ausführung der entsprechenden Kommando-Prozedur.

verarbeitet werden kann).

Der Bezeichner *System.out.println("Hello world!")* besteht aus zwei verschiedenen Teilen, *System.out* und *println("Hello world!")*.

System.out verweist auf die Variable *out* aus der Klasse *System* im package *java.lang*, die die Daten an den Ausgabestrom weiterleitet.

System ist eine Unterklasse von *object* (jede JAVA-Klasse ist eine Unterklasse der Klasse *object*). In vollständiger hierarchischer Schreibweise wäre sie zu dereferenzieren als

java.lang.Object.System

jedoch sind JAVA-Referenzen innerhalb eines package nicht hierarchisch strukturiert, die Bezeichner der Klassen müssen nicht angegeben werden.

Der zweite Bezeichnerteil *println("Hello world!")* entstammt nicht der Klasse *java.System* sondern ist eine Methode der Klasse *java.io.Printstream* und wäre in hierarchischer Schreibweise zu dereferenzieren als

java.lang.Object.io.OutputStream.FilterOutputStream.

PrintStream.println()

wobei *println()* multivalent ist, d.h. abhängig vom jeweiligen Argument, etwa wie das PASCAL *WriteLn()*, weshalb die oben angegebenen formalen Parameter von *main* nötig sind.

(Auf den Abdruck der Schnittstellen der Klassen muß aus Platzgründen verzichtet werden.)

Literatur

- [ArGo96] Arnold, K.; Gosling, J.: The Java Programming Language. Reading, MA: Addison Wesley, 1996.
- [Bau97] Baumann, R.: JAVA Stimulans für den Informatikunterricht. LOG IN, 17 (1997), H. 5, S. 19-26
- [GoRo89] Goldberg, A.; Robson, D.: Smalltalk 80 The language. Reading, MA: Addison Wesley, 1989.
- [Her96] Hermes, A.: OOP im Unterricht Ein Plädoyer für einen gleitenden Paradigmenwechsel. LOG IN, 16 (1996), H. 4, S. 29-33 (Teil 1); H. 5/6, S. 62-67 (Teil 2).
- [Husch97] Husch, B.: Programmierumgebungen für die Schule. LOG IN, 17 (1997), H. 3/4, S. 10-15.
- [Möss92] Mössenböck, H.: Objektorientierte Programmierung in Oberon 2. Berlin: Springer, 1992.
- [Pfi97] Pfister, C.: Component Software A Case Study using BlackBox Components. Zürich: Oberon Microsystems, 1997.
- [Rei91] Reiser, M.: The Oberon System. New York: Addison Wesley, 1991.
- [ReiWi92] Reiser, M.; Wirth, N.: Programming in Oberon. New York: Addison Wesley, 1992.
- [Szy97] Szyperski, C.: Component Software Beyond Object Oriented Programming. Harlow: Addison Wesley, 1997.

Anmerkungen

Das JAVA Development Kit ist kostenfrei zu beziehen über:

<http://www.java.sun.com/>

OBERON-Systeme existieren in zwei Varianten, System 3 und System 4.

Sie sind kostenfrei zu beziehen über:

<http://www.oberon.ethz.ch/oberon/> (System 3 für Windows 3x, 9x, NT, MacIntosh, LINUX u.a.) oder

<http://oberon.ssw.uni-linz.ac.at/Oberon.html> (Version 4 für Windows 3x, 9x, NT, MacIntosh, LINUX u.a.).

Außerdem gibt es die, für die Programmierung von Komponenten optimierte OBERON-Sprachversion Component PASCAL.

Die dazugehörige Entwicklungs- und Laufzeitumgebung BlackBox ist zu Ausbildungszwecken ebenfalls kostenlos zu beziehen über:

<http://www.oberon.ch/>

Eigene Erfahrungen mit dem Einsatz der verschiedenen OBERON-Systeme im Unterricht haben die schnelle Erlernbarkeit der Sprache bestätigt, aber eine gewisse Gewöhnungsbedürftigkeit der unorthodoxen Benutzeroberfläche ergeben, die nicht für alle Schüler problemfrei ist. Diese Gewöhnungsbedürftigkeit besteht jedoch nicht, wenn man die BlackBox-Entwicklungsumgebung einsetzt, deren Benutzeroberfläche den jeweiligen plattformspezifischen Standards entspricht und daher intuitiv verständlich ist.

Erschienen in:

LOG IN 18 (1998) Heft 5, S. 19-23