

The ModulaTor

Oberon-2 and Modula-2 Technical Publication

Ubaye's First Independent Modula-2 & Oberon-2 Journal! Nr. WS, Jan-1997

Oberon-2, a hi-performance alternative to C++

(This article has been submitted for publication. Copyright may be transferred without further notice and this version may no longer be accessible.)

Günter Dotzel
ModulaWare
La Chanenche
F-04340 Meolans Revel (France)
<mailto:gd@modulaware.com>

Wojtek Skulski
Chemistry Department and Nuclear Structure Research Laboratory
University of Rochester
Rochester, NY 14623, USA
<mailto:skulski@nsrl.rochester.edu>

August 26, 1996

Abstract

Oberon is a highly efficient, general-purpose programming language, descendant of Pascal and Modula-2. It is simpler yet more powerful than its predecessors. Oberon programs are structured, modular and type-safe. Object-oriented programming is supported through the type extension mechanism, single inheritance, procedural variables, type-bound methods, data hiding and encapsulation, and garbage

collection. Both standalone and integrated Oberon compilers are available for most types of popular computer platforms.

1. Oberon: the new Pascal

Oberon [1] is a modern, general-purpose programming language which has all the essential features of other popular object-oriented programming languages, such as classes, inheritance, methods and messages. At the same time, "Oberon" is also the name of an extensible operating system [2] written in Oberon programming language. Both, the language and the system, were developed by Prof. Niklaus Wirth and collaborators at Eidgenössische Technische Hochschule (ETH) in Zürich. As a legitimate heir in the Pascal family, designed by the same person who also designed Pascal and Modula-2, Oberon is both an old and new programming language. It relies on three decades of experience and development. At the same time, Oberon is simpler yet more powerful than its predecessors Pascal and Modula-2.

The quotation from Einstein "make it as simple as possible, but not simpler" became the motto of the Oberon project. This design goal was achieved by eliminating many superfluous Modula-2 features, such as nested modules, subrange types, enumerations, variant records and a selective import statement. Module definition and implementation parts have been merged into one text file. In order to support object-oriented programming (OOP), only very few new terms were added. Besides garbage collection, the most important new language feature was type extension. Somewhat surprisingly, the OOP methodology could be fully supported as a special case of more general programming techniques offered by the Oberon language. There was little need to introduce "classes", "inheritance", "methods", etc, as special OOP terms, in addition to those already existing. In a sense, Oberon became the world's smallest, yet fully functional OOP language.

The simplicity, gained by purging the unnecessary while adding only as few new features as possible, resulted in a language which is easy to learn, simple to implement, and also very efficient. Last, but not least, it is a pleasure to work with. In this article we will try to convey this "spirit of Oberon" to the reader.

Perhaps the most important news is that the traditional procedural style is fully supported along with OOP. One can thus write an entirely traditional program in Oberon. This translates into a flat learning curve. Migration from Fortran is almost automatic, at least for those who arrange their Fortran programs neatly. Also, Pascal or Modula-2 programmers can be up and writing Oberon programs in just a couple of hours, after browsing through the compact language report and noting new features of the language. Naturally, learning the OOP techniques will take somewhat longer, but this is not due to the complexity of the language itself, but rather due to the complexity of the subject. As far as the language goes, there is almost nothing to be learned the hard way.

One of the outstanding Oberon qualities is the mandatory modular structure of Oberon programs, a feature retained from Modula-2. A simple example will illustrate the point:

```
(* Simple point-and-click on-screen calculator. How to use:  
(1) mark any integer with right mouse button, anywhere on screen  
(2) click middle mouse button on Adder.AddInt ^ *)
```

```
MODULE Adder;  
IMPORT In, Out; (* Adder uses services of other modules *)  
VAR s: REAL;    (* sum is not accessible from outside *)
```

```

PROCEDURE Clear*; (** clear the sum *)
BEGIN
  Out.String('Clearing sum'); s := 0; Out.Ln
END Clear;

PROCEDURE AddInt*;
(** take integer from screen, add to sum *)
VAR i:INTEGER; x: REAL;
BEGIN
  In.Open; In.Int(i); x := i;
  IF In.Done THEN
    s:=s+x;
    Out.String('Adding '); Out.Real(x,10); Out.Ln;
  ELSE Out.String('Read error'); Out.Ln
  END
END AddInt;

PROCEDURE Show*;
(** show the sum on screen *)
BEGIN
  Out.String('Sum:'); Out.Real(s,11); Out.Ln
END Show;

BEGIN (* module initialization section *)
  s := 0
END Adder.

Adder.AddInt ^ Adder.Show Adder.Clear

```

All Oberon programs must take form of modules, like the simple on-screen adder shown above. Compared with traditional programs written in Fortran, Pascal, or C, Oberon programs have multiple entry points, which in this case were named `Adder.AddInt`, `Adder.Show` and `Adder.Clear`, all three with obvious meaning. All entities marked with asterisks are visible outside the module and accessible for client modules, who can import `Adder` to use the services it exports. Whatever is not marked, is hidden and thus protected from outside access.

In this way, Oberon modules are divided into hidden implementations and public interfaces, both defined by the same source text. An interface of our adder module can be extracted with one of the "browser" standard programs available under all Oberon implementations. Our favorite browser called "Def" (naturally, a module itself), will extract also the specially marked comments to produce an annotated "public view" of our on-screen adder program:

```

DEFINITION Adder;
PROCEDURE Clear;
  (* clear the sum *)
PROCEDURE AddInt;
  (* take integer from screen, add to sum *)
PROCEDURE Show;
  (* show the sum on screen *)
END Adder.

```

Another somewhat unusual feature is that under the Oberon System (to be discussed later), module entry-points can be activated directly from any Oberon text-window by pointing and clicking with the mouse. Thus, it is not by accident that we put three "Oberon commands" right after the source listing above. These commands, syntactically a combination of "Modulename.Entryname", form simple yet fully

functional user interface to the adder program. It is perhaps the world's simplest "point-and-click" user interface. With no special configuration tools, any user can develop amazingly simple "point-and-click" interactive programs in a matter of minutes. Under the Oberon System environment, any text window can serve as user interface to any Oberon program, as illustrated above. Consequently, there is no conventional command shell or command line under the Oberon environment. One should also note, that in fact any Oberon System text window is editable without explicitly invoking an editor program, because a powerful multifont wordprocessor is a standard builtin environment component. In order to start editing any displayed text, it is enough to point and click the mouse at the intended spot. Program texts (typeset in color and multiple fonts) can be compiled directly either from disk files or from any text window. Examples of this facility can be found on the screen shots included with this article.

2. Object-Oriented Programming in Oberon

We now turn our attention to the object-oriented side of the Oberon language. A simple illustration is provided below. Assuming a given structured record type

```
TYPE T = RECORD x, y: INTEGER END;
```

extensions may be defined which contain certain fields in addition to the existing ones. For example, the following declarations

```
TYPE
  T0 = RECORD (T) z: REAL END; (* extension of T *)
  T1 = RECORD (T) w: LONGREAL END; (* extension of T *)
```

define new types T0 and T1, with fields x, y, z and x, y, w, respectively. Both T0 and T1 can be called "subclasses" of the base class T (this terminology is a matter of pure convention.) Furthermore, "methods" and "method calls" are introduced as follows:

```
TYPE
  action = PROCEDURE (a,b: INTEGER): INTEGER; (* a procedural type *)
  T2 = RECORD (T)
    Add: action (* "Add" is a "method" of the T2 class *)
  END;
VAR
  object: T2; result: INTEGER;
BEGIN
  object.x := 123; object.y := 456;
  result := object.Add (object.x, object.y); (* method call *)
END
```

This example shows, that indeed it was possible to introduce essential OOP concept without any special OOP terminology. All that was needed was structured extensible record types with data fields of procedural type. In this approach, a "method" is simply a procedural data field defined for a given record type. Such a method is "polymorphic" in a natural way, because, by their very nature, record fields can be redefined on the fly. Data fields such as x, y, and z, may be thought of as "class attributes".

Strictly speaking, the above approach to OOP is the one initially introduced by the Oberon language [1]. Later on, the language revision called Oberon-2 [3] introduced another, slightly different notation motivated by user convenience:

```
TYPE
  T3 = RECORD (T) END;

PROCEDURE (VAR me: T3) Add(): INTEGER;
BEGIN
  RETURN (me.x + me.y)
END Add;
```

In this example, procedure Add is a virtual method "bound to a type T3" through the "receiver" parameter me. (For clarity, the receiver must be explicitly specified. In Oberon there is no predeclared receiver name such as "self".) In derived types this method can be redefined, under the restriction that the formal parameter list remains the same (in this example it is empty):

```
TYPE
  T4 = RECORD (T3) t, u, v: INTEGER END; (* new data fields *)

PROCEDURE (VAR me: T4) Add(): INTEGER; (* "Add" is redefined *)
BEGIN
  RETURN (me.x + me.y + me.t + me.u + me.v)
END Add;
```

Invoking the method "Add" is performed as follows: result := object.Add(); this time there is no need to explicitly pass the parameters. Hence, the parameter list is now empty. The correct version of the method being called is determined at run time, depending on which type of the "instance variable" one is dealing with. If it is T3, the Add variant bound to T3 will be called, and the sum of two numbers will be returned. If it is T4, five numbers will be added together.

We want to stress, that this more conventional approach to OOP methods was introduced only for convenience of those users who already learned OOP elsewhere. The original Oberon style is still retained for the purists. The difference between the two is of secondary importance, and one can mix both OOP styles in the same program.

Because of the conceptual simplicity outlined above, the Oberon language is easy to learn and to use. It is a reasonable choice for education [5], programming in the large [2] and in consequence also for industrial software development. The rest of this article will be arranged in the "question and answer" style in order to keep presentation simple.

Does Oberon support dynamic arrays and matrices?

The multi-dimensional "open array" is the standard feature of the language. The following example shows, how to allocate at run-time a one- and a two dimensional array with 50 elements in each dimension.

```
TYPE
  Open1Array = POINTER TO ARRAY OF REAL; (*1 dim *)
  Open2Array = POINTER TO ARRAY OF ARRAY OF REAL; (*2 dim *)
```

```

VAR
  My1Array: Open1Array; My2Array: Open2Array;
BEGIN
  NEW (My1Array, 50); (* dynamic allocation *)
  FOR i := 0 TO LEN(My1Array)-1 DO
    My1Array[i] := 0.0;
  END; (* FOR i *)

  NEW (My2Array, 50,50); (* dynamic allocation *)
  FOR i := 0 TO LEN(My2Array,0)-1 DO
    FOR j := 0 TO LEN(My2Array,1)-1 DO
      My2Array[i,j] := 0.0
    END (* FOR j *)
  END (* FOR i *)
END;

```

Comparing this example with Fortran, one can see that array indices always start at 0, like in C! Naturally, one can always find a few such inconveniences in the Oberon language. Another inconvenience is lack of built-in COMPLEX data type (it is however a standard part of AlphaOberon [6]). Complex numbers have been implemented as a library module.

Is Oberon language type safe?

Type safety was the foremost consideration of the language design. Compared with Fortran or C, Oberon is almost absolutely safe, what is quite remarkable for an extensible language relying heavily on dynamically allocated data structures. Safety was achieved by static type checking during compilation time (also across module boundaries), dynamic type checks at run-time, as well as pointer safety facilitated by garbage collection. Both the compiler and the runtime system enforce that if a module interface has changed, all its clients have to be recompiled. Very few security gaps still remaining are well documented and can be easily avoided.

In order to facilitate low-level programming, features to breach the type system are provided through a special standard module named SYSTEM. The correctness of any given program module can be rigorously proven based on the imported module interfaces, under the precondition, that the import of module SYSTEM is not visible in the interface. Although symbolic post-mortem and run-time debuggers are available, in practice there is very little need for debugging Oberon programs, and no need at all for any kind of a "lint utility". Most programming errors are detected at compile time. Few remaining ones are easily pinned down with the help of the ASSERT standard procedure.

Is it garbage collected?

Garbage collection is provided and automatically triggered, when heap space is critical. For example, most implementations of Oberon layered on-top of existing operating systems such as Unix, invoke the garbage collector each time when opening a file.

Can the user define new types/classes?

Yes, this is the single most important new feature of the language, as compared to its predecessors Pascal and Modula-2. Classes are simply record types with procedures

bound to them. There is no need to duplicate the headers of bound procedures in the record as it is done in other object-oriented languages like C++ or Object Pascal. This keeps record declarations short and avoids unpleasant redundancy. The "class browser" standard tool utility allows to list the record types together with all procedures bound to them.

Does it truly support OOP?

Oberon is termed a "hybrid language". It supports OOP via extensible types, single inheritance, polymorphism, value and reference based objects, two categories of visibility, i.e.: public and hidden attributes and methods, run-time type information and persistent objects. However, Oberon is by no means "OOP only". Not everything has to be classes. Quite the opposite, the traditional procedural style is supported along with OOP. One can write an entirely traditional program in Oberon, or one can arrange everything in classes. Both approaches can also be mixed. This gives a programmer the best of both worlds, what can be attractive for those users, who want to learn OOP gradually. Migration from Fortran to Oberon is almost automatic, at least in case of cleanly written Fortran programs.

Encapsulation, dynamic binding, inheritance?

Encapsulation of abstract data types is provided through the module concept, similar to Modula-2. Separate "definition" and "implementation" modules were dropped from Oberon. Exported items are simply marked by an asterisk following their declarations, or a dash in case of the read-only export. A single source file per module simplifies the maintenance overhead. Whatever is not marked for export, is invisible for clients and thus encapsulated.

Automatic dynamic binding is supported for the "type-bound" methods explained above (the Oberon-2 method style). The traditional Oberon-1 methods are simply procedure variables assigned "by hand" to respective record data fields of procedural type. These methods are statically bound to every variable instance. They can be thus reassigned at any time. This is particularly useful in GUI systems, where the behavior of a given object can be changed on the fly by changing its handler procedure (one of object's procedural data fields). As already mentioned, both OOP styles can be freely mixed, where appropriate.

Only single inheritance is provided in Oberon. Arguably, this provides the same flexibility as multiple inheritance known from C++, but avoids the problems associated with the latter. As shown by Mössenböck [3], solutions equivalent to multiple inheritance can be easily achieved using single inheritance mechanisms. Multiple inheritance was therefore not introduced into the Oberon language. The same was true with operator overloading: benefits were not worth the cost and problems.

Is Oberon simple to learn and use?

Oberon is perhaps the simplest of all OOP languages. The complete defining Oberon-2 language report [3] has only 28 pages including examples and appendices with formal description of syntax, definition of terms, compatibility rules and the SYSTEM module. This concise language report compares favorably with hundreds of pages needed to define some other modern programming languages. Because of the

traditional side of the language, Pascal or Modula-2 programmers will need only a couple of hours to start writing Oberon programs. Naturally, learning OOP techniques will take longer, but not because of the complexity of the language itself, but rather because of the complexity of the subject. As far as the language is considered, there is almost nothing to be learned the hard way.

A remarkable feature of typical Oberon programs is their small size and small manpower needed for development and maintenance. Oberon programs are typically measured in kilobytes, not in megabytes. For example, in spite of their great sophistication, most implementations of the Oberon System can be distributed on single floppy disks, and they can be effectively supported by very small developer teams.

What kind of documentation is available?

Many articles and books were published on Oberon, see the bibliography section below. In addition, very good on-line documents are included with every Oberon System release. Text-, graphics-, expression-, and formula-editor, application programming interface, hypertext elements, various tools and games, are all thoroughly documented. The user can print these documents from within his/her environment, to have them handy.

What kind of libraries are available?

There is a standard set of library modules, common to all Oberon System releases, since it is in fact an operating system of its own. Concerning libraries specific to physics or math, there are few. This is naturally due to the fact that physicists have not used Oberon much till now. However, a few library development projects are underway. This relative lack of libraries is not as serious as it sounds, since most existing Fortran and Pascal routines port easily, due to the "traditional side" of the language. It takes usually only about half an hour to port a couple of Numerical Recipes Fortran subroutines. Changes are mostly mechanical.

Can one link Oberon with other languages?

Calling other languages from the Oberon System is system-specific and depends on whatever dynamic link/load facilities exist in the host operating system. Most Oberon System implementations allow operating system calls, such as calling Unix, Mac Toolbox, or Windows API. AlphaOberon allows to call any foreign language routine of any shareable image. This includes all system- and run-time library routines, as well as access to X11/OSF/Motif. Stand-alone Oberon-2 compilers generally allow foreign language calls.

What kind of programming environments are available for Oberon?

Among many integrated Oberon environments available, there is an outstanding one called the Oberon System [2, 4] It originated as a complete graphical operating system for a particular computer hardware developed at ETH-Zürich. Two different variants of the Oberon System, version 3 and 4, emerged by now. Architecturally, V3 is entirely based upon persistent objects, whereas V4 is not, but at the same time V4 is less complex than V3. Both versions include everything needed to efficiently

conduct everyday work, such as a compiler, a programming library including a simple yet very effective graphical user interface, and a sophisticated multifont text processor complete with hyper-text elements and pop-up menus embedded in any text (even in source programs). The standard Oberon System editor is extensible in functionality. Many useful extensions of this basic system already exist, which include text formatting, formula editor, and graphics elements.

Additionally, optional graphical user interface kits are available under both V3 and V4 environments, featuring sophisticated pop-up menus and a wide range of configurable dialog boxes (Dialogs [7], Gadgets [8]). The tool set available under V3 is particularly impressive. It includes a program formatter, spreadsheet, sorter, mailer, WWW browser, Java interpreter and many games such as Tetris and MineSweeper. Most programs are available with source code.

The Oberon System (either V3 or V4) is available for all popular platforms: Amiga, DECAAlpha (OpenVMS), MacII, PowerMac, PC (DOS, Windows, WNT/Win95, OS/2, Linux), NeXt (both Intel and 68k), Unix (DECStation/MIPS, HP-9000/HP-UX, RS6000, SGI, SunSparc/Solaris). On these platforms, the Oberon System runs as an application under the control of the host operating system. The Oberon System has exactly the same look-and-feel, regardless of the platform it runs on. Texts and graphics are portable across all platforms, regardless of byte ordering (little or big endian). Most Oberon-2 programs, written with this environment in mind, can be ported to all platforms mentioned above by simple recompilation.

Of special interest is the Native Oberon System for PC, currently in beta-testing stage. This version of the System V3 runs on a bare PC hardware. It includes all of the tools and components already mentioned.

Dynamic loading is the default mode for Oberon System. No separate linking step is required. Compiled modules are automatically loaded and linked when any item belonging to the module is referenced for the first time. This feature, called "lazy module loading", allows to load and start large application programs very fast. Neither the number of modules nor their names need to be known a priori when invoking a program.

Oberon programs can be extended at run-time by adding new modules, or by replacing an active module by a modified and newly compiled version. This can happen also when other parts of the application remain loaded into computer memory. Oberon System is thus an unusual programming environment, because it allows piece-wise modification of active applications when the applications are running. Since Oberon compilation is blazingly fast, this translates into very efficient software development cycle.

What about conventional programming environments?

For those users who prefer to work under their traditional environment, several non-integrated, command-line Oberon-2 compilers are available, both public domain and commercial. For example, a free o2c Oberon-to-C translator can be integrated in the emacs environment. o2c has been ported to a great variety of platforms. The commercial Oberon-to-C translator Ofront can be used both in the standalone command-line mode and in the integrated environment mode. The same is true for the commercial implementation for DECAAlpha: it features both, a stand-alone compiler producing OpenVMS object files, as well as the version of the compiler embedded in the Oberon System (AlphaOberon). Other variants of the Oberon System for Windows and Mac exist, which are "integrated into" the windowing system

of the host operating environment, and thus follow conventional "look and feel" of respective platforms.

Does Oberon come with source code?

The original Oberon System was published with full source including the compiler. As of time of this writing, a few other implementations were also released with full source: Amiga, Macintosh (both 68k and PowerPC) and Windows versions of the Oberon System V4. (The source of other implementations is available on request.) The Oberon-to-C translator o2c also comes with full source.

Which implementation is good for me?

With so many different Oberon implementations available, one does not fit all. Which one is the best for the reader, depends on the application. If one wants to link Oberon code with existing libraries, then either freeware o2c or commercial Ofront will be the best choice, or commercial AlphaOberon for the DECAAlpha platform. Users interested in "all in one" environment can pick up either V3 or V4 environments. Both are ideal for interactive data processing. Their invaluable asset is virtually no GUI overhead since effective GUIs are provided in different flavors. One can adopt either the austere default Oberon GUI, or very sophisticated add-on packages like Dialogs or Gadgets. One can thus develop fully interactive programs with minimum investment. (An example of an on-screen interactive adder was given above.)

What kind of applications is Oberon best for?

In the past, the programming language Pascal suffered from the "mostly teaching language" opinion. The same should not happen to Oberon, which should be regarded as a general-purpose programming language. The very first program ever written in Oberon was a sophisticated graphical operating system [2] proving the language to be a suitable tool for large-scale system programming. Oberon is currently being used for distributed programming, database access, interactive data analysis, image processing, and for CAD design. As an example, the Trianus project [9], currently underway at ETH Zürich, focuses on designing electronic circuits with Field-Programmable Gate Arrays (FPGAs). For this purpose a multiple-view editor is being developed, which presents a design textually and graphically. With the aid of this editor, a designer can manipulate a circuit layout under the constraints of a textual specification provided by the textual view. By closely coupling the representations, circuit information, e.g. signal delays, is instantly available. An example of interactive data analysis with Oberon is the Voyager project developed at StatLab Heidelberg [10]. It focuses on an extensible portable programming environment for statistical computing and simulation, based on Oberon System. This and many other Oberon projects can be accessed over the Internet. In addition to these "real world" applications, Oberon can and should become a serious alternative to Pascal in education. Several programming environments, which are freely available for the most popular PC and Macintosh platforms, afford to setup student programming courses at no cost. An instructor can teach both traditional procedural programming style and the new object-oriented techniques using the same programming language and environment. Novel object-oriented GUIs such as Dialogs or Gadgets can become standard topics of programming courses focused on

Oberon System. At the same time, versions of the environment embedded in the traditional MS-Windows and MacOS environments (also free for educational use) can be useful to teach more traditional GUI techniques. More details are available through the "Oberon in education" home page [5]

Recently a new exciting Oberon project [11] named "Juice" was released by the University of California at Irvine. Juice is a new technology for distributing executable content across the World Wide Web. It is thus similar to Java from Sun Microsystems. However, Juice outperforms Java in many "downloadable Applets" applications, especially large ones. Rather than being interpreted, as Java applets normally are, Juice always compiles each applet from its mother tongue Oberon into the native code of the target machine. Juice's on-the-fly compilation is not only very fast, but it also generates object code that is comparable in quality to commercial C compilers. Further, Juice avoids many of the Java security issues, because strong type checking makes it virtually impossible to write an applet that violates security rules imposed by the Oberon source language. The Juice project can be accessed at <http://www.ics.uci.edu/~juice/>.

3. Summary

The simplicity gained by purging unnecessary features is an invaluable asset of Oberon, which is perhaps the only programming language which solves more problems than it creates. Oberon is a sound tool to conduct serious work. The reader who wants to try out one of existing Oberon implementations, can download one them from the Internet and install in less than an hour. For beginners, one of the integrated versions is recommended. Most integrated environments fit on a single floppy, and they include documentation and host of useful tools. For serious development, either the integrated or standalone compiler can be a better choice, depending on concrete project. The Oberon development community is very vigorous. New versions of Oberon tools appear frequently.

4. Acknowledgments

We are indebted to our friends from ETH Zürich and from Univ. Linz for all their great Oberon work and for many discussions. Guy Laden contributed valuable comments on the draft version of the manuscript.

5. References

Oberon on the Web

A complete summary and details of all existing implementations of Oberon systems, compilers, related documentation, applications, an research papers, can be found on the Web at <http://www.math.tau.ac.il/~laden/Ob-pkgs.html>

The collection of The ModulaTor TechJournal is at http://www.modulaware.com/mdltr_.htm

Bibliography

[1] : N. Wirth and M. Reiser: [Programming in Oberon. Steps beyond Pascal and Modula.](#) Addison Wesley, 1992, ISBN 0-201-56543-9. Tutorial for the Oberon programming language and concise language reference.

[2] : N. Wirth and J. Gutknecht: [Project Oberon. The Design of an Operating System and Compiler.](#) Addison Wesley, 1992, ISBN 0-201-54428-8. Program listings with explanation for the whole system, including the compiler for NS32000 processor.

[3] : H. Mössenböck: [Object-Oriented Programming in Oberon-2.](#) Springer, 1993, ISBN 3-540-56411-X. Principles and applications of object-oriented programming with examples in the language Oberon-2.

[4] : M. Reiser: [The Oberon System. User Guide and Programmer's Manual.](#) Addison Wesley, 1991, ISBN 0-201-54422-9. Addison Wesley, 1992, ISBN 0-201-56543-9. User manual for the programming environment and reference for the standard module library.

[5] : "Oberon in Education" home page: [http://www-cs.inf.ethz.ch/Oberon/Education.](http://www-cs.inf.ethz.ch/Oberon/Education) See also Jürg Gutknecht's article [Oberon in Education](#) in the ModulaTor.

[6] : Günter Dotzel: OpenVMS Alpha Modula-2 and Oberon-2 Compiler Project. In: Peter Schulthess (Hsg.): Proceedings of the Joint Modular Languages Conference, Universitätsverlag Ulm, 1994. [Revised edition in HTML at http://www.modulaware.com/max_sum.htm](#)

[7] : M. Knasmüller: Oberon Dialogs, User's Guide and Programming Interface. Institut für Informatik, Johannes Kepler Universität Linz, Report No. 1, 1994. <http://www.ssw.uni-linz.ac.at/Projects/Dialogs.html>

[8] : J. L. Marais: The Gadgets User Interface Management System. Department Informatik, ETH Zürich, Report No. 144, 1990. <http://huxley.inf.ethz.ch/~marais/Spirit.html>

[9] : N. Wirth home page: <http://www-cs.inf.ethz.ch/Wirth/Gruppe.html>

[10] : The Voyager project: <http://statlab.uni-heidelberg.de/projects/voyager/>

[11] : The Juice project: <http://www.ics.uci.edu/~juice/>





Fig. 1. Oberon System 3 graphical user interface named Gadgets. The figure shows a snapshot of the Oberon desktop with several overlapping documents. Every graphical element in this figure, from the desktop to the smallest button, is a Gadget itself. The Native PC Oberon System (now in beta development stage) will feature Gadgets user interface.

Fig. 2. The Oberon System Version 4 featuring a very efficient user interface consisting of tiled windows. In addition to multiple fonts and colors, the standard Oberon editor Edit supports "active text elements" such as pull down menus and hypertext folds. Figures, like the one shown, can be edited directly in the text where they appear. Programs can be compiled directly from editor windows.

This article is also available for download in GNUzipped PostScript format at <ftp://nuchem.nsr.rochester.edu/pub/Oberon/CiP/NewestDraft.ps.gz>

This article appeared in Computers in Physics, Vol 11, No. 1 (Jan-1997), American Institute of Physics.

[[Home](#) | [Site_index](#) | [Contact](#) | [Legal](#) | [Buy_products](#) | [OpenVMS_compiler](#) | [Alpha_Oberon_System](#) | [XDS_family](#) | [ModulaTor](#) | [Bibliography](#) | [Oberon\[-2\]_links](#) | [Modula-2_links](#) | [Effekta_onduleurs](#) | [General book recommendations](#)]

 Books  Music  Video	Enter keywords... <input type="text"/> <input type="submit" value="Search"/>
	

webmaster@modulaware.com, 20-Mar-1999. © (1996-1999) Günter Dotzel, Wojtek Skulski.