Oberon News

Number 2, July 1994

Institute for Computersystems
Swiss Federal Institute of Technology
Zürich, Switzerland

The Oberon User Group
Bergstrasse 5
CH–8044 Zürich
Switzerland

The Institute for Computer Systems of the ETH Zürich and the Oberon User Group of Switzerland are pleased to announce that the Oberon newsletter is now the official information source about Oberon and related events. The newsletter is published on a semi–regular basis four times a year, to announce new Oberon versions, report about Oberon events, and to give information about the latest research and dissertations of the Institute for Computer Systems. We do our best to present in each newsletter topics that we think might be interesting for Oberon users. In this issue we have diverse topics like using Oberon in art and in education, in addition to many announcements.

We hope that you enjoy this issue. Happy reading!

DOS–Oberon System 3 Version 1.5 Release

Recently, after many months of hard work, the new DOS–Oberon System 3 version has been released. Although no conceptual changes have been made since the last version, a considerable part of the work involved incorporating suggestions from users and making smaller improvements (and of course, many bugs have been fixed!). Most important, the main change in the new version is not only the release contents, but a new, more open, attitude towards Oberon. As Oberon is our main vehicle of research and the principal programming language taught at ETH, we have many active Oberon programmers. Oberon is used extensively in student works, resulting in many new Oberon tools and packages per year. Most of the programs are thrown away and never used again, although many are of such a high quality that Oberon users start using them for private purposes. Most of these programs never leave the domains of ETH, often giving an impression of closedness to the Institute. However, in the last few years, many of the projects have shifted their emphasis from more technical and specific research to more end–user oriented applications, making them much more suitable for general use by the Oberon community. This can be partially attributed to Oberon System 3, where the Gadgets toolkit increases programmer productivity by providing large sets of program components or gadgets for reuse, and in this way enables us to give students more ambitious projects to be completed in the same time frame.

Some of the more useful applications are now provided in an application package collection with each DOS–Oberon release. Most important, more example programs and the source code of some of the application packages give more realistic examples of typical Oberon programs. In our opinion we should not only teach how to build better software, but also provide examples of how to do so, and this is exactly why so much is now donated to the public domain.

The largest improvement is the new System 3 online tutorial system with a set of tutorials ranging from using the class Oberon system to programming new gadgets (more details can be found in another article). A further improvement is in the handling of diskettes. The old version could only read and write double density diskettes; the new version reads and writes both double and high density diskettes, and provides transparent support for MSDOS diskettes in addition to our own backup format. Further improvements include better support for the Oberon desktop, an improved gadget locking model, print support for all standard gadgets, and a new gadget to make fixing program syntax errors a little less tedious. In addition, the new notebook gadgets are provided in source form, giving a stable basis for the development of container gadgets.

In conclusion, the Oberon System 3 distribution now includes the base system, the Gadgets user interface toolkit, the tutorial system, an assembler toolkit, the source of the display drivers, sample programs, the event simulation package, an enhanced text editor, a file archiving tool, a game, the demo Leda page layout version and an offline text formatter.
–Johannes Marais, Institute for Computer Systems

The Oberon Tutorial System of Oberon System 3

Each DOS–Oberon System 3 version includes an electronic tutorial system. The idea is to provide the system documentation in an electronic book form that not only allows you to jump from topic to topic in the well–known hypertext fashion, but also to make the book active by having working examples integrated inside it. So it is nothing strange to find completely working user interfaces floating in the book, or to include in the book a working panel that explains how to use that very panel. To promote the use of electronic books in Oberon, we also provide a tutorial development kit with each System 3 release in the hope that application developers will document their work in this way. We also request our students and other application developers to document in the same way (some of the packages provided with DOS–Oberon already have electronic book descriptions).

Each electronic book is structured as a normal book: you have a table of contents, the contents as paragraphs with footnotes, and an index of terms. Inside the book, in addition to buttons and other gadgets you click on, you will find highlighted words that act as hyperlinks to other places

in the book, to other books, to footnotes, or as executors of Oberon commands. Although books are fundamentally text based, you can also add panels and other gadgets inside the text. Buttons allow you to navigate backwards and forwards through the book or to backtrack your steps. A history facility ensures that you don't get lost in hyperspace.

Behind the scenes, each book is generated from a text file containing markup symbols and embedded elements. A book compiler compiles the book by reading the text file, linking the hypertext jumps with each other, collecting the index terms and building the table of contents. The compiled book is thus ensured to be consistent and can be used immediately. The editing of the book is done with a standard text editor; in this way larger changes can be made without too much trouble. Of course, to get you started with building your own books, a meta–tutorial will give you the needed instruction.

The System 3 release contains a number of tutorials including topics on how to use the textual user interface of Oberon, an introduction to the Gadgets system, how to use the Gadgets system and how to program new gadgets. –Johannes Marais, Institute for Computer Systems

The Backdrop Generator

The Backdrop generator is one of the fun Oberon System 3 applications included in each DOS–Oberon release. Backdrops or wallpapers, as they are known from other systems, are texture bitmaps that we use to beautify our desktop backgrounds. The main problem with these backdrops is finding some you like and the time you waste trying out the ones you have found! As our contribution to your lack of productivity, we now have a backdrop generator for Oberon that allows you to create and then modify backdrops on the fly. Each backdrop is generated by a user parameterized algorithm for a certain backdrop style. The student who created this tool let his fantasy free and came up with a whole set of interesting texture algorithms including fractals, surfaces, marble, textiles, spirals, bows, trees, cells, bricks, threads, clouds, molecules, coins and plasma. Each algorithm can be parameterized with a few numbers, or even be combined with others. There are countless combinations that you can try out. Afterwards, should the colors or contrast not suit your taste, a color manipulation feature allows you to remap colors, make the image darker or lighter etc. And finally you can install your new backdrop immediately. How the backdrops are generated can be read in the corresponding electronic book. We hope you have lots of fun! –Johannes Marais, Institute for Computer Systems

The Offline Text Formatter

The rule–based offline text formatter of Oberon System 3 allows you to create high quality documents with little effort. In the same manner as the well known TeX typesetting program, the formatter interprets markup symbols in a conventional text file to control the typesetting. A user–defined stylesheet controls the text formatting and can be used for creating uniform documents between several people. The guiding principle is to keep things as simple as possible for a person unexperienced in electronic typesetting. This has been made possible through the efforts of our own font designer and typesetting expert, who has made the rules of typesetting and good taste explicit in several dozen typografic rules. These rules control how the text and embedded figures are poured into the columns on the page. More demanding users can override these intelligent defaults, although this is seldom necessary. Users who have used offline text formatters might remember some of the set of cryptic markup sequences that have to be inserted into the text stream; the Oberon formatter though, many will be glad to know, uses a much simpler and intuitive strategy. The formatter tries to determine the semantic of the text contents by analyzing the fonts used (what size and style for example) and a few symbols like parenthesis and braces. This means that the source text remains easily readable and easier to edit. For example, footnotes are simply written in square brackets and titles are written in a slightly larger font. Furthermore, gadgets and pictures can be included and placed automatically on the page. As examples, a few demanding page layouts are provided with the formatter, such as a dictionary and newspaper article. The documention of the formatter is an example in itself. The offline formatter is provided with source code with each DOS–Oberon System 3 release. –Johannes Marais, Institute for Computer Systems

Leda Demo Version Available

Leda, the page layout system developed at the Institute for Computer Systems as a Ph.D. study and lately sold as a commercial product, is now provided in a demo version with each DOS–Oberon System 3 version.

Leda, an advanced, fully integrated and easy–to–use document editor for Oberon System 3, allows the simple and efficient creation of documents of different types and complexity. No distinction is made between short letters, long reports or complicated page–layouts. Leda provides enough functionality to solve sophisticated document layouts (this document was created using Leda). It distinguishes itself from other modern document editors in two ways.

The Leda–System respects basic typographical rules of documents. The typography influences sizes and attributes that can be inserted into a document. Not only are fonts, but also the formatting of the text and the layout of the pages influenced by typographical rules. The consequent use of the typographical rules simplifies document editing. A good–looking document can be created in a short time with little effort by using defaults provided by the system. Additional functionality allows the expert further control over the attributes of a document.

Additionally Leda provides a seamless transition between simple text formatting and page–layout. In the base functionality Leda supports block adjusted formatting of text. Multiple columns and a marginal form are also available. A more flexible layout can be obtained by placing diagrams and pictures in a

layout raster on the page.

Gadgets can be integrated into Leda documents, either flowing inside the text stream or as freely placeable boxes. In addition, Leda supports the direct in–place editing of mathematical formulas.

The demonstration version of Leda allows you to create complete documents but to print only the first page of the document. The full version of Leda can be ordered from A+L AG, Däderiz 61, CH–2540 Grenchen, Switzerland, Tel. +41/65-/52 03 11, Fax +41/65/52 03 79. –Johannes Marais, Institute for Computer Systems

New Dissertations

Four dissertations have recently been completed at the Institute for Computer Systems, ETH Zürich.

Separate Compilation and Module Extension, Régis Bernard Crelier

As continuous evolution in hardware results in more powerful computers, new programming techniques and concepts must be developed to master the consequently increasing software complexity. Separate compilation of modules is such a technique that has proven valuable in Modula–2 and in Oberon, among other strongly–typed languages.

The module is both the structural unit and the compilation unit of programs. Replacing a module by a new one does not affect the rest of the system, providing that the module interface has not changed. Otherwise, client modules of the modified interface have to be recompiled to maintain system consistency. The last opportunity to detect an inconsistency is when modules are linked to form an executable unit. The check usually consists in comparing, for each imported interface, the expected key of that interface, as known at compilation time of the client, with the key of the effectively supplied interface. A mismatch indicates an inconsistency. The model is simple and efficient, but not very flexible. Indeed, a minor modification of an interface, such as the insertion of a new procedure, can trigger many unnecessary recompilations.

This thesis presents two new models for fine–grained consistency checking and their implementation. These models allow the interface of separately compiled modules to be extended without requiring a recompilation of client modules. This is particularly valuable in systems with dynamic loading, where the clients of a library are not known when the library is revised or extended. Interface editing that does not require client recompilation is not restricted to extensions, since the modification of an existing item does not invalidate clients not using this particular item. Even if they use it in a way that is upward–compatible with the modification, they still do not need a recompilation.

These techniques have been implemented in the Oberon system, but they are neither specific to the Oberon Language nor to the Oberon System. They can be applied to any modular programming system in order to improve its safety, its flexibility or both. Furthemore, these techniques are not available to the programmer as a separate tool whose use remains optional, but have been fully and transparently integrated into the compiler and module loader. Safety must not be optional.

Hermes – Supporting Distributed Programming in a Network of Personal Workstations, Spyridon Gerassimos Lalis

With distributed programming it is possible to support cooperation among users in a network, and to develop programs that use resources of remote machines to enhance their availability and performance. This potential is even more important nowadays where networks of workstations become an increasingly attractive alternative to big mainframes for organizing bussinesses and computing environments. In this work, a system that promotes distributed programming in a network of personal workstations is presented. It is implemented on top of an existing operating system as a collection of modules that introduce new functionality in an incremental fashion. The key idea of our approach is to view data items as abstract objects coupled to abstract descriptions indicating how their content can be written and read. This allows generic data transfer programs to be built without a priori knowing the type of the data items that are to be transfered. As an extension of this model, an asynchronous mechanism is developed for exchanging arbitrarily complex messages over the network. At the application level, sending and receiving of messages occur asynchronously to each other so that a decoupling between the communicating processes is achieved.

Finally, using the provided primitives a component is built which supports programming of extensible application objects that can be dynamically installed, referenced, and collected over a network. Such objects are automatically notified about incoming messages, thus they operate like special state machines whose transitions are triggered by message events; this technique is appropriate for capturing a wide range of distributed programs.

Due to its clean internal structure, the system presents itself as a set of well separated, yet cooperating parts which can also be individually accessed by the programmer. This allows for great flexibility in application development. Furthermore, the system encourages a disciplined programming style and guarantees type safety which we consider to be an important property of development environments. Our implementation also demonstrates that the proposed approach leads to acceptable performance at only a modest software cost.

Metaprogramming in Oberon, Josef Templ

The term metaprogramming refers to programming at the level of program interpretation, or in other words, to extending the interpreter of a given programming language in an application specific way. Traditionally, this concept is available only in dynamic typed and interpreted languages as Smalltalk or Lisp. This thesis investigates the possibilities of metaprogramming in a statically typed and efficiently compiled programming language. In the course of a case study, we introduce metaprogramming facilities for the Oberon

programming language and system.

The result is a variant of the Oberon operating environment which allows a seamless integration of useful met–level facilities. The key to this integration is a generalized notion of persistent objects and object libraries and its application to components of Oberon programs. Types and procedures are considered to be persistent objects collected in a special kind of library, namely module. We introduce a metaprogramming protocol which allows to manipulate arbitrary data structures based on the notion of *object riders*. An object rider is an iterator which can be used to scan the components down to an arbitrary nesting level. We introduce also facilities for controlling procedure activations based on the notion of *active procedures*. An active procedure is a procedure object which has its own instance specific behaviour expressed by a message handler. Active procedures can individually respond to invocation messages and perform any computation as response.

We investigate the implications of this approach with respect to the overall system structure and to the implementation of critical components of the run–time system, such as the library loader and the garbage collector. A new approach to safe library loading and unloading is introduced as well as a simple finalization technique and a way for optimizing libraries with a large number of objects. We show that the integration of meta–programming facilities does not introduce undue static or dynamic complexity into the Oberon system. A number of realistic applications serve as proof–by–example of the feasibility of the metaprogramming approach.

Code-Generation On-the-Fly: A Key to Portable Software, Michael Franz, Verlag der Fachvereine, Zurich. ISBN 3–7281–2115–0

A technique for representing programs abstractly and independently of the eventual target architecture is presented that yields a file representation twice as compact as machine code for a CISC processor. It forms the basis of an implementation, in which the process of code generation is deferred until the time of loading. At that point, native code is created on-the-fly by a code-generating loader.

The process of loading with dynamic code-generation is so fast that it requires little more time than the input of equivalent native code from a disk storage medium. This is predominantly due to the compactness of the abstract program representation, which allows to counterbalance the additional effort of code-generation by shorter input times. Since processor power is currently rising more rapidly than disk-access times and transfer rates are falling, the proposed technique is likely to become even more competitive as hardware technology evolves.

To users of the implemented system, working with modules in the abstract representation is as convenient as working with native object-files. Both kinds of file-representation coexist in the implemented system; they are completely interchangeable and modules in either representation can import from the other kind. Separate compilation of program modules with type-safe interfaces, and dynamic loading on a per-module basis are both fully supported.

Deferring code-generation until loading time can provide several new capabilities, especially when the intermediate program representation is machine-independent and thereby portable. It is argued that the combination of portability with practicality denotes an important step toward a software-component industry. Further benefits include a potential for reducing the number of recompilations after changes in source text, and a mechanism to decide at load time whether or not run-time integrity checks should be generated for a library module, eliminating the need to distinguish between development and production library configurations. All of these new possibilities have the potential of lowering the cost of software development and maintenance.

In the long run, fast on-the-fly code-generation may even replace binary compatibility for achieving software portability among processors implementing the same architecture. Already today, different models of a processor family are diverging more and more and it is becoming increasingly difficult to serve all of them equally well with just one version of native code. If code is generated only at the time of loading, however, it can always be custom-tailored toward the specific processor that it will eventually run on.

–Compiled by Johannes Marais, Institute for Computer Systems

Oberon for Windows

Oberon for Windows is available for almost a year now. The statistics on our ftp server *neptune.inf.ethz.ch* reveal that it has become one of the most popular implementation of the Oberon family, in terms of number of downloads. While even version 1.0 was running quite stable, the current version 1.31 contains some refinements and bug fixes. A nice point–and–click installation program has been added, making the installation of Oberon *foolproof*. You just specify the target directory and click ok, and the rest will be done automatically. Editing of *autoexec.bat* is no longer necessary.

A *telnet* client application provides a terminal emulation viewer class that allows to connect to remote systems. The terminal viewer is fully integrated into the Oberon system, i.e. text may be copied over from and to the terminal viewers. The telnet client makes use of any Windows–compliant sockets implementaion installed on a system.

The Kernel has been overhauled and at that time object finalization has been made safe. Any Oberon application may now register objects for finalization, together with a finalization procedure. When the garbage collector reclaims memory, it calls the registered finalization procedure which in turn may perform cleanup operations. The file system is a client of this finalization service. Finalization handlers for file objects delete temporary files on disk if they are no longer needed *[Editors note: The object finalization implementation is based on the dissertation of Josef Templ, Metaprogramming in Oberon].*

Oberon for Windows has the future built in already: early tests have shown that it runs flawlessly on Beta versions of Microsoft's upcoming

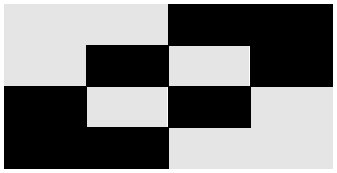
Chicago as well as on Daytona (Windows NT version 3.5). Some restrictions that are present now on Windows 3.1 systems will vanish when using Chicago. A port of Oberon System 3 is nearly finished, and will be advertised when complete. Stay tuned! –Matthias Hausner, Institute for Computer Systems

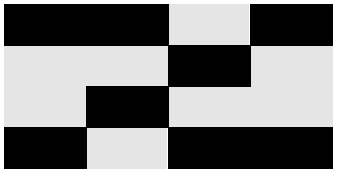
ArtworkCode 86 or how to visualize 1296 pictures

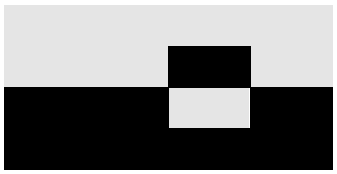
Visualization is undoubtedly one of the most attractive application of modern computers. In the run–up to *Kicking Boxes Billiard*– a current exhibition at the Graphische Sammlung of ETH Zuerich that is devoted to contemporary geometric art– we were approached by the artist Peter Schweri with the idea of visualizing his ArtworkCode 86(TM).

The basic concept of ArtworkCode 86 is *customizableart* in the sense that each individual may select *his* or *her* specimen from a complete collection of pictures. In its simplest form, ArtworkCode 86 is a set of pictures consisting of 4 x 4 colored double–squares. If (as in the examples below) two colors are used (e.g. black and white, red and green, cyan and magenta) and if the columns are required to be balanced in color (2 double–squares of each color), the complete set consists of 6 to the 4th = 1296 pictures.

Essentially, we developed two







```
MODULE ArtworkCode;

  IMPORT Files, Texts, Display, Oberon;

  CONST D = 12; H = 8; W = 2*H; dX = 4*W + D; dY = 4*H + D; (*in units of 0.1 mm*)

  VAR WT: Texts.Writer; col: ARRAY 6, 4 OF BOOLEAN;

  PROCEDURE WriteLn (s: ARRAY OF CHAR);
  BEGIN Texts.WriteString(WT, s); Texts.WriteLn(WT)
  END WriteLn;

  PROCEDURE WriteInt (i: LONGINT);
  BEGIN Texts.WriteInt(WT, i, 1); Texts.Write(WT, " ")
  END WriteInt;

  PROCEDURE PrintColumn (i: INTEGER; X, Y: LONGINT);
    VAR j: INTEGER;
  BEGIN j := 0;
    REPEAT
      IF col[i, j] THEN
        WriteInt(X); WriteInt(Y); WriteInt(W); WriteInt(H); WriteLn("rect")
      END;
      Y := Y + H; INC(j)
    UNTIL j = 4
  END PrintColumn;

  PROCEDURE PrintPicture (N: INTEGER; X, Y: LONGINT);
    VAR j: INTEGER;
  BEGIN
    WriteLn("1 0 1 setrgbcolor");
    WriteInt(X); WriteInt(Y); WriteInt(4*W); WriteInt(4*H); WriteLn("rect");
    WriteLn("0 1 1 setrgbcolor"); j := 0;
    WHILE j # 4 DO
      PrintColumn(N MOD 6, X, Y); N := N DIV 6; X := X + W; INC(j)
    END
  END PrintPicture;

  PROCEDURE Print*;
    VAR N: INTEGER; X, Y: LONGINT; ch: CHAR;
      f: Files.File; RF: Files.Rider; T: Texts.Text; RT: Texts.Reader;
  BEGIN Texts.OpenWriter(WT);
    WriteLn("%!"); WriteLn("/rect {");
    WriteLn("/rh exch def"); WriteLn("/rw exch def");
    WriteLn("newpath"); WriteLn("moveto");
    WriteLn("rw 0 rlineto"); WriteLn("0 rh rlineto");
    WriteLn("rw neg 0 rlineto"); WriteLn("closepath fill");
    WriteLn("} bind def");
    WriteLn("0.2835 0.2835 scale"); (*to units of 0.1 mm*)
    WriteLn("90 rotate"); WriteLn("120 -1856 translate");
    WriteLn("/Times-Roman findfont"); WriteLn("28 scalefont");
    WriteLn("setfont"); WriteLn("0 0 0 setrgbcolor");
    WriteLn("0 0 moveto"); WriteLn("(Artwork Code 86 by Peter Schweri) show");
    WriteLn("0 -30 moveto"); WriteLn("(Copyright (C) Peter Schweri) show");
    WriteLn("180 rotate");
    N := 0; X := -4*W; Y := -1622;
    REPEAT
      PrintPicture(N, X, Y); INC(N); Y := Y +dY;
      IF N MOD 36 = 0 THEN X := X - dX; Y := -1622 END
    UNTIL N = 1296;
    WriteLn("showpage");
    NEW(T); Texts.Open(T, ""); Texts.Append(T, WT.buf);
    f := Files.New("Artwork.PS"); Files.Set(RF, f, 0);
    Texts.OpenReader(RT, T, 0); Texts.Read(RT, ch);
    WHILE ~RT.eot DO Files.Write(RF, ch); Texts.Read(RT, ch) END;
    Files.Register(f)
  END Print;

BEGIN
  col[0, 0] := TRUE; col[0, 1] := TRUE; col[0, 2] := FALSE; col[0, 3] := FALSE;
  col[1, 0] := TRUE; col[1, 1] := FALSE; col[1, 2] := TRUE; col[1, 3] := FALSE;
  col[2, 0] := FALSE; col[2, 1] := TRUE; col[2, 2] := TRUE; col[2, 3] := FALSE;
  col[3, 0] := TRUE; col[3, 1] := FALSE; col[3, 2] := FALSE; col[3, 3] := TRUE;
  col[4, 0] := FALSE; col[4, 1] := TRUE; col[4, 2] := FALSE; col[4, 3] := TRUE;
  col[5, 0] := FALSE; col[5, 1] := FALSE; col[5, 2] := TRUE; col[5, 3] := TRUE
END ArtworkCode.
```

(trivial) Oberon programs for ArtworkCode 86, one that generates a complete catalogue on a single page in the form of an array of 36 x 36 pictures, and one that displays form–filling any arbitrary picture of the collection (given by its ordinal number). In both cases, the Oberon program produces a portable PostScript(TM) file that may be fed to any PostScript printer or photo exposure device of any size and quality. Thanks to this technique we were able without additional effort to produce

(a) a large poster–sized version of the catalogue and (b) high–quality lithographs of some carefully selected pictures. For the interested reader we present the Oberon program that generates the complete ArtworkCode 86 catalogue in green on red. –Jürg Gutknecht, Institute for Computer Systems. ArtworkCode 86 is a trademark of Peter Schweri. PostScript is a trademark of Adobe Company.

Joint Modular Languages Conference (JMLC)

This conference will be held from September 28th–30th 1994 in Ulm, Germany. Having evolved from both the former *International Modula–2 Conference* and the former *European Modula–2 Conference*, JMLC is actually two conferences in one.

Besides of unification, the conference has also widened its scope. Although still concentrating on the Pascal–Modula–Oberon line (Niklaus Wirth will be the keynote speaker), JMLC features comparative language studies and contributions that focus on other advanced modular languages, such as the Beta language.

Also noticeable are tutorials on Oberon, Ada 9X and C++ that are offered as a prelude to the conference and an extensive social program, aimed at introducing historical Ulm to accompanying persons. The conference fee is budgeted to approximately 350 DM. Discounts are available for GI, SI and BCS SIG members and for students. –Jürg Gutknecht, Institute for Computer Systems. Further information can be obtained from and applications can be placed at:

JMLC Conference
Secretary
Verteilte Systeme, Informatik
Universitaet Ulm
Oberer Eselsberg o–27
D89069 ULM (Germany)
Phone: (++49)–731/502–4140
Fax: (++49–731/502–4142
E–mail: schulthe@informatik.uni–ulm.de

The Oberon User Group

The Oberon User Group (OUG), a sub–goup of the Schweizerischen Informatiker Gesellschaft (SI), has the goal to popularize Oberon and to collect all information about the Oberon language and system. Every year we organize the Oberon Day, one of the major *Oberon* events of the year, where we present a full day of talks by well–known speakers around a specific topic. We also publish the Oberon Newsletter together with the Institute for Computer Systems. The User group provide user support by e–mail and has a telephone hotline (details at the end of the newsletter). –Markus Dätwyler, Oberon User Group

The Oberon Day '94

The programming language and operating system Oberon has been successfully used as a computer science education medium at ETH and other educational institutions for the last few years. The pragmatic simplicity and understandability of the system, together with its complete and concise documentation makes it an ideal educational tool. Oberon success stories include applications like the *Electronic Book* and a system for the design and programming of digital components (FPGA). Oberon is freely available for different computer architectures, making it a favourite of students around the world.

After the successful Oberon Day '93 organized by the Oberon User Group and the Institute for Computer Systems of the ETH Zürich, this year's Oberon Day is dedicated to Oberon in education. Together with the fathers of Oberon, Prof. N. Wirth and Prof. J. Gutknecht, speakers from different backgrounds will talk about their use of Oberon in education. Oberon and its applications will be demonstrated live and attendees will have the possibility of buying the system for a small fee.

The event will take place on the 14th of September 1994, in the Auditorium Maximum of the ETH Zuerich. All talks will be held in German. More details, a program and a registration form can be obtained from the Oberon User Group, or from the Secretary of the SI, Frau A. Nicolet ++41/1/371 73 42. –Markus Dätwyler, Oberon User Group

Welcome to Hades

The Oberon User Group manages an internet FTP server called hades-.ethz.ch (129.132.71.5) on which public domain Oberon software is archived. The directories are organized according to the system in /pub/Oberon, with each system sub–divided again for Oberon V4 and Oberon System 3. A special directory contains programs that have been written for non–ETH Oberon implementations. You may upload your programs to /pub/incoming, from where they will be copied by the system administrators to their correct directory. Don't forget to include a README file in ASCII when uploading. The Oberon User Group cannot accept responsibility for the functionality of Oberon programs on the FTP server, and does not adjust software for newer Oberon versions. –Patrick Saladin, Oberon User Group

Oberon for Amiga™

The Institute for Computer Systems is proud to announce the availability of the Oberon System V4 for Amiga computers. Oberon for Amiga is an implementation of the standard Oberon System Version 4, featuring an Oberon–2 compiler. All Oberon programs written for any Oberon System V4 can be compiled without modification. The software package oberon-.lha is available without fee via anonymous internet ftp from neptune.inf.ethz.ch (129.132.101.33) in directory /pub/Oberon/Amiga/. –Stefan Ludwig, Institute for Computer Systems

Oberon in a Digital Design Course

For the second year, a course in digital design for Computer Science students at ETH was held with great success. During the course, students solve exercises in digital design by implementing a circuit on an FPGA board (Field–Programmable Gate Array), using CAD software written entirely in Oberon. The software package consists of a hardware description language compiler, a

graphical design editor, and a design checker for comparing a formal description of a design with its implementation. The portability of Oberon allowed an easy migration of the tools to various other machines, such that students could solve the exercises on their computers at home, and only needed the FPGA board for final verification of the designs. –Stefan Ludwig, Institute for Computer Systems

Using TrueType™ fonts with Oberon System 3

Tired of having to select a font for your Oberon documents from a very narrow set of choices? Oberon System 3 now supports Apple's TrueType font technology, which is incorporated in such widely used systems as Apple Macintosh or Microsoft Windows. This means that Oberon users may now use TrueType fonts within their documents, on screen as well as on paper.

Unlike traditional Oberon bitmap fonts, which need one file for every point size, TrueType fonts come with only one file holding all necessary information. There's not even a distinction between screen fonts and printer fonts anymore, since a TrueType font does not make a fundamental difference between the two. This allows users to have lots of fonts available at reasonable storage requirements.

Integration into Oberon System 3 is so smooth that most users will hardly ever notice, because all existing bitmap fonts will continue to work as before. The only difference lies in the fact that if a font is requested but cannot be found, the TrueType machinery will look for an appropriate TrueType description instead of immediately returning a default font. The bitmap patterns that are needed to display characters on the screen are generated the first time they are requested and are kept in memory for further uses. This makes sure that computation effort is only spent where it is needed, since no pattern will ever be generated for characters that are never used, and keeps response times short (as long as only short text stretches are affected).

Another way to use TrueType fonts is to convert them to traditional Oberon font files, which is also possible. These can of course be used with Oberon V4, too, and may be edited manually to improve their appearance on the screen.

The use of TrueType fonts within Oberon documents opens a whole new range of possibilities, and we hope that a lot of users will take advantage of this opportunity and use their TrueType fonts with Oberon, too *[Editor's note: The TrueType interpreter is available with the Leda page layout system demo version in each DOS–Oberon release].* –Erich Oswald, Institute for Computer Systems

TCP/IP and Oberon

The growing interest in distributed services using the client/server paradigm has led to the development of a large collection of network protocols in the computer science community. One, TCP/IP, currently dominates the scene as it is already serving as a de facto standard in the world of UNIX. Most remote services including telnet, ftp and www are based on the reliable connections offered by TCP/IP.

The abbreviation TCP/IP indicates that TCP is based on IP, both of which are layers in the Internet layering model. Compared with the OSI reference model, they correspond to the network layer (IP) and the transport layer (TCP). The Internet protocol family contains several other protocols, i.e. arp (address resolution) and icmp (internet correction message), which are used by TCP/IP.

We have implemented the entire Internet protocol family for the client side with the exception of fragmentation of IP packets and dynamic routing. We support two different versions, i.e. one for the standard Oberon sytem and another for Concurrent Oberon. The two versions differ from one another in dealing with asynchronous input from a network. All ported versions of Oberon can rely on a unique TCP/IP interface so network applications are portable. The complete implementation is rather small (only around 15KB). For further information feel free to contact gitsels@inf.ethz.ch. –Martin Gitsels, Institute for Computer Systems

FTP Server for Research Reports

Most of the research reports, dissertations and technical reports of the Institute for Computer Systems are available electronically via Internet FTP from *neptune.inf.ethz.ch* in the */doc* directory. In addition to the text files containing the report abstracts, you will find the reports in Postscript format. Some of the Oberon related reports are listed in the next paragraphs; additional or older reports are listed in the abstracts on the server. Unfortunately we don't sell dissertations directly; these have to be ordered from the person who wrote it, or if it has an ISBN number, directly from your book store. If you don't have electronic access to the technical reports, please write to us and we will send you a copy if available. Please keep in mind that we print only a limited number of copies.

Technical Report 156, Mar. 1991:
R. Griesemer, *On the Linearization of Graphs and Writing Symbol Files*
C. Pfister (ed.), B. Heeb, J. Templ, *Oberon Technical Notes*

Technical Report 198, Jul. 1993:
N. Wirth, *An Extension–Board with an FPGA for Experimental Circuit Design*
S. Ludwig, CL – *An Editor for the CLi6000 Field Programmable Gate Array and its Implementation*
S. Ludwig, *CL–Editor User Manual*

Technical Report 212, Feb. 1994:
J. Supcik, *HP–Oberon(TM): The Oberon Implementation for Hewlett–Packard Apollo 9000 Series 700*

Technical Report 215, May 1994:
H. Eberle, S. Gehring, S. Ludwig, N. Wirth, *Tools for Digital Circuit Design using FPGAs*

–Johannes Marais, Institute for Computer System

Literature

Several books have been written about the Oberon System and Lan-

guage. We recommend these books for serious Oberon users. However, if you want to try out Oberon before buying a book, most Oberon releases have enough online information to get a new user started with Oberon.

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.

M. Reiser: The Oberon System. User Guide and Programmer's Manual. Addison Wesley, 1991, ISBN 0–201–54422–9.
User manual for the programming environment and reference for the standard module library.

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.

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.

How to get Oberon

All the different Oberon versions are available free of charge from our Internet FTP server *neptune.inf.ethz.ch* in the /pub/Oberon directory. The sub–directory *System3* contains the Oberon System 3 versions. Classic Oberon is available for Amiga, DEC-Station, MS–DOS, Microsoft Windows and Windows NT, HP 700, Mac II, IBM RS6000, SPARC and Silicon Graphics machines. Oberon System 3 Version 1.5 is currently only available for MS–DOS and SPARC computers.

If you do not have access to Internet, you can order diskettes from the address below. We charge a fee of Sfr 50.00 to cover our costs. We accept payment via *Eurocard/Mastercard* or *VISA*. To order by credit card, specify your credit card number, expiration date, and your name exactly as it appears on the card.

If you already purchased an Oberon version from us, we will upgrade you to a newer version for Sfr. 20. The upgrading policy applies only to versions of the same architecture; this means you cannot upgrade from an older Oberon V4 for Windows version to a newer DOS–Oberon version or vice–versa.

How to contact us

Institut für Computersysteme
ETH Zentrum
CH–8092 Zürich
Switzerland

Telephone +41 (1) 6327311
Fax +41(1) 261 53 89 until 12 Aug 94
Fax +41(1) 632 12 20 after 12 Aug 94
e–mail oberon@inf.ethz.ch

The Oberon User Group
Bergstrasse 5
CH–8044 Zürich
Switzerland
Hotline: +41(1) 632 72 13
e–mail oberon–user@inf.ethz.ch

The following page contains a typical Oberon System 3 display snapshot. The lefthand area is the so–called Oberon desktop showing two documents, the backdrop generator panel and the online tutorial system. On the righthand you see a typical tool with a set of useful menus and buttons floating in the text. Below the tool we have opened a tool which allows you to visualize the grid–fitting of TrueType fonts. This tool is included in the TrueType application package of DOS–Oberon System 3 and allows you to observe in a step–by–step manner how a TrueType character comes to life.

Acknowledgements:

Big thanks to our contributors, and to Hans Eberle, Dominique Lebegue, and Stefan Ludwig for their effort in proof reading the newsletter.