

Oberon: The Power of Purity

Jürg Gutknecht
ETH Zürich
March 2004

清淨

- Purity
 - A quantitative assessment of homogeneity or uniformity (The American Heritage Dictionary)
 - Being undiluted or unmixed with extraneous material (Hyperdictionary)
- Impurity
 - Lack of consistency or homogeneity
 - Worthless material that should be removed

Eighteen Years After

- How it All Started
 - Sabbatical at Xerox PARC
 - 10 MHz/2 MB/40 MB Hardware
- Lasting Impressions
 - Instantaneous Responses
 - Superfast Compilation
 - No System Crashes
 - Effective Textual User Interface
 - Gadgets Component Showcase

This Presentation ...

- Focuses on a *tool* for the support of software design and implementation, in contrast to a *product* or *result*
- Shows how the philosophy and design principles underlying this tool guarantee effectivity, versatility and power
- Aims at encouraging non-computer-scientists to use the tool for problem solving
- Neither intends to evangelize nor to claim cure-all properties of the presented tool

A Quotation

„The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities“

Edsger W. Dijkstra



DAVID NOTON - Tread Boldly

like this ...

... or like this?



The Tool ...

- Is a programming language called *Oberon*, a carefully crafted consolidation of *Pascal* and *Modula-2* after a long period of working experience
- Uses a simple, clear and expressive syntax
- Provides a set of effective, transparent and resource-efficient constructs
- Propagates a uniform, non-restrictive and consistent computing model

ETH Language Tradition

Year	Language	Influential Constructs
1960	Algol	Procedures
1970	Pascal	Data Structures
1980	Modula(-2)	Modules
1990	Oberon	Type Extension
2000	Active Oberon	Active Objects
2004	Zonnon	Composition & Dialogs

Some Examples

Included Constructs	Excluded Constructs
Modules	Nested Modules
VAR parameters	Cardinals
Small sets	Large sets
Procedure variables	Subranges
Modules	Nested Modules
Pointers	Implementation inheritance
Type Extension	Variant Records

Random Number Generator

```
MODULE RandomNumbers;  
  VAR z: LONGINT; (*global variable, hidden*)  
  PROCEDURE Next*(): REAL; (*interface*)  
    CONST a = 16807; m = 2147483647;  
      q = m DIV a; r = m MOD a;  
    VAR g: LONGINT;  
  BEGIN (*implementation*)  
    g := a*(z MOD q) - r*(z DIV q);  
    IF g > 0 THEN z := g ELSE z := g + m END;  
    RETURN z*(1.0/m) (*value*)  
  END Next;  
BEGIN z := 31459  
END RandomNumbers.
```

Tree Insertion

```
Node = POINTER TO RECORD
  left, right: Node;
  key: INTEGER
END;

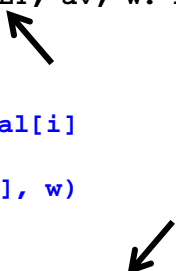
PROCEDURE Insert
  (VAR r: Node; new: Node);
BEGIN
  IF r = NIL THEN r := new
  ELSIF new.key <= r.key THEN
    Insert(r.left, new)
  ELSIF new.key > r.key THEN
    Insert(r.right, new)
  END
END Insert;
```

```
// Java
class Node {
  int key;
  BinaryTree left, right;
  public Node(int key) {
    this.key = key;
    left = new BinaryTree();
    right = new BinaryTree();
  }
}

class BinaryTree {
  Node root;
  void insert (Node node) {
    if ( root == null )
      root = node;
    else if ( node.key <=
      root.key)
      root.left.insert(node);
    else
      root.right.insert(node);
  }
}
```

The Rucksack Problem

```
PROCEDURE Try (i: INTEGER; s: SET; av, w: REAL);
BEGIN
  IF av - val[i] > vmax THEN
    IF i = n-1 THEN
      opts := s; vmax := av - val[i]
    END
    ELSE Try(i+1, s, av - val[i], w)
  END
END;
IF w + weight[i] <= wmax THEN INCL(s, i);
  IF i = n-1 THEN
    IF av > vmax THEN opts := s; vmax := av END
  ELSE Try(i+1, s, av, w + weight[i])
  END
END;
END Try;
```



Machine Code

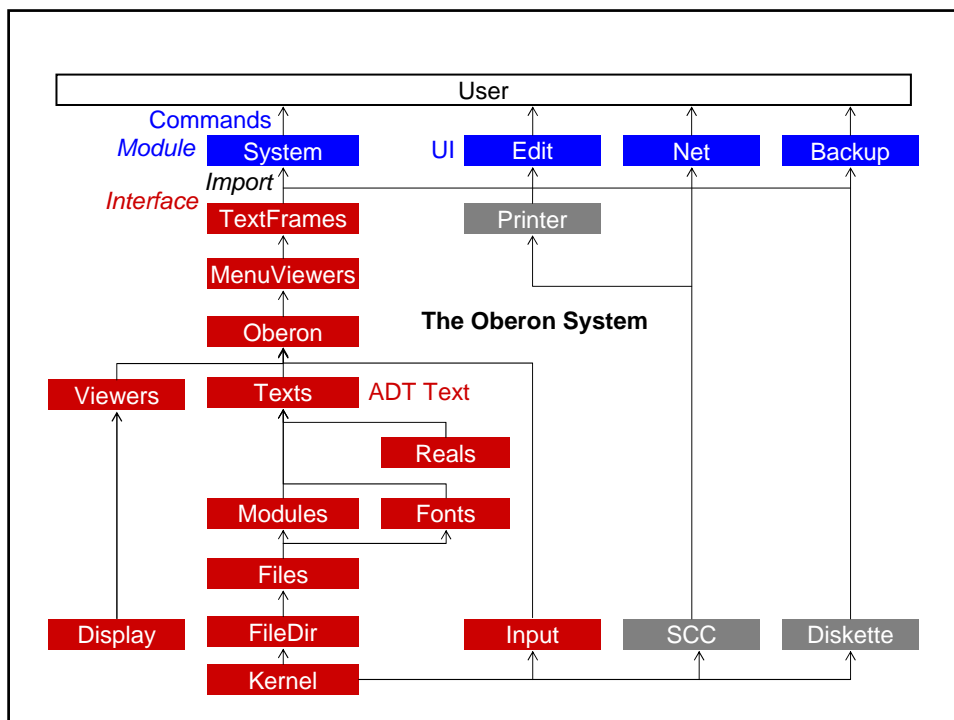
<code>INCL(s, i)</code>		<code>Include Element i</code>
<code>MOV EBX, 20[EBP]</code>		<code>Load i from frame</code>
<code>BTS 16[EBP], EBX</code>		<code>Set bit i in s</code>

Key Virtues

- Full Support for Strong Typing
 - Safeguard against corruption & viruses
 - Precondition of automated garbage collection
- Full Support for All Paradigms
 - Algorithms and Data Structures
 - Modular Programming
 - Object-Oriented Programming
 - Component Modeling

Modules, Interfaces and Import

- Multiple Roles of Modules
 - System Managed Object with Interface
 - Container of Related Data Types
 - Provider of Abstract Data Types (ADT)
 - Unit of Separate Compilation
 - Unit of Loading on Demand
- Import Relation for Modules
 - Supports Separation of Concerns
 - Specifies Static System Structure
 - Reflects Dependencies



Text as an Abstract Data Type

- Integrated in Core System
- Accessed by ADT *Reader, Scanner, Writer*
- Consumed and Produced by Programs
- Parsed by Commands
- Used for Powerful Textual User Interface

Object Spectrum

Role	Level	Ingredients	Model
Operand	Record	Data Fields	A&D
Servant	+ Functional	+ Methods	OOP
Actor	+ Behavior	+ Activities	Actor

Focused by OO Languages

Simulation Engine

```
PROCEDURE MyEventHandler (e: Event);  
BEGIN ...  
END MyEventHandler;
```

```
TYPE  
  Event = POINTER TO RECORD  
    next: Event;  
    t: REAL;  
    handle: PROCEDURE (this: Event)  
  END;  
  
Simulate(simPeriod) {  
  GetNext(e);  
  WHILE e.t <= simPeriod DO  
    now := e.t; e.handle(e)  
    GetNext(e)  
  END
```

e.handle := MyEventHandler

Eventhandler

"Make it as Simple as Possible..."

Quotation A. Einstein

```
TYPE  
  Message = RECORD END;  
  Object = POINTER TO RECORD  
    f: PROCEDURE (me: Object; msg: Message);  
  END;  
  
(*method call*)  
... x.f(me, myMessage);
```

Type Extension Applied Twice

```
TYPE
  OpMessage = RECORD (Message)
    op: INTEGER
  END;

  TextMessage = RECORD (Message)
    text: ARRAY 100 OF CHAR
  END;

  MyObject = POINTER TO RECORD (Object)
    state: INTEGER
  END;
```

Runtime Dispatching Handler

```
PROCEDURE myf (x: Obj; m: Message);
BEGIN
  IF m IS OpMessage THEN
    CASE m(OpMessage).op OF ... END
  ELSIF m IS TextMessage THEN
    (*parse m(TextMessage).text*)
  ELSE (*unknown message received*)
  END
END myf;

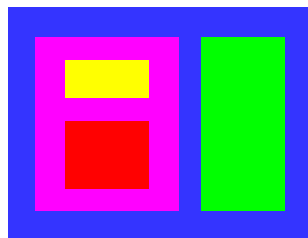
VAR x: MyObject;
... NEW(x); x.f := myf; ...
```

Instance-centered
coupling of functionality

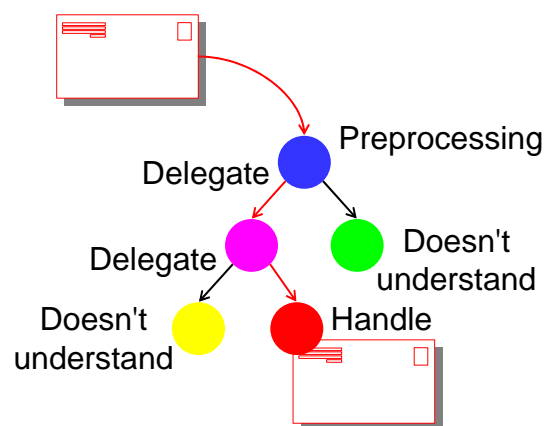
Method Call Machine Code

x.f(x, m)	Method call
PUSH [-4]	x
PUSH [12]	adr(m)
PUSH -104	tag(m)
MOV EBX, [-4]	x
CALL 0[EBX]	x.f
PROCEDURE myf	Entry protocol
PUSH EBP	Save old stack frame
MOV EBP,ESP	New fp is top of stack
END	Exit protocol
MOV ESP,EBP	Restore fp
POP EBP	
RET 12	Remove pars and return

Component Hierarchy



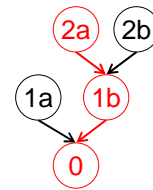
**Container
Hierarchy**



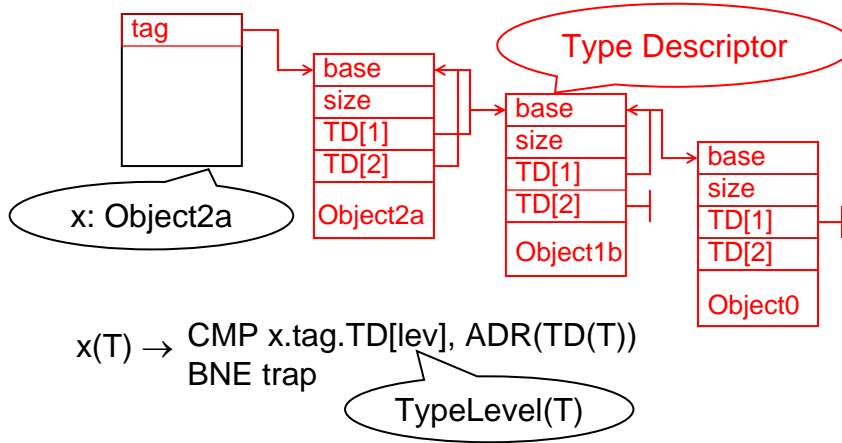
**Parental Control
Design Pattern**

... But Not Simpler"

Quotation A. Einstein



- Oberon Runtime Type Support



Runtime Typing Machine Code

IF m IS OpMessage THEN	Type test
MOV EBX,12[EBP]	Type tag
MOV EBX,-12[EBX]	Extension level
CMP EBX,[8]	OpMessage descriptor
JNZ 53 (00000086H)	If negative
m(OpMessage).op	Type guard
MOV EBX,12[EBP]	
MOV EBX,-12[EBX]	
CMP EBX,[8]	
JZ 3 (00000062H)	
PUSH 6	Trap no
INT 3	Trap

Outlook: Active Objects



```

TYPE
  Particle = POINTER TO RECORD
    col: INTEGER; t, dt: REAL;
    x, k0, k1, k2, k3, q: Vector;
    f: PROCEDURE (me: Particle; x: Vector): Vector;
  BEGIN { ACTIVITY }
    LOOP Flip(x[0], x[1], col);
      PASSIVATE dt*10000;
      Flip(x[0], x[1], col); Draw(col, x[0], x[1]);
      k0 := f(x);
      k1 := f(x + dt/2 * k0);
      k2 := f(x + dt/2 * k1);
      k3 := f(x + dt * k2);
      x := x + dt * (1/6*k0 + 1/3*k1 + 1/3*k2 + 1/6*k3);
      t := t + dt
    END
  END Particle;
  
```

Intrinsic Behavior

Outlook: Active Objects

```

TYPE
  LorenzParticle = POINTER TO RECORD (Particle);
    sigma, r, b: REAL
  END LorenzParticle;

  PROCEDURE Lorenzf (me: Particle; x: Vector): Vector;
    VAR y: Dynamics.Vector;
  BEGIN
    y[0] := me(LorenzParticle).sigma*(x[1]-x[0]);
    y[1] := -x[0]*x[2] + r*x[0] - x[1];
    y[2] := x[0]*x[1] - b*x[2];
    RETURN y
  END Lorenzf;
  
```

Conclusion

- Oberon and its successors are quite successful in the hands of specially skilled software constructors.
- The common programming paradigm has shifted in the meantime from full custom algorithm & data development to extensive reuse of libraries
- Commercial languages have taken up many of the virtues discussed, for example, linking loader technology and fully managed runtimes
- Moore's law has partly spoiled the show of resource efficiency. A decrease of response time from 0.1 sec to 0.01 sec is far less dramatic than a decrease from 1 sec to 0.1 sec. A new chance opens in the area of the "disappearing computer"

Conclusion Continued

- Modern systems have reached a state of "unmastered complexity. Calls for simplicity are en vogue. Bill Gates quoting Mark Twain at the 1998 Microsoft company meeting:

I am sorry that I wrote such a long letter,
I didn't have time to write a shorter one
Mark Twain in a letter to a friend

- The most effective key to simplicity is the art of uniforming what can be uniformed and separating what should be separated
- The question remains what a complex system can do that Oberon cannot.