# Chronological Overview

## ETHZ Language/ Compiler Development

<table>
<thead>
<tr>
<th>Year</th>
<th>Language</th>
<th>Project/ Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1970</td>
<td>Pascal</td>
<td>Data Structures</td>
</tr>
<tr>
<td>1980</td>
<td>Modula(-2)</td>
<td>Modules</td>
</tr>
<tr>
<td>1990</td>
<td>Oberon</td>
<td>Type Extension</td>
</tr>
<tr>
<td>1999</td>
<td>Devlab Redmond</td>
<td>Oberon0 for .NET</td>
</tr>
<tr>
<td>2000</td>
<td>MSDN Orlando</td>
<td>Oberon for .NET</td>
</tr>
<tr>
<td>1990</td>
<td>Active Oberon</td>
<td>New Object Model</td>
</tr>
<tr>
<td>2001</td>
<td>PLI/Babel Florence</td>
<td>Active Oberon for .NET</td>
</tr>
<tr>
<td>2001</td>
<td>MSDN Los Angeles</td>
<td>Active Oberon for .NET</td>
</tr>
</tbody>
</table>
Project 7/7+ With MS Research

Project Tasks

- Implement Academic Language for .NET Runtime
- Integrate Academic Language into Visual Studio
- Demonstrate Language Interoperability on .NET
- Provide Feedback to .NET Architects

Project Opportunities

- Increase Popularity of Academic Language
- Get New Teaching Vehicle and Topic
- Get Advanced Platform
  - For Compiler Experiments
  - For Language Research
.NET Environment

- "Managed" Code and Data at Runtime
- Common Language Runtime *CLR*
- Virtual Object Model *CTS*
- Reflection and Reflection Emit Tools
- Intermediate Language and Assembler MSIL
- Compiler Integration Tool *CCI*
- Rich GUI API *WinForms*
- Rich Web API *WebForms*
- Development Environment *VS*

Topic of Companion Lecture
## Oberon for .NET Language

### Mapping Oberon to CTS

<table>
<thead>
<tr>
<th>POINTER, RECORD Pair</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORD Type</td>
<td>Value Class</td>
</tr>
<tr>
<td>Type-Extension</td>
<td>Class Extension</td>
</tr>
<tr>
<td>Module</td>
<td>Sealed „Static” Class</td>
</tr>
<tr>
<td>Import</td>
<td>Loading Code in Constructor</td>
</tr>
<tr>
<td>Exported Item</td>
<td>Public Item</td>
</tr>
<tr>
<td>Procedure</td>
<td>Method</td>
</tr>
<tr>
<td>Procedure Variable</td>
<td>Function Pointer</td>
</tr>
</tbody>
</table>

*Static Fields & Methods Only*
Mapping Oberon Modules (1)

```oberon-mapping
MODULE M;
IMPORT N;
TYPE
  X* = POINTER TO RECORD
    VAR t*: T; u: U; ...
  END X;
  Y = RECORD ... END Y;
VAR a: A; b*: B;
PROCEDURE P* (t: T): X;
BEGIN ...
END P;
PROCEDURE Q (i: INTEGER);
BEGIN ... N...
END Q;
BEGIN (* module initialization *) ...
END M.
```
Mapping Oberon Modules (2)

Assembly → M
PE Module → M
Namespace → M

X

Y

Module Initialization in Class Constructor

Class
Value Class
Sealed Static Class
Oberon for .NET Compiler (1)

One Pass Recursive Descent Strategy
- One Parse Procedure for Each EBNF Production
- Parser Invariantly Looks Ahead One Symbol and Recursively Calls Corresponding Parse Procedure
- Parser Additionally Performs Semantic Functions
  - Internalize Declarations into Symbol Table
  - Type Check Statements and Generate Code

Interoperability
- Use Heuristics to Locate Assemblies
- Use Reflection API for Consumed Entities
- Generate MSIL Text and Run Assembler

Alternatively Use Reflection Emit API
Oberon for .NET Compiler (2)

Compilation Process in a Nutshell

VAR a, b, c: REAL;
...
IF c > a + b THEN

ParseIfStat
ParseExpr

Transient „Item”

LDLOC c
LDLOC a
LDLOC b
ADD
CMP
BLE ...

IL Code

Symbol Table

REAL
BOOL
CHAR
Oberon for .NET Compiler (3)

Modular Structure

Parser & Symbol Table
- 3200 Source Lines
- 82 KB PE Code

Code Generator
- 1800 Source Lines
- 53 KB PE Code

Symbol Scanner
- 400 Source Lines
- 20 KB PE Code
Oberon for .NET Compiler (4)

Construction Strategy

- Define Language Subset Oberon-0 From
  N. Wirth, Compiler Construction, Addison-Wesley 1996
- Implement Oberon-0 as X-Compiler
- Port Oberon0 for .NET X-Compiler to .NET
- Extend to Oberon for .NET Compiler
- Apply Self-Compiling Test
- Extend to Oberon’ for .NET
- Extend to Oberon” for .NET
- ...

Interop Adjustments
Language Evolution
Oberon for .NET Compiler (5)

X-compilation and Run

Oberon-0

Oberon-0 Compiler

IL

PE

Portable Executable

Oberon-0

Win .NET Runtime

JIT Compiler

PE

Ro

Rw

Oberon-0

IL

Oberon Runtime

Win .NET Runtime

Portable Runtime
Oberon for .NET Compiler (6)

Compiler Bootstrapping on .NET
Interoperability Requirements

.NET Language

- Number Formats
- Char Codes
- Enumerations
- Overloading
- Exceptions

- Number Formats
- Char Codes
- Multiple Base Classes
- ...

Must Produce to Subset

Must Consume from Superset

CLR
Oberon for .NET Compiler (7)

Compiler in Its Current State
- Essentially Implements Oberon Language
- Passed Self-compiling Test on .NET
- Accepts Ordinary Text Files
- Is Launched From Command Line
- Takes Assembly Locations from Command Line
- Produces Textual MSIL Code
- Jits to ca. 50% Faster Code than Native Oberon
- Has a Size of 5400 Source Lines/ 155 KB PE Code
- Is Used as Exercise Platform for First Year Teaching of Programming at ETHZ (Ca. 350 Students)
- Is Downloadable via http://www.oberon.ethz.ch
"Zonnon" Language Evolution

- Interoperability Adjustments etc.
  - Unified Modifier Concept \(\{ Modifier \}\)
  - Simple Exception Mechanism
  - Block Statement
    - BEGIN \{ Modifier \}
    - ON excpX DO ... ON excpY DO ...
    - END
  - Predefined \(\text{PAR, EXCL, ACT}\)
- Enumeration Types \(Color = (\text{Red, Green, Blue})\)

- New Object Model
  - Objects With Thread of Activity \(Active\ Object\s\)
  - Unified Abstraction Mechanism \(Definition\)
Active Objects (1)

Classical Computing Model
Processes Operating on Objects

Process P
Call
Synchronize

Object x
Object s
Object z

Shared Resource

Process Q
Active Objects (2)

The New Computing Model
Robin Milner
Active Objects (3)

OBJECT X;
VAR t: T;
PROCEDURE p (u: U; VAR v: V);
BEGIN { EXCLUSIVE } ...
END p;
PROCEDURE q;
BEGIN condition := TRUE
END q;
BEGIN { ACTIVE }...
BEGIN { EXCLUSIVE }...
AWAIT condition;...
END
END Z;

State
Methods
 „Behavior“

Under Mutual Exclusion
Separate Thread Precondition of Continuation
Active Objects (4)

Movie = OBJECT
VAR first, cur: Picture;
   X, Y: REAL; stopped: BOOLEAN;
INITIALIZER (orgx, orgy: REAL; video: Picture);
BEGIN X := orgx; Y := orgy; first := video
END;
PROCEDURE Stop;
BEGIN stopped := TRUE
END Stop;
BEGIN { ACTIVE }
stopped := FALSE; cur := first;
WHILE ~stopped DO
   cur.Display(X, Y); cur := cur.next (*ring*)
END
END Movie;
Definitions (1)

Conventional Concepts of Abstraction

Base Class Abstraction

`C` "Extends" `C0` Base Class

`C` Implements `I0`, `I1`, ...

Interface Abstraction

`C` Implements `I0`, `I1`, ...

`I0`, `I1`, ... Interfaces
Definitions (2)

Unified Concept of Abstraction

"Facette"

Custom Implementation

Default Implementation

Definition

"Aggregation"

Client Object

Server Object as Composition of Facettes

Client Object

Client Object
Definitions (3)

What Is It (Primarily)?

- **JukeBox: Player or Store?**
  ```java
class JukeBox extends Player implements Store
class JukeBox extends Store implements Player
  ```

- **Sprite: Animation or Thread?**
  ```java
class Sprite extends Figure implements Runnable
class Sprite extends Threads implements Animation
  ```

- **GUIPanel: Applet or Container?**
  ```java
class GUIPanel extends Applet implements Container
class GUIPanel extends Container implements Applet
  ```
**Zonnon** Program Structure (1)

### Basic Program Units

<table>
<thead>
<tr>
<th>Unit</th>
<th>Semantics</th>
<th>Instantiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Interface defining signatures of methods and state variables</td>
<td>None</td>
</tr>
<tr>
<td>Implementation</td>
<td>Generic (partial) implementation of corresponding definition</td>
<td>None</td>
</tr>
<tr>
<td>Module</td>
<td>System managed object</td>
<td>By system</td>
</tr>
<tr>
<td>Object</td>
<td>Program managed object type</td>
<td>By program</td>
</tr>
</tbody>
</table>

### Source Code Packaging

- Implicit Namespaces \(A.B.\) ...
IMPLEMENTATION My.Store;
VAR rep*: Lib.Song;
PROC Clear;
BEGIN rep := NIL
END Clear;
PROC Add (s: Lib.Song);
BEGIN s.next := rep; rep := s
END Add;
BEGIN Clear
END Player.

OBJECT My.JukeBox IMPLEMENTS My.Player, My.Store;
Mapping „Zonnon” to .NET (1)

Mapping Active Objects

- A Object Type with active body

**Mapping Method „Mass Notification”**

Derive A from class Thread in `System.Threading`

<table>
<thead>
<tr>
<th>Source Construct</th>
<th>Target Construct in C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN { ACT } S END</td>
<td>start() { S };</td>
</tr>
<tr>
<td></td>
<td>...; this.start()</td>
</tr>
<tr>
<td>AWAIT c</td>
<td>while !c { Monitor.Wait(this); }</td>
</tr>
<tr>
<td>BEGIN { EXCL } S END</td>
<td>Monitor.Enter(this); S; Monitor.PulseAll(this); Monitor.Exit(this);</td>
</tr>
</tbody>
</table>
Mapping „Zonnon” to .NET (2)

Mapping Definitions

- D Definition
- I_D Corresponding Generic Implementation
- O_D Implementing Object Type

Mapping Method „Source Aggregation”
Map D, I_D to „Symbol File” S_D
Aggregate SD when compiling O_D

Mapping Method „Binary Aggregation”
Map D, I_D to interface ID and class SD
Let O_D implement ID and delegate to SD

Mapping Method „Inheritance”
Map D, I_D to abstract class CD
Derive O_D from CD by extension

for Optimization and Consumed Foreign Classes
Mapping „Zonnon” to .NET (3)

C#:

```csharp
public interface ID {
    T x { get; set; }
    void f(T y); /* impl by SD.f */
    void g(T y); }

public class SD {
    public static void f(D me, T y)
    { T z; me.x = y; z = me.x; } }

sealed class CD: ID {
    T D_x;
    public void g(T y) {
        T z; z = x; x = y;
    }
    public void f(T y) {
        SD.f(this, y); /* delegate */
    }
    public T x {
        get { return D_x; }
        set { D_x = value; }
    }
}
```

Zonnon:

```zoonon
DEFINITION D;  
VAR x: T;
PROCEDURE f (y: T);
PROCEDURE g(): T;
END D;

IMPLEMENTATION D;
PROCEDURE f (y: T);
    VAR z: T;
    BEGIN x := y ; z := x
END f;
END D;

OBJECT C IMPLEMENTS D;
PROCEDURE g (y: T)
    IMPLEMENTS D.g;
    VAR z: T;
    BEGIN z := x; x := y
END g;
END C;
```