Constructive Computer Science at ETH Zuerich

Jürg Gutknecht, Prof. ETH Zürich, November 2025

While Computer Science at ETH exhibits a large variety of facettes from formal theory to systems design, digital security and teaching methodology, to name just a few, the discipline of systems construction has become one of its landmarks from very early on. In this short report I will focus on this discipline, its history and impact on both academia and industry,

First Constructive Phase

Computer Science at ETH and in particular the Department of Informatics grew out of Numerical Mathematics in the 1950s. In 1950 a member at the renowed « Institute of Applied Mathematics », professor Eduard Stiefel and staff decided to acquire on loan one of the very first computers worldwide, a German *Zuse Z4*. No real programming language was available at this time just something called « Plankalkül» instead. As a proof of concept, Stiefel used Plankalkül to demonstrate the Z4´s power by a calculation of the Euler number «e» up to a very large number of positions.

Soon after, however, Stiefel and his colleague Heinz Rutishauser came up with the idea of building their own computer, which they did and called it *ERMETH*, see Figure 1. ERMETH became operational in 1957. Its relais based architecture was composed of roughly 1500 tubes and diodes with an attached disk storage of a capacity of 10000 words anda weight of 1.5 tons. This time, the institute's proof of concept was computing « pi ». In 1958 an intercontinental team of scientists gathered at ETH with the aim of developing a more advanced programming language called *Algol* (for algorithmic language). The first result was Algol 58, followed by Algol 60. Although strongly geared towards mathematical applications, Algol introduced some key programming concepts like structured programming, recursive functions, and a meta language called BNF (Backus Naur form) for formally defining syntaxes.

Fig. 1. The Institute of Applied Mathemathics and its ERMETH computer

After Algol 60 the path of language design split in two. On the one hand e Algol oup continued the Algol development up to a version Algol68 that was not received well because of its excessive complexity and clumsiness. On the other hand, a young engineer started to develop a

light weight version of Algol, first called Algol W and then *Pascal*. The name of this engineer was Niklaus Wirth, and Pascal was finally released together with a virtual *P-code* machine acting as a frontend target for compilers. This was a significant innovation at the time that (1.) simplified any compiler writer's life and (2.) made compiler frontends inherently portable. As a consequence, a number of Pascal compilers like *UCSD Pascal* from the University of California at San Diego became quickly available, and Pascal spreaded world-wide. In 1970 Wirth won the prestigious Turing Award.

The P-code embodies a model of a simple stack oriented architecture that releives compiler frontends from tasks like register management, memory management, and the like. Here is an example of an arithmetic expression and its translation into P-code :

- Arithmetic expression (a + b*(x – y) / b
- P-code LD a LD b LD x LD y - * + LD b /
  where LD means loading a value onto the stack

However, Pascal compilers are not just comparatively simple pieces of software, they are strong in terms of early recognition of programming errors, mainly due to Pascal's « strong typing » principle that does not allow data of one type to be treated as data of another type.

Second Constructive Phase

The second and up to now last constructive phase of Computer Science at ETH started in 1980 after Niklaus Wirth returned from a sabbatical at the famous research lab *Xerox PARC* in California. It was the time when « personal computer », if at all, was identified with keyboard and a 24 x 80 screen as its primitive user interface. In Palo Alto, Wirth learnt about the Xerox *Alto* computer, the first representative of a radically new generation of personal workstations. Bitmap based displays and « mouselike « input devices meant a substantial upgrade of the user interface, and a significantly increased level of computing power allowed for totally new applications like advanced text processing, graphic designs, and photo editing, to name just a few. The underlying motto was *WYSIWYG*, « what you see is what you get ».

In any case, Wirth was fascinated of the new generation of computing workstations and he became eager to build one of his own, however, following his trademark, at just a fraction of the Alto's complexity. The final result of this endeavor was *Lilith*, the first personal workstation of its kind in Europe. Several editions of ever smaller versions of the Lilith were finally built, see Figure 2. What was Wirth's and his team's scientific approach?

Fig. 2. The Lilith workstation (final version)

First, both the Pascal language and the associated P-code had to be upgraded carefully from a language for specifying algorithms to a language for specifying systems, or, put differently from a language for programming in the small to programming in the large, while preserving most of Pascal's characteristics. The main upgrade of the new language called *Modula* and later *Modula-2* against Pascal was the module construct. Modules are building blocks of a software system. They come in pairs of *implementation* and *definition*, where the latters are subject to be imported from other modules.

Second, the P-code was upgraded to an *M-code* that, as a technical innovation, was microcoded in hardware, more precisely in 16 bit AMD 2901 processor hardware. Thanks to this implementation of the M-code directly in hardware, any kind of a «low level» system software cushion between hardware and application software, typically implemented in an unsafe assembler language was abandoned with the Lilith's system design. In the end the entire software from the operating system (with name Medos) and device drivers all the way up to applications and user interface presented itself quite uniformly as one homogeneous network of Modula-2 modules, see Figure 3.

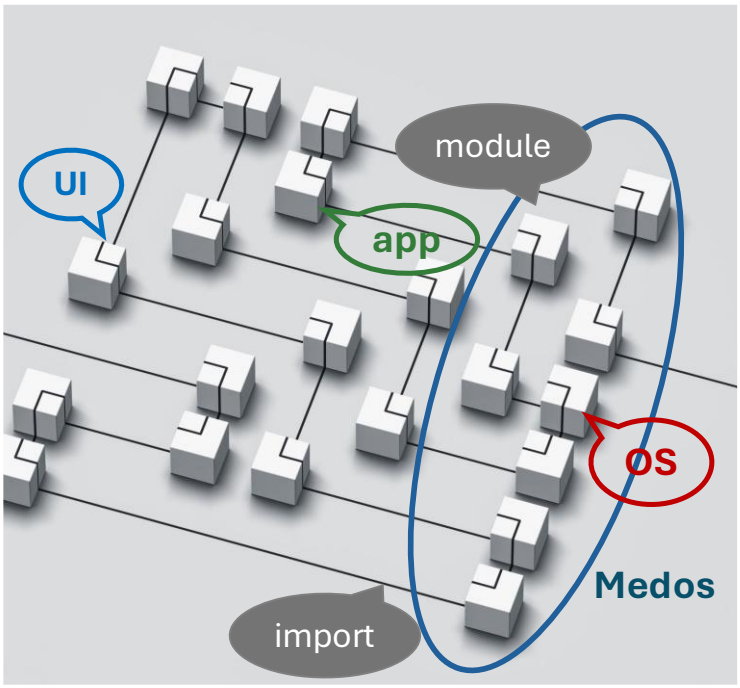

Fig. 3. The homogeneous Modula-2 Software System

Thanks to the M-code, writing Modula-2 compilers was a relatively easy exercise. Nevertheless, it had to be done, and it was done in two steps: (1.) generating a Modula-2 crosscompiler running on a different platform, and (2.) porting the crosscompiler to the Lilith platform. Figure 4 illustrates these two steps in terms of so-called «*Tombstone diagrams*» also called «*T diagrams*». Notably the first T diagram in the devopment of the cross compiler represents an existing Pascal compiler running on a DEC PDP-1 computer, while the final diagram in the porting series allows the «Wirth test» to be performed, namely compiling the compiler with itself and comparing the two involved object codes (the two codes should be equal).

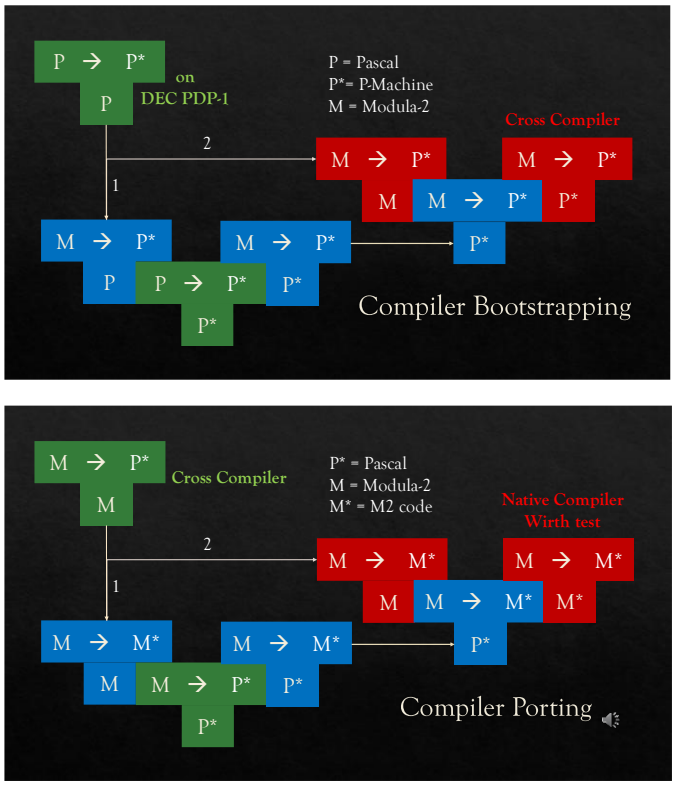

Fig. 4. The two-phased Modula-2 Compiler Construction

Modula-2 was made available on several platforms, and it was used internationally in a diversity of mostly safety-critical applications. However, it did not match Pascal in its comprehensive worldwide spread. Also, an attempt to commercialize Lilith by a company called *DISER* (Digital Image and Sound) was not successful. The commercial time was simply outrunning academia.

However, the academic time was not stopping either. Both hardware and software developed quickly. Hardware according to Gordon Moore's law stating that (simplified a bit) hardware performance doubles every two years. (Looking back from today this law cannot be confirmed in terms of processor performance that drags behind but has significantly been outdone in terms of memory capacity.) In contrast, software started to slow down mostly because of increasing complexity such as object-oriented programming, a situation bearing a sarcasting slogan « Hardware is getting faster slower than software is getting slower ».

In the year1986 the Wirth group decided to replace the outdated pair (Lilith, Modula-2) with a modernized pair *(Ceres, Oberon)*, see Figure 5. Ceres was put on a 32 bit NS2032 processor architecture replacing the bitsliced 16 bit AMD 2901 architecture used for Lilith. A noticable progress languagewise from Modula-2 to Oberon was the introduction of a very lightweight version of object-orientation called *type-extension*. Type-extension allows new structures (record types) to be derived from existing ones, without giving up strong type-checking at compile time.For example,

```
TYPE
    T = RECORD a : ... ; b : c : ... END ;
    T1 = RECORD(T) c : ,,, ; d: ... END ;
```

in Oberon means that type T1 is derived (« extended ») from type T, inheriting all its infrastructure and being type-kompatible with T.

Fig. 5. Niklaus Wirth and the Ceres Workstation

As a proof of concept, type extension was heavily used and tested in a project for developing a modern graphical user interface (GUI). The outcome was *Gadgets*, a framework that would not have to hide itself still today, s. Figure 6. In Gadgets, each component like « menu », « textfield », « slider», «checkbox» etc. is derived via type extension from a base type *gadget*. The essential point here is that, thanks to type extension, new components can smoothly be integrated into the Gadgets system at any time. As a matter of fact, new GUI components were developed and neatly integrated into Gadgets years after its development.

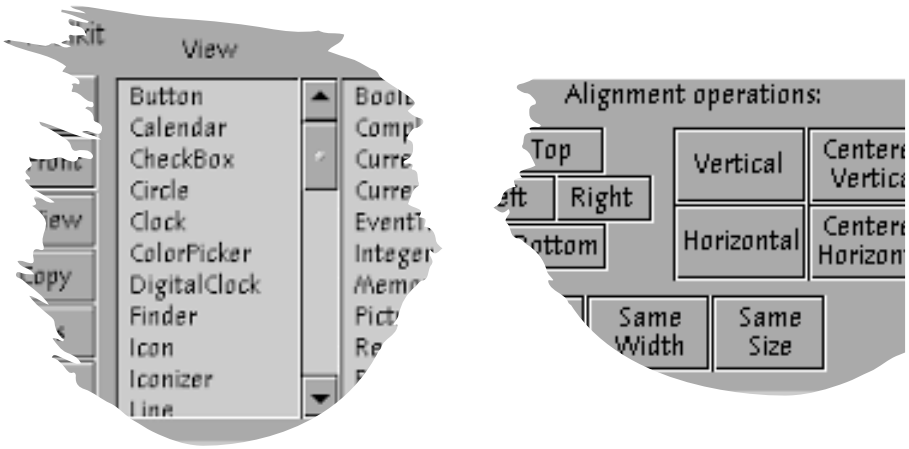

Fig. 6. The Gadgets GUI Framework

A final and last step towards integrated system construction concluding the second phase is based on FPGA (Field Programmable Gate Arrays) technology. Wirth developed a hardware description language called *Lola* with a syntax and style along the lines of Modula.2 and Oberon but with a completely different semantics.

For example, the Lola excerpt below describes no actions but rather a composition of hardware elements such as bits an registers :

```
TYPE Counter(N) ;
VAR
    ci : BIT ; co : BIT ;
    c : [N]  BIT ; q [N] BIT ;
BEGIN
    q.0 := REG(q.0 - ci) ; c.0 := q.0 * ci ;
    FOR i := 1.. N-1 DO
        q.i := REG(q.i – c[i – 1]) ; c.i := q.0 * c[i – 1]
    END ;
    co : = c[N – 1]
END  Counter
```

The Lola language was used in student labs and, quite recently, by a developer outside of ETH for recreating the Ceres hardware.

Summary

We can identify two phases of constructive computer science at ETH, the first phase growing out of applied mathematics in around the 1950s and 1960s, and a second phase representing a pioneering effort towards making personal workstations and their concept known and available in Europe. In both phases the development of simple, safe and powerful programming languages attracting an international community of users was a principal issue and result.