CHAPTER 1

HISTORY AND MOTIVATION

The most important thing in the programming language is the name. A language will not succeed without a good name. I have recently invented a very good name and now I am looking for a suitable language.

— Donald Knuth (19??)

How could anyone diligently concentrate on his work on an afternoon with such warmth, splendid sunshine, and blue sky. This rhetorical question was one I asked many times while spending a sabbatical leave in California in 1985. Back home everyone would feel compelled to profit from the sunny spells to enjoy life at leisure in the country-side, wandering or engaging in one's favourite sport. But here, every day was like that, and giving in to such temptations would have meant the end of all work. And, had I not chosen this location in the world because of its inviting, enjoyable climate?

Fortunately, my work was also enticing, making it easier to buckle down. I had the privilege of sitting in front of the most advanced and powerful work-station anywhere, learning the secrets of perhaps the newest fad in our fast developing trade, pushing colored rectangles from one place of the screen to another. This all had to happen under strict observance of rules imposed by physical laws and by the newest technology. Fortunately, the advanced computer would complain immediately if such a rule was violated, it was a rule checker and acted like your big brother, preventing you from making steps towards disaster. And it did what would have been impossible for oneself, keeping track of thousands of constraints among the thousands of rectangles laid out. This was called computer-aided design. "Aided" is rather a euphemism, but the computer did not complain about the degradation of its role.

While my eyes were glued to the colorful display, and while I was confronted with the evidence of my latest inadequacy, in through the always open door stepped my colleague (JG). He also happened to spend a leave from duties at home at the same laboratory, yet his face did not exactly express happiness, but rather frustration. The chocolate bar in his hand did for him what the coffee cup or the pipe does for others, providing temporary relaxation and distraction. It was not the first time he appeared in this mood, and without words I guessed its cause. And the episode would reoccur many times.

His days were not filled with the great fun of rectangle-pushing; he had an assignment. He was charged with the design of a compiler for the same advanced computer. Therefore, he was forced to deal much more closely, if not intimately, with the underlying software system. Its rather frequent failures had to be understood in his case, for he was programming, whereas I was only using

it through an application; in short, I was an end-user! These failures had to be understood not for purposes of correction, but in order to find ways to avoid them. How was the necessary insight to be obtained? I realized at this moment that I had so far avoided this question; I had limited familiarization with this novel system to the bare necessities which sufficed for the task on my mind.

It soon became clear that a study of the system was nearly impossible. Its dimensions were simply awesome, and documentation accordingly sparse. Answers to questions that were momentarily pressing could best be obtained by interviewing the system's designers, who all were in-house. In doing so, we made the shocking discovery that often we could not understand their language. Explanations were fraught with jargon and references to other parts of the system which had remained equally enigmatic to us.

So, our frustration-triggered breaks from compiler construction and chip design became devoted to attempts to identify the essence, the foundations of the system's novel aspects. What made it different from conventional operating systems? Which of these concepts were essential, which ones could be improved, simplified, or even discarded? And where were they rooted? Could the system's essence be distilled and extracted, like in a chemical process?

During the ensuing discussions, the idea emerged slowly to undertake our own design. And suddenly it had become concrete. "crazy" was my first reaction, and "impossible". The sheer amount of work appeared as overwhelming. After all, we both had to carry our share of teaching duties back home. But the thought was implanted and continued to occupy our minds.

Sometime thereafter, events back home suggested that I should take over the important course about System Software. As it was the unwritten rule that it should primarily deal with operating system principles, I hesitated. My scruples were easily justified: After all I had never designed such a system nor a part of it. And how can one teach an engineering subject without first-hand experience?

Impossible? Had we not designed compilers, operating systems, and document editors in small teams? And had I not repeatedly experienced that an inadequate and frustrating program could be programmed from scratch in a fraction of source code used by the original design? Our brain-storming continued, with many intermissions, over several weeks, and certain shapes of a system structure slowly emerged through the haze. After some time, the preposterous decision was made: we would embark on the design of an operating system for our workstation (which happened to be much less powerful than the one used for my rectangle-pushing) from scratch.

The primary goal, to personally obtain first-hand experience, and to reach full understanding of every detail, inherently determined our manpower: two part-time programmers. We tentatively set our time-limit for completion to three years. As it later turned out, this had been a good estimate; programming was begun in early 1986, and a first version of the system was released in the fall of 1988.

Although the search for an appropriate name for a project is usually a minor problem and often left to chance and whim of the designers, this may be the place

to recount how Oberon entered the picture in our case. It happened that around the time of the beginning of our effort, the space probe Voyager made headlines with a series of spectacular pictures taken of the planet Uranus and of its moons, the largest of which is named Oberon. Since its launch I had considered the Voyager project as a singularly well-planned and successful endeavor, and as a small tribute to it I picked the name of its latest object of investigation. There are indeed very few engineering projects whose products perform way beyond expectations and beyond their anticipated lifetime; mostly they fail much earlier, particularly in the domain of software. And, last but not least, we recall that Oberon is famous as the king of elfs.

The consciously planned shortage of manpower enforced a single, but healthy, guideline: Concentrate on essential functions and omit embellishments that merely cater to established conventions and passing tastes. Of course, the essential core had first to be recognized and crystallized. But the basis had been laid. The ground rule became even more crucial when we decided that the result should be able to be used as teaching material. I remembered C.A.R. Hoare's plea that books should be written presenting actually operational systems rather than half-baked, abstract principles. He had complained in the early 1970s that in our field engineers were told to constantly create new artifacts without being given the chance to study previous works that had proven their worth in the field. How right was he, even to the present day!

The emerging goal to publish the result with all its details let the choice of programming language appear in a new light: it became crucial. Modula-2 which we had planned to use, appeared as not quite satisfactory. Firstly, it lacked a facility to express extensibility in an adequate way. And we had put extensibility among the principal properties of the new system. By "adequate" we include machine-independence. Our programs should be expressed in a manner that makes no reference to machine peculiarities and low-level programming facilities, perhaps with the exception of device interfaces, where dependence is inherent.

Hence, Modula-2 was extended with a feature that is now known as type extension. We also recognized that Modula-2 contained several facilities that we would not need, that do not genuinely contribute to its power of expression, but at the same time increase the complexity of the compiler. But the compiler would not only have to be implemented, but also to be described, studied, and understood. This led to the decision to start from a clean slate also in the domain of language design, and to apply the same principle to it: concentrate on the essential, purge the rest. The new language, which still bears much resemblance to Modula-2, was given the same name as the system: Oberon [1] [2] In contrast to its ancestor it is terser and, above all, a significant step towards expressing programs on a high level of abstraction without reference to machine-specific features.

1 N. Wirth. The programming language Oberon. Software - Practice and Experience 18, 7, (July 1988) 671-690.

2 M. Reiser and N. Wirth. Programming in Oberon - Steps beyond Pascal and Modula. Addison- Wesley, 1992.

We started designing the system in late fall 1985, and programming in early 1986. As a vehicle we used our workstation Lilith and its language Modula-2. First, a cross-compiler was developed, then followed the modules of the inner core together with the necessary testing and down-loading facilities. The development of the display and the text system proceeded simultaneously, without the possibility of testing, of course. We learned how the absence of a debugger, and even more so the absence of a compiler, can contribute to careful programming.

Thereafter followed the translation of the compiler into Oberon. This was swiftly done, because the original had been written with anticipation of the later translation. After its availability on the target computer Ceres, together with the operability of the text editing facility, the umbilical cord to Lilith could be cut off. The Oberon System had become real, at least its draft version. This happened around the middle of 1987; its description was published thereafter 3, and a manual and guide followed in 1991 5.

The system's completion took another year and concentrated on connecting the workstations in a network for file transfer 4, on a central printing facility, and on maintenance tools. The goal of completing the system within three years had been met. The system was introduced in the middle of 1988 to a wider user community, and work on applications could start. A service for electronic mail was developed, a graphics system was added, and various efforts for general document preparation systems proceeded. The display facility was extended to accommodate two screens, including color. At the same time, feedback from experience in its use was incorporated by improving existing parts. Since 1989, Oberon has replaced Modula-2 in our introductory programming courses.

---

3 N. Wirth and J. Gutknecht. The Oberon System. Software - Practice and Experience, 19, 9 (Sept. 1989), 857-893.

5 M. Reiser. The Oberon System - User Guide and Programmer's Manual. Addison-Wesley, 1991.

4 N. Wirth. Ceres-Net: A low-cost computer network. Software - Practice and Experience, 20, 1 (Jan. 1990), 13-24.