The Rise and Fall of Extensible Programming Languages

Tomas Petricek
Charles University in Prague
tomas@tomasp.net

Abstract. The term *extensible programming language* is not something that would be familiar to programmers and computer scientists today. Yet, in the1960s and 1970s, extensible programming languages were a household name. Two symposia on extensible languages were held in 1969 and 1971; a 1974 paper author[1] counted 27 extensible languages proposed by 55 people; Harvard caried out an Extensible Languages Research Program and "A Survey of Extensible Programming Languages" reviewing the work in the area was published in 1974. What were extensible programming languages, where did the term go and why?

In this paper, I give a brief summary of the answer. Extensible programming languages was an umbrella term for languages with a wide range of extensibility mechanisms. All those mechanisms shared the same motivation – to let the language user adapt it to their particular needs – but they differed significantly in their technical realization, ranging from (what we would today call) syntactic macro mechanisms to new procedure and new datatype definitions. Work on all the techniques that were, in the late 1960s and early 1970s, falling under the term extensible programming languages continues to this day, but it is rarely, if ever, treated as belonging to the same research programme. Extensible programming languages provide an intriguing case of a short-lived research programme that brought distinct ideas together, but never managed to integrate them in a way that would ensure its longevity.

Appearance of extensible programming languages

The exact origins of the term *extensible programming language* are difficult to trace. The term existed by 1968 when "Working Conference on Extensible Languages" was held at Carnegie-Mellon University. Authors writing about extensible programming languages,[2] refer to earlier prior work on macros and LISP II, work on high-level programming languages[3] or Brooker's Compiler-compiler,[4] but those did not yet use the term. By 1969 when the Extensible Languages Symposium was held in Boston, the idea was clear:

> *[An extensible language] includes a base language and a meta-language. A program in this system consists of, first, statements in the meta-language which expand, contract, or otherwise modify the definition of the base language to produce a derived language, and, second, statements in the derived language which constitute the executable part of the program. Thus, the system includes facilities to define and then to program in a limitless variety of programming languages – languages which are used for business, scientific, or systems applications, languages which may be simple or complex.*[5]

[1] Standish (1975)
[2] Wegbreit (1970)
[3] Including Cheatham (1966), which includes a reference to an "in preparation" (but apparently never published) paper "ALGOL-E, An Extensible Version of ALGOL-60" outlining macro support for ALGOL 60
[4] A project with a history that has been well documented by Lavington et al. (2016)
[5] Christensen (1969)

The summary in terms of a meta-language and a base language is an appealing abstraction of a messier technical reality, but it served well to unify a wide range of language mechanisms, as well as to provide an attractive motivation for the research programme.

Motivating the extensible language concept

Authors writing about extensible programming languages typically contrast the notion with other categories of programming languages accepted at the time.[6] For example, Solntseff and Yezerski (1974) contrast the notion with universal programming languages (PL/I, Algol 68, UNCOL) that aim to support all known application domains, but end up being too complex (and still lack open-endedness for possible future new application areas) and special-purpose languages that are hard to maintain if used in a greater number. An alternative approach is:

> *the construction of a "growing" or "extensible" language which starts off with a few features, but which can be extended by the user, who can define additional features as he needs them. Thus, the translator of such a language will have built-in open-endedness which will allow the user to tailor the language to his specific needs.*

Variations on this motivation appear in most publications on the topic.[7] However, as the authors of the 1974 survey[8] note:

> *In reading the literature on extensible languages, one is struck by the apparent diversity of directions taken by workers in the field, so that it is difficult to compare the success of the various schemes.*

The goal of the survey is to "systematize the available information". A 1975 paper[9] also notes there are "quite a few dimensions to extensibility—more than we at first suspected" and aims to make sense of those.

Extensible programming languages, technically speaking

The early work cited by research on extensible programming languages often refers to facilities for extending the language syntax (macros), but also mechanisms for the introduction of new data types (a topic actively discussed throughout the 1960s in the context of work on the successor of Algol 60). The range of expected mechanisms is broad. According to Solntseff and Yezerski (1974), an extensible language should have the means for the introduction of:

> *(1) new data types;*
> *(2) new data transformations ([i.e.,] introduction of prefix, infix, or postfix operators (…));*
> *(3) new modes of sequencing of both the primitive and defined data transformations;*
> *(4) new statement types or new syntactic forms.*

The authors go further and characterize extension mechanisms by the stage of the language-translation process at which they occur, listing six stages from lexical and syntactic analysis (syntactic extensions) to intermediate-language analysis and machine code generation.

[6] Listed, for example, in the influential book by Sammet (1969)
[7] For example, Schuman, Jorrand (1970)
[8] Solntseff and Yezerski (1974)
[9] Standish (1975)

The relatively polished abstract analysis in terms of stages hides the messy reality of numerous, largely disconnected, mechanisms that exist in various combinations (and various subsets) in the languages presented in the context of extensible programming languages. They include:

- Textual macros processed prior to parsing, without understanding of the base language
- Systems that describe transformations of the parsed (abstract syntax) tree of a program
- Mechanisms for introducing new user-defined datatypes (with custom operators)
- Systems that adapt the execution at the machine-code level, e.g., by microcoding.

Another attempt to make sense of the different implementations of extensible programming languages was to focus on the form of extension techniques. Standish (1975) lists:

- *Paraphrase*, where a new meaning is defined by (a longer combination) of known terms,
- *Ortophrase*, where orthogonal features are added to the language (typically requiring a "surgery on the underlying guts of a language processor") and
- *Metaphrase*, where interpretation rules of the language are altered to allow new behavior

Extensibility mechanisms as seen today

The diverse range of mechanisms all fall under the general unifying definition of "extending the language" and they all contribute to the objective of adapting the language to user's needs. But they include notions that a contemporary computer scientist would likely not group together:

- Definition of new data types in terms of known datatypes is a basic feature now present in virtually all programming languages. Many languages support operator overloading (redefinition of operator meaning) and some let users define new operators.
- Macros is a specialized functionality offered in a number of programming languages, but not all – the resulting diversity of notations is increasingly seen as a drawback.
- Redefinition of control structures exists as "built-in" feature in a specialized context (monadic notations in functional programming languages), but rarely elsewhere.

Even though the concept of an extensible programming language has disappeared, many of the ideas that appeared in those languages continue to exist, although their meaning has changed and they are typically (perhaps with the exception of macros) no longer seen as "extending the language" – just its ordinary use.

What happened with the extensible language concept

Even among the early proponents of the extensible language concept, there is a disagreement as to whether the research programme has achieved its goals. On the one hand, Cheatham (1971) believes that:

> *[It] has been a few years since "Extensible Languages" were being touted as the "wave of the future" and promised for delivery "tomorrow." There are clearly many at this conference who feel that we are still not there (…). I do not share this opinion. Rather, I think that in a very strong sense "we have arrived" - that there exist languages and host systems which fulfill the goals of extensibility.*

On the other hand, Standish (1975) was of the belief that:

> *[S]ome of us were gripped by a curious euphoria in those days. A number of us believed that users would be able to extend the base of an extensible language rapidly and cheaply to encompass the data, operations, notation, and control natural to many diverse application areas. (…) These beliefs were probably a bit over-ambitious.*

Regardless of whether the research program succeeded or not, much less has been written about the topic of extensible programming languages after 1975.[10] Arguably, many of the mechanisms considered as “extensions” became ordinary parts of many programming languages (and were no longer thought of as extending the language). Other mechanisms kept this meaning (especially macros), but were later thought of and talked about in that more specific context, either using the term macros, or by adopting the term extensible language, but limited to this specific context.

Why has the notion of an extensible programming language disappeared? An answer suggested by this paper is the contrast between its conceptual and technical reality – while the abstract description of the idea (extension via a meta-language) provides a nice unifying view of the mechanisms encompassed by the notion, their actual implementation has remained a irreconcilable mix of different singleton mechanisms.

References

Wegbreit, B. (1970). Studies in extensible programming languages. Harvard University

Christensen, C. (1969) Chairman’s Introduction. ACM SIGPLAN Notices 4 (8): 2–2

Lavington, S. et al. (2016). Tony Brooker and the Atlas Compiler Compiler. Retrieved 19 May 2023. From: http://curation.cs.manchester.ac.uk/atlas/elearn.cs.man.ac.uk/_atlas/docs/Tony%20Brooker%20and%20the%20Atlas%20Compiler%20Compiler.pdf

Cheatham Jr, T. E. (1966). The introduction of definitional facilities into higher level programming languages. In Proceedings of the November 7-10, 1966, fall joint computer conference (pp. 623-637).

Sammet, J. E. (1969). Programming languages: History and fundamentals. Prentice-Hall, Inc.

Schuman, S. A., & Jorrand, P. (1970). Definition mechanisms in extensible programming languages. In Proceedings of the November 17-19, 1970, fall joint computer conference (pp. 9-20).

Solntseff, N., & Yezerski, A. (1974). A survey of extensible programming languages. In International Tracts in Computer Science and Technology and Their Application (Vol. 7, pp. 267-307). Elsevier.

Standish, T. A. (1975). Extensibility in programming language design. In Proceedings of the May 19-22, 1975, national computer conference and exposition (pp. 287-290).

Cheatham Jr, T. E. (1971). Extensible language-where are we going. ACM SIGPLAN Notices, 6(12), 146-7.

[10] There has been a renewed interest in the topic in recent times, but this is out of scope of this paper.