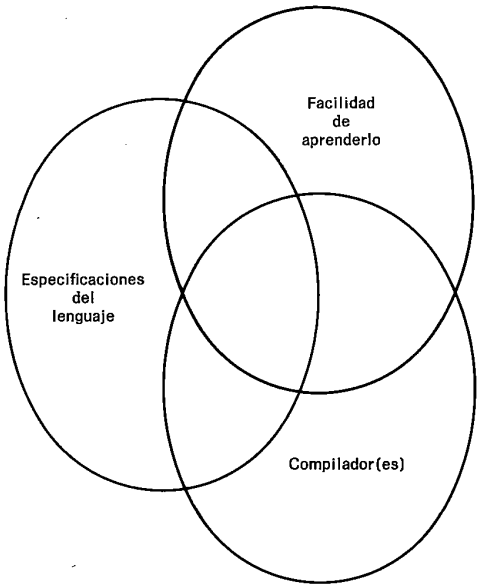
Evolución de los lenguajes de programación (y II)

Félix SALTOR SOLER

El hecho de que el uso de lenguajes de segunda generación sea todavía significativo en programación de aplicaciones, a pesar de las razones técnicas y económicas en favor de su sustitución por los lenguajes funcionales que señalábamos en la primera parte de este artículo (14), implica que hay otros factores detrás de la elección de lenguaje. Uno de ellos, y no el menor, es la rutina y la suspicacia, cuando no la oposición frente al cambio; esta misma rutina que los informáticos, «apóstoles del cambio» en los métodos de los demás, tanto criticamos en los departamentos «usuarios», pero que somos los primeros en tener respecto a nuestros propios métodos.

Otros factores, según Cheatham (72), son: las actitudes (de fabricantes de «hardware» y de «software» de gobiernos, de comités de normalización y de grupos de usuarios); la *disponibilidad de traductores* fiables, bien documentados y mantenidos para varios tipos de ordenadores; la definición de una *norma* satisfactoria; la disponibilidad de *cursos* adecuados; la aparición de *grupos de usuarios* fuertes de distintas disciplinas, y la constitución de una buena *biblioteca de funciones* standard, factores imposibles de evaluar «ab initio» para un nuevo lenguaje.

Figura 3. Factores en la evaluación de un lenguaje.



G. Weinberg viene a decir (15) que en la evaluación de un lenguaje no intervienen solamente sus especificaciones, por importante que sea este punto, sino también (figura 3):

- la existencia de un compilador, para un ordenador asequible, suficientemente bueno en cuanto a inclusión de opciones o características (por ejemplo, extensibilidad mediante «macro facilities») que la norma o las especificaciones del lenguaje dejan en libertad de adoptar (o no consideran), pero que pueden resultar convenientes o necesarias en nuestro caso, en cuanto a fiabilidad y mantenimiento por el autor, en cuanto a rendimiento propio y de los programas generados, en cuanto a documentación y en cuanto a operatividad para explotación; y
- la disponibilidad de textos, cursos y otros medios para aprender a programar efectivamente en el lenguaje en cuestión, adecuados a los programadores o a los aprendices de programador a los que se dirijan.

En resumen, y según recientes palabras de Valentine (74), «el éxito o fracaso de un lenguaje depende a menudo, desgraciadamente, de factores no relacionados con sus méritos técnicos».

Clases de lenguajes funcionales

Pero quizá nos hayamos extendido demasiado, para lo que deba ser este artículo, en estas cuestiones, y sea tiempo de volver a los lenguajes funcionales. La evolución desde los lenguajes uno-a-uno a los lenguajes de tercera generación o funcionales tiene lugar siguiendo dos direcciones distintas (figura 4).

a) Aumentando su nivel sintáctico mediante las agrupaciones de frases en bloques para su ejecución (grupos DO de FORTRAN y PL/I, *for* del ALGOL) o para su declaración y/o delimitación del ámbito de los identificadores (grupos *begin* del ALGOL, PROCEDURE y BEGIN del PL/I), mediante el IF, la generalización del formato libre, etc., al tiempo que su nivel semántico queda incrementado gracias a los datos de distintos tipos (coma fija o flotante, COMPUTATIONAL, PICTURE, etcétera) o a la referencia directa a funciones usuales (exponenciación, raíz cuadrada, logaritmos, funciones trigonométricas, etc.); ambos niveles se refuerzan con las agrupaciones de datos (matrices, estructuras), las expresiones aritméticas en que pueden aparecer múltiples operadores (quedando determinado el orden de las operaciones mediante prioridades y paréntesis), etcétera. Ello da lugar a los lenguajes funcionales *algorítmicos* (clase 3.a) que veremos más adelante.

b) Aumentando fundamentalmente su nivel semántico, para facilitar en gran manera la preparación de programas de ciertos tipos, a base de especificar solamente lo que caracteriza al programa concreto, pero no el algoritmo general: lenguajes funcionales *generadores* o clase 3b, de los que el exponente típico es el RPG (16). En estos lenguajes, en contraposición a los de la clase 3a (y a los de las dos generaciones anteriores), no se describe por tanto el *algoritmo*,

es decir, la secuencia determinada de pasos bien definidos (17), que debe ejecutar el ordenador, y por ello, el orden de las «frases» de un «programa fuente» no corresponde a un orden de ejecución por la máquina. En consecuencia, son los traductores (de programa fuente a programa objeto) correspondientes a estos lenguajes los que llevan incorporados los algoritmos de los programas objeto que generan; las especificaciones de los programas fuente sirven solamente para suministrar argumentos como parámetros de dichos algoritmos y para seleccionar un algoritmo concreto de los varios incluidos en el traductor. Es en este sentido que los programas objeto son *generados*, más bien que resultado de una traducción frase a frase. Además de los RPGs, entran en esta clase los lenguajes de definición de Tablas de Decisión para programas traductores de las mismas, y los lenguajes para generadores de programas de servicio como los generadores de programas de clasificación (18).

Esta contraposición semántica entre los lenguajes funcionales generadores y los algorítmicos puede expresarse también diciendo que en los primeros se especifica solamente *el qué* se quiere obtener del ordenador, pero no *el cómo* (los programas en estos lenguajes son primordialmente *declarativos*, pues lo son la mayoría de sus frases), mientras que en los algorítmicos se especifica explícitamente *el cómo* se quiere que funcione el ordenador, y sólo a través de este cómo (el algoritmo), y si el programador no se ha equivocado, *el qué* (los programas son primordialmente *imperativos*, pues la mayoría de sus frases suelen ser ejecutables, es decir, imperativas); de ahí las «ordenadas» de ambas clases en la figura 4 (19).

También se ha indicado en la figura 4 el bajo nivel sintáctico de los lenguajes funcionales generadores: son a menudo de formato tabular, y las agrupaciones de un programa apenas si existen o son obligadas (por ejemplo: especificaciones de entrada, de salida y de cálculo). Por esta razón, y por su carácter de no algorítmicos, muchos autores no los consideran verdaderos lenguajes; ahora bien, su importancia práctica impondría en cualquier caso su consideración en esta visión general.

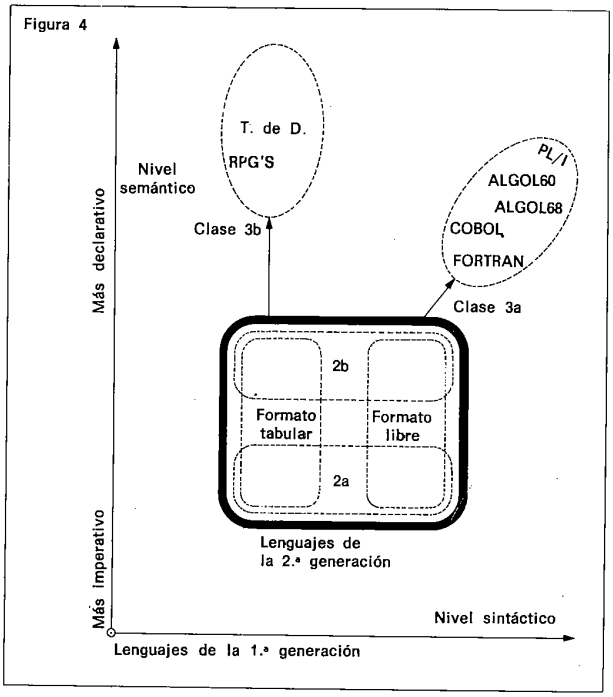
Además de estas dos clases 3a y 3b, incluiremos entre los lenguajes funcionales o de tercera generación a los «intermedios»; pero, antes de hablar de ellos, analicemos la evolución de los funcionales algorítmicos, caracterizada por una primera proliferación por aplicaciones y subsiguientes intentos de consolidación.

Lenguajes funcionales algorítmicos para aplicaciones científico-numéricas

Tras esfuerzos pioneros como los del equipo de la doctora Grace Hooper para el Univac I (lenguajes A o para Algebra: Math-matic; lenguajes B o para gestión —«Business»—: Flow-matic), el primer lenguaje funcional ampliamente usado fue el FORTRAN. Su desarrollo se debe a un equipo de IBM encabezado por John Backus, y estaba orientado, como el ordenador 704 para el que se preparó, a aplicaciones científico-numéricas. Aparte de sus méritos propios, el predominio del mercado estadounidense de ordenadores científicos potentes que alcanzó el 704 en 1957-1958 dio al Fortran una posición casi de standard *de facto* para programas científico-numéricos en Estados Unidos de Norteamérica, posición que ha subsistido hasta cierto punto a través de las versiones posteriores (Fortran II y Fortran IV principalmente).

Independientemente, en Europa, con un mercado de ordenadores científicos menos potentes y más diversificado (y, por tanto, con grandes problemas de compatibilidad entre máquinas), surgía la iniciativa de crear un lenguaje algébrico universal (independiente de las máquinas) e internacional, que desembocó (20) en el informe preliminar de Zurich de 1958 de un lenguaje llamado ALGOL 58. Una conferencia en París en 1960 dio lugar al ALGOL 60, revisado en 1962 en una reunión en Roma. En las especificaciones del ALGOL (en una u otra versión) se basaron una serie de distintos lenguajes (los «dialectos» del Algol), de los que han sobrevivido los extendidos hacia aplicaciones específicas: para mando militar, el JOVIAL (Jules Schwartz Own Version of the International

Figura 4



Algebraic Language) de la SDC, y para simulación (21) el SIMULA de Dahl y el Norwegian Computing Center de Oslo.

Las especificaciones del ALGOL superaban a las del FORTRAN en varios puntos, e incluso se adelantaban «demasiado» a su tiempo en cuestiones como la recursividad o la asignación dinámica de memoria; sin embargo, ya hemos visto que hay otros factores tras la elección de lenguaje, y en particular el ALGOL, en cuanto a «facilidad de aprenderlo» (figura 3), dejaba al principio mucho que desear, pues solamente se disponía de sus especificaciones, redactadas en forma difícilmente inteligible por el programador medio. Por todo ello no fue adoptado extensamente más que en la URSS y otros países de Europa oriental, y en universidades, sobre todo europeas (22).

Un punto a destacar en las especificaciones del Algol es la descripción de su sintaxis, que marca un hito fundamental en la definición de lenguajes, al emplear un metalenguaje formal (23), la llamada «Backus Normal Form» o «Backus-Naur Form» (BNF), descrita en Backus (60). Ello constituye un avance importante en varios aspectos, en particular en la eliminación de ambigüedades en la descripción de lenguajes y en la preparación de compiladores «*syntax-directed*» (24).

Lenguajes funcionales algorítmicos para aplicaciones de gestión

El intento de crear un lenguaje universal para aplicaciones de gestión partió más bien de fabricantes de ordenadores, y el gobierno estadounidense proporcionó, al patrocinar en 1959 la formación de CODASYL *(Conference on Data Systems Languages)*, el marco adecuado para reunirse entre sí y con los grandes usuarios. Los dos puntos de vista existentes, el de que era urgente disponer de un lenguaje común, aunque no fuere perfecto, y el de que hacía falta profundizar en los problemas de la programación antes de poder especificar un lenguaje standard, dieron lugar a la constitución de dos comités: el a «corto plazo» y el a «plazo intermedio». El comité a corto plazo, al que se había encomendado examinar las técnicas y lenguajes existentes y recomendar cómo incluirlas en el nuevo lenguaje, fue más allá y, basándose en el Flowmatic de Univac y el Commercial Translator de IBM, diseñó el COBOL. El comité a plazo intermedio encontró este COBOL insatisfactorio, e inferior al FACT que Honeywell acababa de anunciar, y se produjo la «batalla de los comités». El comité a corto plazo obtuvo el apoyo del consejo ejecutivo de CODASYL, con

Aplicaciones / Año	científico-numéricas	de gestión	tratamiento de sartas	tratamiento de listas	tratamiento de fórmulas algébricas
1957	Fortran				
1958	Algol 58				
1959					
1960	Algol 60	COBOL			
1961			COMIT	IPL-V	
1962				LISP 1.5	
1963				SLIP	ALPAK
1964			SNOBOL		FORMAC
1965			L[6]		
1966	← PL/I →				
1967					
1968	← ALGOL 68 →				

Figura 5. Evolución de la aplicabilidad de los lenguajes funcionales algorítmicos

lo que su COBOL, con ciertos retoques (COBOL 60), fue adoptado por dicho organismo (25).

A pesar de la reticencia de algunos fabricantes (especialmente Honeywell e IBM), el anuncio por el mayor usuario, el gobierno estadounidense, de que no contrataría ordenadores que no dispusiesen de compilador COBOL a menos que el fabricante pudiera demostrar, que esta ausencia no iba en detrimento de su eficacia, obligó a todos ellos a ofrecer COBOL en todas sus máquinas, y este lenguaje (con las modificaciones posteriores que se le han ido introduciendo) pasó a ser el más usado. La existencia actual de lenguajes técnicamente superiores no lo ha desbancado (y me remito a las razones antes apuntadas) de esta posición de primacía.

Otros lenguajes funcionales especializados

Paralelamente a la introducción de lenguajes para aplicaciones científico-numéricas y para aplicaciones de gestión, aparecen lenguajes para otras áreas de aplicación.

Para *tratamiento de listas* mencionaremos en primer lugar el IPL (Information Processing Language) de Newell, Simon y Shaw, cuya primera versión es de 1956 (el IPL-V es de 1960). Más conocido, y quizá con mayor influencia en el software posterior, es el LISP (LISt Processor) debido a John McCarthy (1960); el LISP 1,5 es de 1962, y el proyecto de LISP 2 fue abandonado. Finalmente, SLIP (Symmetric LIst Processor), de Weizenbaum, es una «extensión» del Fortran, de 1963.

Respecto al *tratamiento de sartas* («strings»), y sobre todo de sartas de caracteres, área no siempre claramente distinguida de la anterior, hay que citar el COMIT de V. Yngve (1961), el L[6] (Bell Telephone Laboratories Low-Level Linked List Language), y sobre todo los lenguajes SNOBOL StriNg Oriented symBOlic Language; el primero es de 1963/64, y el SNOBOL 4 —actualmente en su versión 3.11— de 1967/68), debidos a Griswold, Polonsky y Farber, sin olvidar a Gimpel, con su cohorte de dialectos o versiones especiales (SPITBOL, SITBOL, FASBOL, ELFBOL, SNOBOL [—1], SNOBOL-10, SNOBAT, etc.).

Es discutible si debe incluirse aquí el área del tratamiento de fórmulas algébricas (ALPAK, FORMAC, Fórmula Algol, etc.); baste remitir al lector a Bobrow (68). Análogamente, para el mando numérico de máquinas-herramientas, véase Hatvany (73). Y no consideramos los lenguajes estrictamente de simulación, ni siquiera los de simulación discreta (GPSS, SIMSCRIPT, CSL, SOL, GASP, etc.).

Los intentos de consolidación y aplicabilidad

En una segunda fase de la evolución de los lenguajes funcionales algorítmicos aparecen lenguajes que cubren múltiples áreas, que requerían antes lenguajes distintos, para «consolidar» en un solo lenguaje no sólo las aplicaciones científico-numéricas y las de gestión, sino otras áreas más «especializadas». Esta evolución acompaña a la del hardware, ambas condicionadas por la de las utilizaciones, y se centra fundamentalmente en el PL/I y el Algol 68, como la figura 5, tomada de Duby, indica (26).

El PL/I (originalmente siglas de Programming Language/I), destinado por IBM a cubrir los 360 grados de las aplicaciones en cuanto a lenguaje, como el sistema/360 debía hacerlo en cuanto a hardware, fue diseñado en 1963/64 por un comité de tres personas de IBM y tres de usuarios, en el marco de SHARE (27),

y ampliamente revisado subsiguientemente. Se va implantando progresiva pero muy lentamente (28).

El Algol 68 no es un nuevo dialecto del Algol 58 o 60, sino un nuevo lenguaje, patrocinado igualmente por la IFIP (29) a través de su Working Group 2.1, y debido fundamentalmente a Van Wijngaarden. No es apenas usado, no tanto por dificultad de aprenderlo, pues existen descripciones informales (incluso en castellano: Sanchis y Morales [73]), como por la escasez de compiladores.

Mientras el PL/I quiere poner al alcance del programador medio las posibilidades del Fortran, del Algol 60 y del Cobol, el Algol 68 se orienta al rigor y elegancia matemáticas y a la generalidad. En ambos casos son interesantes los métodos de formalización, ya no sólo de la sintaxis, sino también de la semántica (método ULD del Laboratorio de IBM en Viena, y método del Mathematisch Centrum de Amsterdam, respectivamente).

Hemos visto en la figura 5 cómo la *aplicabilidad* de estos dos lenguaje es mayor que la de lenguajes funcionales algorítmicos anteriores, lo cual consiguen mediante un lenguaje más complejo. Pues es una regla general que *al aumentar el nivel semántico decrece la aplicabilidad,* excepto si se aumenta la complejidad, como la figura 6 refleja (30). Esto se manifiesta especialmente en la clase 3b de lenguajes funcionales generadores, de alto nivel semántico, pero de aplicabilidad limitada, mientras que en lenguaje de máquina o en Assembler «teóricamente se puede hacer todo».

Lenguajes funcionales «intermedios»

Con la aparición de los lenguajes funcionales, la programación de aplicaciones pasa a efectuarse en estos lenguajes, pero la programación de sistemas sigue generalmente usando los lenguajes uno-a-uno (31). Pronto entre los que programan sistemas surge la pregunta: ¿no podríamos reunir en un lenguaje «lo mejor de ambos mundos» para poder tanto usar estructuras compuestas como descender al nivel del bit de un registro?

Aparecen así una serie de lenguajes, que reciben nombres como lenguajes de programación de sistemas, «systems implementation languages», «machine oriented high order languages» (MOHOLs), «machine oriented languages» (MOLs), etc., y que aquí serán denominados lenguajes *intermedios* (clase 3c). Los dividiremos, de acuerdo con Lawson (32), en dos categorías:

a) lenguajes orientados a la máquina: PL 360 de Wirth y sus derivados franceses LPS, PL/S (Programming Language/Systems, antes llamado BSL) ampliamente usado por IBM para 360/370, BLISS del PDP-10, MARY de la Universidad de Trondheim; y

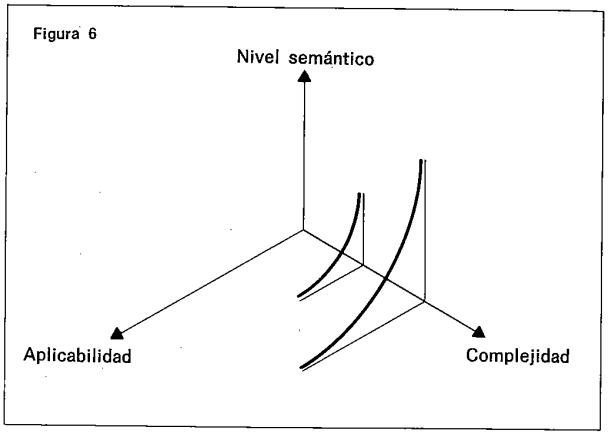
b) lenguajes generales de construcción de software, como el británico BCPL (33), el SILL (Systems Implementation Language from Linköping) de Lawson, o el lenguaje del Burroughs 6700.

El empleo de estos lenguajes para la escritura de traductores y partes de los sistemas de explotación ha limitado la utilización de los lenguajes uno-a-uno a casos especiales (34).

«CUARTA GENERACIÓN»: lenguajes dialógicos

Con la introducción de terminales de teleproceso aparece la posibilidad de programar desde ellos de forma interactiva, en *conversación* o *diálogo* entre el programador y el sistema. Diremos entonces que el sistema es *conversable,* que la programación es *dialogada* (o interactiva) y que el lenguaje, si está diseñado para ello, es *dialógico.* No entran, pues, en esta clase las versiones para sistemas conversables de los lenguajes funcionales ya vistos (Fortran, Cobol, PL/I,

Figura 6



etcétera), sino los lenguajes que permiten la interacción con el sistema también durante la ejecución del programa, es decir, los lenguajes para uso «*en inmediato*» (o «en directo») más bien que «*en diferido*» (35).

De estos lenguajes dialógicos, que dan lugar a una nueva generación (clase 4), el más popular es probablemente el BASIC, desarrollado en el Dartmouth College (New Hampshire) en 1961, y que viene a ser como un Fortran dialógico.

El APL («A Programming Language»), definido en 1962 por K. Iverson y desarrollado luego con la ayuda de Falkoff, es un lenguaje dialógico de tratamiento de matrices, pero de utilización para aplicaciones no sólo numéricas, sino también de gestión, de enseñanza, etc. Los programas en APL tienen un «estilo» que apasiona a sus fanáticos y exaspera a sus detractores. Extensiones interesantes del APL son el PPL y el APLSV.

Finalmente, puede considerarse, como hace Cheatham (72), que el LISP, antes citado para el tratamiento de listas, es inherentemente dialógico y por lo tanto de esta cuarta generación.

Con las clases así obtenidas podemos construir el *esquema clasificatorio* de la figura 7.

De lo actual hacia...

Las características que, a mi parecer, marcan el estado actual y la evolución próxima de la tecnología de los lenguajes son las siguientes:

1) El centro de interés de la investigación se ha desplazado del lenguaje en que se programa al área *qué es programar y cuál es su metodología* (36), e impacta en aquél a través de ésta. Así, se reconoce que aún estamos demasiado influidos por lenguajes anteriores y que mezclamos en nuestros lenguajes y programas cuestiones de distintos niveles, como señala Wirth (74), y que hay que adoptar la *programación estructurada,* todo lo cual induce al uso de lenguajes como Simula, BLISS, y, sobre todo, PASCAL, desarrollado en 1969 por N. Wirth, de la Eidgenosse Technische Hochschule de Zürich. O se apunta hacia lenguajes verdaderamente extensibles (en tipos de datos, en particular), y se experimenta con el PPL, con el mismo PASCAL, o con el EL-1 de Wegbreit (Harvard).

2) El «ideal» de un lenguaje con un nivel sintáctico igual o superior al de los funcionales algorítmicos y al mismo tiempo con un nivel semántico semejante al de los funcionales generadores (figura 4) y con amplia aplicabilidad, comportaría una complejidad grande (figura 6). La solución podría ser no conocer todo el lenguaje, sino un subconjunto menos complejo, y que un sistema conversable nos guiara en el uso del resto del lenguaje, que debería, pues, ser dialógico. En todo caso estamos lejos de ello.

Félix Saltor

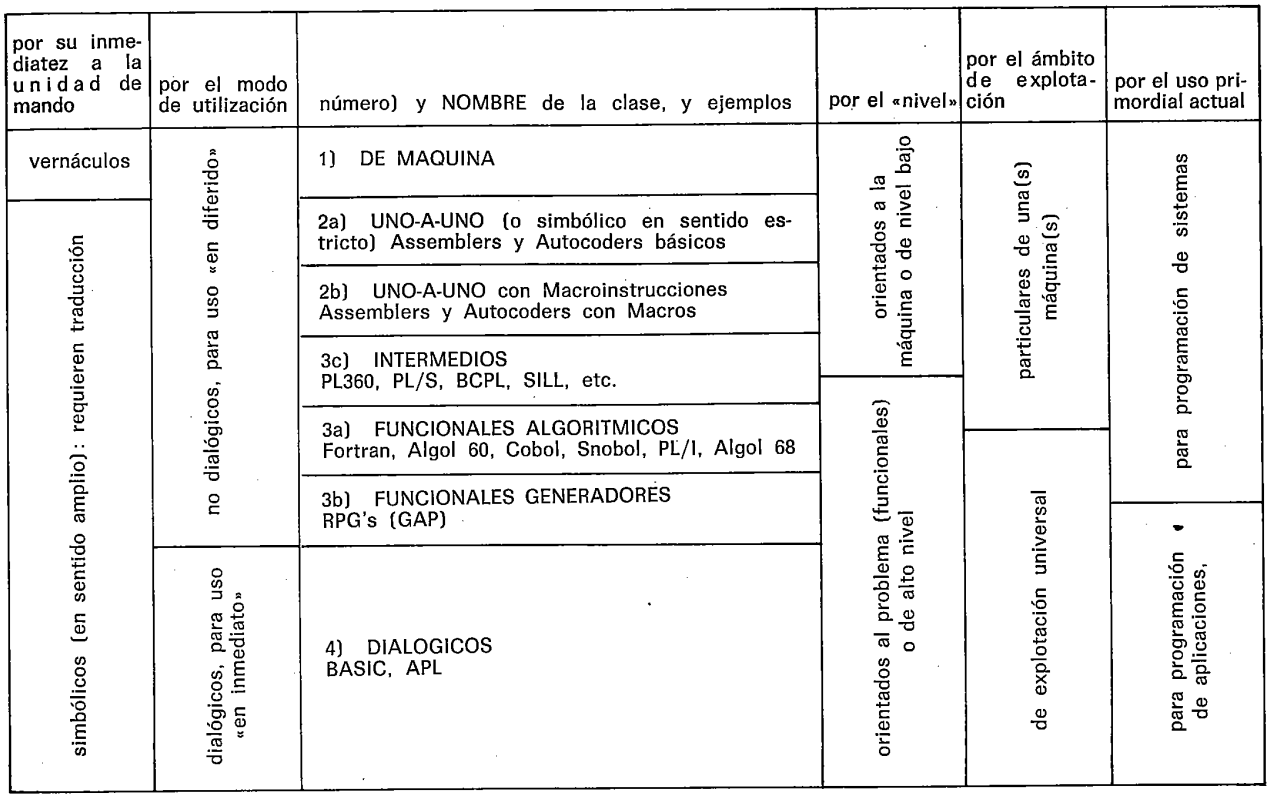


Figura 7. Esquema clasificatorio de los lenguajes de programación

NOTAS

(14) La primera parte de este artículo apareció en NOVATECNIA, Extraordinario n.º 8, 1974, que al mismo tiempo hacia de n.º 0 de NOVATICA. He de señalar que la figura 1 apareció con errores (que podrán subsanarse refiriéndose a la figura 4 del presente número), y que la ilustración del pie de la página 29 no se debía a mí, sino que fue tomada por NOVATICA de un documento de Infotech.

(15) En el curso de septiembre a diciembre de 1967 del IBM European Systems Research Institute.

(16) En sus diversas versiones, que serán, pues, otros tantos lenguajes. RPG son las siglas de Report Program Generator, y a veces es designado como GAP o Generador Automático de Programas.

(17) Habría que añadir, en rigor, la propiedad de que siempre termine, para que sea un algoritmo y no simplemente un *procedimiento*, o referirse a descripciones como la de Trajtenbrot (60): «Una prescripción exacta del orden determinado en que ha de ejecutarse un sistema de operaciones para resolver todos los problemas de un cierto tipo».

(18) La dirección de esta evolución, centrada en lo semántico, marginando lo sintáctico y a costa, como luego se señala, de su aplicabilidad, es la misma que conduce a los lenguajes de análisis (y en cierto modo a los de interrogación espontánea de bases de datos), que quedan fuera de este artículo (ver Nota 3). Incluso para los lenguajes de esta clase 3b es discutible hasta qué punto son los programadores sus usuarios preferentes.

Además, es claro que el paso siguiente (ya dado en bastantes casos) corresponde a los programas generalizados parametrizables, tales que lean sus argumentos y ejecuten la función correspondiente (p. ej., una clasificación) sin generar programa objeto, o más concretamente, *sin traducción*. Pero en este caso las especificaciones de estos argumentos no pueden ya considerarse como «programas fuente», ni su escritura como una programación.

(19) De todos modos, la distinción entre frases declarativas y frases imperativas no es absoluta, «pues incluso las oraciones llamadas declarativas (DECLARE X FIXED DECIMAL) provocan una acción del ordenador, e incluso las oraciones llamadas ejecutables (A = B + C) tienen un valor afirmativo», como decíamos en *Introducción a la Semiótica para informáticos*, en Aladjem y otros (71).

(20) Además de la Gesellschaft für Angewandte Mathematik und Mechanik (GAMM), donde surgió la idea, intervinieron por parte europea la Association Française de Calcul, la British Computer Society y la Nederlands Rekenmachine Genootschap, y por parte estadounidense la Association for Computer Machinery (ACM) con la colaboración de SHARE y USE, a través, entre otros, de Backus, Bauer, Katz, McCarthy, Naur, Perlis, Rutishauser, Turanski, Vauquois y Van Vijngaarden. Las similitudes entre Fortran y Algol se explican en parte por la participación de Backus en ambos proyectos.

(21) Remarquemos que a pesar de su nombre SIMULA (o Simula 67) no es útil solamente para simulación, y es interesante por varios conceptos, como los procesos «casi paralelos» (corutinas), las «activaciones» (con acciones y propiedades) de «clases de objetos», y su comunicación. De aquí su mención, a pesar de excluir los lenguajes de simulación. (Algo similar ocurrirá con el SIMPL/I, extensión del PL/I.)

(22) Respecto a la historia del Algol, véase Bemer (69).

(23) Un *metalenguaje* es el «lenguaje» que se emplea para describir un cierto lenguaje (en nuestro caso, para describir un lenguaje de programación). Antes de la BNF, el metalenguaje usual era el inglés, es decir, un lenguaje no formal.

Destaquemos que la especificación en BNF, originada por Backus en el campo de la informática, resulta ser esencialmente equivalente a las *Gramáticas de Estructura Sintagmática* (más exactamente, a las Gramáticas «Context-free») originadas por Noam Chomski en el estudio de las lenguas naturales, así como a los *Sistemas de Producción de Post*, originados por éste en el estudio de la lógica simbólica.

Una introducción a la *Descripción formal de los lenguajes de programación* se encuentra en Aladjem y otros (71).

(24) A diferencia de los compiladores «*syntax-controlled*», en los que la sintaxis del lenguaje fuente está «metida dentro» de su algoritmo, en los compiladores «*syntax-directed*» el algoritmo de análisis sintáctico es generalizado, y la sintaxis del lenguaje fuente reside formalizada externamente al mismo.

(25) Esta descripción sigue la de Rosen (67). Son notables ciertas analogías entre esta batalla de los comités y el «debate» actual sobre la propuesta del Data Base Task Group del CODASYL.

Por otra parte, sería curioso estudiar hasta qué punto ciertas características del COBOL puedan deberse a la activa participación femenina en el comité que lo diseñó.

(26) La figura se ha tomado de los apuntes de las explicaciones de J. Duby en el IBM European Systems Research Institute durante el curso de septiembre a diciembre de 1970.

(27) SHARE es una asociación de empresas usuarias de ordenadores IBM, originalmente orientada a aplicaciones científicas, como GUIDE lo estaba a las de gestión. Su equivalente europeo es SEAS.

(28) Philippakis (73) le asigna un índice de uso de 4, frente a 59 para el COBOL, 5 para el Fortran y 6 para los Generadores (datos de EE.UU. de 1972). Factores específicos contra su introducción pueden haber sido el que hasta muy recientemente no haya tenido compiladores para ordenadores de otros fabricantes ni un proyecto en marcha para su normalización, y la falta de apoyo suficiente por parte de la propia IBM.

(29) International Federation for Information Processing, fundada como consecuencia del Congreso Internacional de París de 1959. Representa a España en la IFIP el Consejo Superior de Investigaciones Científicas, mediante un «Comité español de la IFIP», al parecer.

(30) Está claro que esta figura 6 es meramente una ilustración de la regla enunciada, sin valor cuantitativo. Digamos que hacer avanzar la tecnología de los lenguajes no es tanto desplazarse por la superficie esquematizada en la figura, como desplazar a dicha superficie.

(31) Con notables excepciones, que llegan a escribir el programa en lenguaje funcional y traducirlo luego a mano a lenguaje de segunda generación.

(32) En el Seminario «Técnicas recientes de implantación de software de sistemas» de la A.T.I. (Barcelona, junio de 1974).

(33) BCPL (Basic CPL) es una versión del CPL orientada a la eficiencia de ejecución. CPL es un interesante lenguaje general, diseñado por un equipo del Cambridge University Mathematical Laboratory y del London Institute of Computer Science encabezado por Christopher Strachey y pensado inicialmente (1963) para la escritura de compiladores; CPL son las siglas de Combined Programming Language (o, según algunos, de Christopher's Private Language o de Cambridge Plus London).

(34) No hemos situado la clase 3c en el sistema de coordenadas nivel sintáctico-nivel semántico, ya que estos lenguajes (sobre todo los de la primera subcategoría) se distinguen porque son más extensos los conjuntos de puntos que les corresponden en dicho sistema, mientras que representamos a cada lenguaje por un solo punto.

(35) Prefiero la expresión «en diferido» a la «en batch», porque «batch» tiene varias acepciones y no puede contraponerse a «en teleproceso», como señalaba Plateau (67) en un memorable artículo.

(36) Este desplazamiento puede advertirse en los últimos congresos de la IFIP. En el de Edimburgo (1968) aparecieron todavía nuevos lenguajes (Algol 68) o extensiones a los conocidos; en el de Liubliana (1971) solamente dos sesiones se dedicaron a Algol 68 y a APL, mientras en algunas comunicaciones se trataba de programar sin GO TO; en el de Estocolmo (1974) se habló ampliamente de programación estructurada y apenas de lenguajes.

BIBLIOGRAFÍA

ALADJEM y cols. (71): «Lingüística e Informática». ATI, Barcelona.

BACKUS (60): «The Syntax and Semantics of the proposed International Algebraic Language of the Zurich ACM-GAMM Conference», *Procedures of the International Conference of Information Processing*, UNESCO, París, 1959. R. Oldenbourg, München & Butherworks, London.

BEMER (69): «A politico-social history of ALGOL», *Annual Review in Automatic Programming*, Vol. 5.

BOBROW (68) (ed.): «Symbol manipulation languages and techniques» (Conferencia de IFIP en Pisa) North Holland, Amsterdam.

CAMPS y MARTÍ (67): «Visión general del software». ATI, Barcelona. (Hay una edición revisada de 1972.)

CHEATHAM (72): «The recent evolution of programming languages» en *Information Processing 71*, North Holland, Amsterdam.

FREIXA (67): *Cálculo digital*. Real Academia de Ciencias y Artes, Barcelona.

HATVANY (73): «Computer languages for numerical control» (Conferencia de IFIP/IFAC en Budapest), North Holland, Amsterdam.

HIGMAN (67): *A comparative study of programming languages*.

PHILIPPAKIS (73): «Programming language usage». *Datamation*, October.

ROSEN (67): *Programming systems and languages*. McGraw-Hill, New York.

SAMMET (69): *Programming languages: History and fundamentals*. Prentice-Hall.

SANCHIS y MORALES (73): *Algol 68/60*. Madrid.

TEICHROEW (72): «Improvements in systems building». Seminario sobre «Diseño de sistemas informáticos». ATI, Barcelona.

TEICHROEW (74): «Improvements in the systems life cycle». en *Information Processing 74*. Stockholm.

TRAJTENBROT (60): «Algoritmos y calculadores automáticos» (en ruso). Moscú, 2.ª ed., 1960. Trad. *Introducción a la teoría matemática de las computadoras y de la Programación*. Siglo XXI, México, 1967.

VALENTINE (74): «Comparative notes on ALGOL 68 and PL/I». *The Computer Journal*, Vol. 17, N. 4. London, nov. 1974.

WIRTH (74): «On the design of programming languages». En *Information Processing 74*. Stockholm.