UBL (Lenguaje de la Universidad de Barcelona): un lenguaje para la enseñanza de la programación en castellano

Jose María Blasco y Guillermo Alonso

Centro de Cálculo. Departamento de Aplicaciones Científicas.
UNIVERSIDAD DE BARCELONA.

Comunicación presentada en las JORNADAS SOBRE "INFORMATICA Y EDUCACION EN LA ENSEÑANZA BASICA Y MEDIA"

Madrid, 26-27-28 Noviembre 1984

Resumen

Se describen las características del lenguaje UBL, concebido como una herramienta pedagógica en la enseñanza de la programación, que se está utilizando en los cursos de Informática del Centro de Cálculo de la Universidad de Barcelona y está siendo objeto de una experiencia valorativa en el marco de un proyecto conjunto con el ICE y la Facultad de Psicología de la U.B. El lenguaje, partiendo de la estructura de PASCAL, incorpora elementos propuestos en lenguajes más recientes (ADA, MODULA-2, CLU, ALPHARD o REXX). También proporciona tres opciones lingüísticas para escribir los identificadores reservados y predefinidos: castellana, catalana e inglesa. Por otra parte se comenta la relación entre la formalización matemática y el aprendizaje de la programación. Por último se tratan cuestiones de implementabilidad, eficiencia y valoración del lenguaje, y se comentan las perspectivas actuales del proyecto UBL.

Palabras clave

programación, lenguajes de programación, enseñanza de la programación

Introducción

La Universidad de Barcelona, a través de su Centro de Cálculo, ha venido impartiendo cada año dos cursos de formación informática, de programador y analista de aplicaciones científicas, orientados fundamentalmente a profesores, doctorandos y alumnos de segundo y tercer ciclo.

Hace dos años se planteó la necesidad de disponer de una herramienta didáctica que permitiese la introducción de las técnicas más recientes de programación, de la manera más asequible para las personas que siguen los cursos mencionados, teniendo en cuenta que provienen de distintas especialidades, tienen distinta experiencia previa y utilizarán distintos lenguajes en aplicaciones muy diversas en sus respectivos departamentos.

Por ello se puso en marcha el proyecto UBL (Lenguaje de la Universidad de Barcelona) con dos objetivos principales:

Por una parte disponer de un compilador implementado de un lenguaje que, aunque no coincidiera con ninguno de los utilizados habitualmente, permitiera incorporar los elementos necesarios para desarrollar los conceptos de programación que se deseasen enseñar, sin tener que exponerlos de forma únicamente teórica.

Y, por otra parte, se trataba de proporcionar a los alumnos un lenguaje de programación con identificadores (reservados y predefinidos) en una lengua familiar (castellano o catalán, además de la versión en inglés). La opción de utilizar lenguajes de programación basados en la propia lengua ha sido recientemente defendida por diversos autores, como Dijkstra [Dijk 80], Botella [PBot 84] o González [Gonz 84], en el sentido de que la utilización de los elementos lingüísticos habituales facilita notablemente la comprensión de las estructuras y contenido semántico de los programas, permitiendo al principiante concentrarse en los conceptos subyacentes a la programación, sin sufrir la carga adicional de una codificación extraña.

El planteamiento expuesto se completa considerando que, una vez se ha aprendido a programar correctamente con un lenguaje potente y familiar, resulta fácil enseñar como utilizar esas técnicas generales de programación con cualquier otro lenguaje. Este proceso permite además que el alumno conozca de entrada la mayoría de las posibilidades de las modernas técnicas de programación y pueda apreciar las limitaciones de lenguajes de mucho arraigo pero poco idóneos para programar estructuradamente, como BASIC O FORTRAN, motivándole fuertemente para que utilize lenguajes potentes y estructurados, como PASCAL o PL/I (que en sus versiones más modernas dispone al menos de un repertorio suficiente de instrucciones estructuradas).

El Centro de Cálculo ha desarrollado (en parte con el soporte de una beca de estudios de IBM) una versión operativa de UBL, que se describe a continuación.

Descripción del lenguaje UBL

UBL pertenece a la familia de los lenguajes secuenciales imperativos (como BASIC, FORTRAN, PASCAL, PL/I o ADA) y comparte con algunos de ellos muchas de sus características. Está inspirado principalmente en Pascal, e incorpora estructuras que pueden hallarse en ADA [Ada 83], MODULA-2 [Wirth 82], ALPHARD [Shaw 77], CLU [Lisk 77] o REXX [IBM 83]. Partiendo de un conocimiento básico de PASCAL, describiremos aquí algunas de las características distintivas de UBL respecto de ese lenguaje.

- El punto y coma se utiliza como terminador de instrucciones y no como separador.
- Existen tres opciones para escribir los identificadores reservados y predefinidos: castellana, catalana e inglesa.
- Incorpora un mecanismo de tipos parecido al de PASCAL, con algunas variaciones tomadas de ADA (como las "incomplete type declarations" para definiciones recursivas de "pointers"); incorpora también los tipos usuales (Entero, Caracter, Real y Lógico).
- La mayoría de construcciones se escriben en estilo parentizado

```
mientras C haz
  S
fin mientras;
```

  Se permite escribir sólo **fin** para construcciones textualmente muy cortas, que ocupen una sola línea.
- Incorpora también el concepto de **modulo** (como los **package**s de ADA), lo cual permite la creación, verificación e utilización de múltiples niveles de abstracción conceptual.

- Define cuatro tipos de subprograma:
  - las **acciones** (llamados "procedures" o "subprogramas" en otros lenguajes), que aumentan el repertorio de instrucciones utilizables;
  - las **funciones**, que permiten la creación de abstracciones de evaluación;
  - las **condiciones** (sinónimo de funciones lógicas), que se proporcionan dada su frecuencia de utilización;
  - y las **secuencias**, subprogramas que **producen** una serie de valores (en contraste con las funciones, que sólo **producen** uno) y se manipulan exclusivamente mediante los operadores de alto nivel **para** y **existe**:

    ▲

    para variable **en** Secuencia **talque** condición **haz**
      Instruccione(s)
    fin para;

    extensión de la instrucción **for** de PASCAL en la que **to** y **downto** quedan incluidos como las secuencias predefinidas ASC y DESC; tiene como efecto la ejecución de las instrucciones, tomando la variable los diversos valores producidos por la secuencia, condicionada a la verificación de la condición.

    ▲

    existe variable **en** secuencia **talque** condición

    predicado que devuelve un valor de tipo Lógico y, si éste es Cierto, asigna a la variable el primer valor de (producido por) la secuencia que verifique la condición.

  Algunos lenguajes experimentales han introducido conceptos parecidos; pueden consultarse las referencias [Shaw 77] y [Lisk 77] para una primera aproximación; se notará que lenguajes más recientes (véase [MacL 83]) incorporan también conceptos similares, aunque sin mencionarlo explícitamente.

Ejemplos: El primer ejemplo es un algoritmo para calcular la media de dos números reales:

```
programa Media es
  var a, b, media: Real;
haz
  Escribe_linea ' Escribe dos números reales:';
  Lee a,b;
  media ← (a + b)/2;
  Escribe_linea ' Su media es:',media;
fin programa;
```

Como segundo ejemplo, el siguiente programa en UBL, que, a partir de una frase acabada por un punto, cuenta el número de letras A que aparecen en la frase.

```
programa Cuenta_las_as es
  var c: caracter; numero_de_as: entero;
haz
  numero_de_as ← 0;
  escribe_linea 'Escribe una frase acabada por un punto:';
  repite
    lee c;
    si c = 'A' entonces
      numero_de_as ← numero_de_as + 1;
    fin si;
  hastaque c = '.';
  escribe ' Hay ',numero_de_as,' letras A.';
fin programa;
```

El lenguaje está pensado de modo que un subconjunto operativo (incluyendo tipos básicos, declaraciones, asignación, condicionales y repetición) pueda ser aprendido en pocos días; dada su potencia, un gran número de conceptos avanzados de programación pueden estudiarse sin abandonar la notación.

Como se habrá visto en los ejemplos, el lenguaje, sin dejar de aportar novedades y modificaciones al conjunto de los ya existentes, se mantiene (en programas sencillos) suficientemente cercano a otros

(BASIC, FORTRAN, PASCAL, ADA) como para que el aprendizaje posterior de éstos no se haga muy difícil.

Se ha comprobado que la actividad de programar está estrechamente ligada (además de a la comprensión y dominio de la propia lengua) a la capacidad de formalización matemática (véase [Dijk 83]). A este respecto, podría ser interesante coordinar el estudio de las Matemáticas con el aprendizaje de la programación. El lenguaje UBL está bien preparado para ello, ya que incluye entre sus posibilidades la definición de productos cartesianos (**tupla**), conjuntos (**conjunto**), aplicaciones (**aplicacion**), y el manejo de valores lógicos (tipo Lógico) y de pseudo-cuantificadores sobre secuencias (**para** y **existe**). Como ejemplo presentamos el siguiente algoritmo escrito en UBL para la confección de un listado de los 100 primeros números primos; aunque no es óptimo, es una traducción (utilizando el concepto de secuencia) de los que suelen ser creados por los alumnos en una primera aproximación al problema, y permite apreciar la elegancia y simplicidad de este concepto.

```
programa Generador_de_numeros_primos es
  const Numero_de_primos = 100;
  const Infinito = 2000000000; (* mas o menos *)
  var T: tabla [1..Numero_de_primos] de Entero;
  var l, p, i: Entero;
haz      (* 1,2 y 3 ya nos los sabemos *)
  T[1] ← 1; T[2] ← 2; T[3] ← 3;
  l ← 3;      (* Ya tenemos tres primos *)
  mientras l < Numero_de_primos haz
    (* asc(a,b) representa la secuencia de los enteros ascendentes entre a y b *)
    si existe p en asc (T[l]+2,Infinito) talque
      no existe i en asc (2,l) talque p mod T[i] = 0 entonces      (* p es primo: *)
      l ← l + 1; T[l] ← p;
    fin si;
  fin mientras;
  escribe_linea 'Tabla de los 100 primeros numeros primos:';
  para i en Asc(1,100) haz
    escribe_linea T[i];
  fin para;
fin programa;
```

[Notas: '←' es el operador de asignación; los comentarios se han escrito entre '(*' y '*)'].

Hay que decir que las secuencias, a pesar de ser construcciones de alto nivel, están implementadas con una eficiencia (en cuanto a consumo de recursos máquina) similar a la de las construcciones clásicas.

Implementación, operatividad y valoración

Existe una implementación operativa (Version 0 Release 1.1) del lenguaje UBL tal como se presenta aquí; actualmente funciona en los ordenadores de este Centro de Cálculo, un 4341-2 y un 3083-XE de IBM, bajo el Sistema Operativo VM/SP Release 3.1 (HPO); con el 3083 se consiguen velocidades de compilación de 3350 líneas/minuto, para programas densos.

El compilador (en una primera versión) se utilizó como soporte del "Curso de Programador en Aplicaciones Científicas 1983-84" impartido en este Centro de Cálculo, y se está utilizando en el curso homónimo que se desarrolla este año, con una asistencia de 108 alumnos.

Asimismo, en el marco de un proyecto conjunto del Centro de Calculo, el Departamento de Psicología Experimental y el Instituto de Ciencias de la Educación de la Universidad de Barcelona, se han realizado diversas experiencias (seguimiento y análisis del Curso de Programador, comparación entre diversas opciones del lenguaje), cuyos primeros resultados se presentaron en [Tub 84a] y forman parte de una Tesina presentada en la Facultad de Psicología en Septiembre de 1984 [Tub 84b]. Una valoración general de la experiencia se expone en otra comunicación presentada en estas Jornadas [Tub 84c]. Entre las conclusiones de este trabajo, interesa resaltar aquí que se observaron diferencias significativas en la velocidad de aprendizaje, comprension global y grado de creatividad en favor de los alumnos que aprendieron a programar en catalán, respecto a un grupo de control en inglés.

Perspectivas del Proyecto UBL

A la vista de los resultados obtenidos al utilizar UBL como herramienta pedagógica en nuestros cursos de Informática y en la experiencia de valoración comentada, creemos que sería interesante potenciar el desarrollo y utilización de "software" para la enseñanza de la programación en castellano. En este sentido, tenemos constancia de otros proyectos, como el lenguaje Merlín (desarrollado y utilizado en la Facultad de Informática de la U.P.B. [PBot 83]) y versiones castellanas de lenguajes foráneos (vease [PBot 84]).

Asimismo, creemos que sería interesante valorar la utilización de UBL como herramienta pedagógica en niveles educativos distintos del universitario. También podría estudiarse la utilización de otras lenguas. A este respecto, el lenguaje UBL esta preparado para la adaptación a otros idiomas con sintaxis similar a la del castellano.

Por otra parte, está realizándose un entorno de programación completo para el lenguaje, incluyendo un Editor Inteligente, y un Sistema de Depuración Interactiva; se piensa en construir traductores a otros lenguajes y una herramienta que permita la compilacion incremental.

También se estudia la posibilidad de implementar el lenguaje y su entorno en microordenadores.

Bibliografía

[ADA 83] *Reference Manual for the Ada Programming Languaje*, ANSI/MIL-STD-1815A-1983.

[Dijk 82] Dijkstra, E. W.: *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag 1982.

[Dijk 83] Orejas, F.; Llamosi, A.: *Tot fent el cafè amb el professor Dijkstra*, (Ciència), Noviembre 1983: 46-51.

[Gonz 84] Gonzalez, M: *Una informática en castellano para la enseñanza*, El Pais, 29 de Mayo de 1984.

[IBM 83] IBM: *VM/SP System Product Interpreter Reference Release 3*, SC24-5239-0.

[Lisk 77] Liskov, B.;Snyder, A.;Atkinson, R.;Schaffert, C.: *Abstraction Mechanisms in CLU*, CACM 20, no. 8, Agosto 1977.

[MacL 83] MacLennan, B. J.: *Abstraction in the Intel iAPX-432 Prototype Systems Implementation Languaje*, SIGPLAN Notices, 18, no. 12, Diciembre 1983.

[PBot 83] Botella, P.;Orejas, F. *Merlí: Report Preliminar*, Departamento de Programación, Facultad de Informática de Barcelona, 1983.

[PBot 84] Botella, P.: *Reflexiones Pedagógicas en torno a la Enseñanza de la Programación*, comunicación presentada en el 1er. Simposio sobre Informática y Educación, celebrado en Tucumán (Argentina).

[Shaw 77] Shaw, M.;Wulf, W. A.;London, R. L.: *Abstraction and Verification in Alphard: Defining and Specifying Iteration and Generators*, CACM 20, no. 8, Agosto 1977.

[Tub 84a] Tubau, E.;Sopena, J. M.;Blasco, J. M.;Sebastian, N.; Alonso, G.: *Valoración pedagógica de las opciones lingüísticas del lenguaje experimental UBL en la enseñanza de la programación*, comunicación presentada en las "Primeras Jornadas Nacionales sobre Informática en la Enseñanza" (Barbastro, 11-14 de Julio 1984)

[Tub 84b] Tubau, E.: *Psicología del Software: Factors Congnitius en l'aprenentatge i utilització de llenguatges de Programació*, Tesis de Licenciatura presentada en la Facultad de Psicología de la Universidad de Barcelona, Septiembre de 1984.

[Tub 84c] Tubau, E.;Sopena, J. M.;Blasco, J. M.;Sebastian, N.; Alonso, G.: *Aprender a Programar en la propia Lengua: una experiencia de valoración comparativa*, comunicación presentada en las "Jornadas Sobre 'Informática y Educación en la enseñanza Básica y Media'" , 26-28 Noviembre 1984.

[Wirth 82] Wirth, N.: *Programming in Modula-2*, Springer-Verlag, Berlin 1982.

Apéndice: Resumen de las caracteristicas del lenguaje UBL.

UBL es un lenguaje secuencial imperativo, en la tradicion de PASCAL. Describimos muy sucintamente sus principales caracteristicas, en su version castellana; para ello, utilizamos las convenciones sintacticas usuales: los corchetes indican opcionalidad, las llaves repeticion, y la barra eleccion (se escribiran estos caracteres entre comillas si forman parte del lenguaje, y no del metalenguaje).

Declaraciones

Pueden declararse variables, constantes, tipos, acciones, funciones, condiciones y secuencias. A diferencia del lenguaje PASCAL, las declaraciones pueden ocurrir en cualquier orden.

En las siguientes definiciones, 'id' representa un identificador.

- Variables

  [**var**] id {,id} : tipo;

- Constantes

  const id = expresion_constante;

  Una expresion_constante es una expresion que involucra solo constantes y operadores predefinidos.

- Tipos

  tipo id **es** tipo;

  Tambien pueden hacerse "declaraciones incompletas" del estilo "**tipo** T;" para resolver el problema de la doble recursion con pointers (que en UBL se llaman **nombres**), y "**tipo** T **es nombre** tipo;", que define T como el tipo cuyos valores son nombres de objetos de tipo 'tipo'.

- Acciones

  accion id [parametros] [**es** {declaracion}] **haz**
  {instruccion}
  fin [id];

  (Corresponden a los **procedure**s de Pascal)

Existen tres tipos de parametro: por copia, por nombre con acceso de lectura/escritura (**var**-parametros) y por nombre con acceso de solo lectura (**const**-parametros).

- Funciones

  funcion id [parametros]: id_tipo [**es** {declaracion}] **haz**
  {instruccion}
  fin [id];

  'id_tipo' es el identificador del tipo de valor que la funcion produce.

- Condiciones

  condicion id [parametros] [**es** {declaracion}] **haz**
  {instruccion}
  fin [id];

  Son funciones de tipo Logico; se introducen por motivos pedagogicos.

- Secuencias

  secuencia id [parametros]: id_tipo [**es** {declaracion}] **haz**
  {instruccion}
  fin [id];

Subprogramas que "producen" una secuencia de valores (utilizables con la instruccion **para** y el predicado **existe**); constituyen una forma de corutina. 'id_tipo' es el identificador del tipo de los valores que la secuencia produce.

Tipos

Ademas de declaraciones incompletas y tipos **nombre**, pueden definirse:

- Enumerados (ordenados, no ordenados y ciclicos).
- Subrangos (con la notacion "["min..max"]").
- Conjuntos
- Tuplas (o productos cartesianos; a veces llamados "records").
- Tuplas con variantes.
- Tablas y aplicaciones (o "arrays").
- Filas

Instrucciones

- Instrucciones simples:
  - asignacion (con el simbolo "←")
  - invocacion a una accion (no lleva parentesis para encerrar los argumentos)
  - **nada** (accion de efecto nulo)
  - **vale** (para dar valor a una funcion o condicion y "volver" a la rutina activante; utilizable solo desde una funcion o condicion)
  - **produce** (para producir un elemento de una secuencia y transferir el control al bucle activante; utilizable solo desde una secuencia)
  - **acaba** (una accion)
  - **sal** (de un bucle)
- Instrucciones de toma de decision
  - Instruccion **si**

```
si condicion entonces
  instrucciones
[sino
  instrucciones]
fin [si];
```

  - instruccion **decide**

```
decide
  cuando condicion = > instrucciones;
  {cuando condicion = > instrucciones;}
  [cuando otros = > instrucciones;]
fin [decide];
```

  - Instruccion **segun**

```
segun expresion es
  cuando valor_o_rango { "|" valor_o_rango} = >
    instrucciones;
  {cuando valor_o_rango { "|" valor_o_rango} = >
    instrucciones;}
  [cuando otros = >
    instrucciones;
fin segun;
```

- Instrucciones iterativas
  - Instruccion **repite**

```
repite
  instrucciones;
hastaque condicion;
```

  - Instruccion **mientras**

```
mientras condicion haz
  instrucciones;
fin [mientras];
```

  - Instruccion **itera** (debe utilizarse en conjuncion con una accion **sal**, **acaba**, **vale** o **produce**, para evitar la iteracion infinita)

```
itera
  instrucciones;
fin [itera];
```

  - Instruccion **para**

```
para variable en iteracion [talque condicion] haz
  instrucciones;
fin [para];
```

    "Iteracion" es el nombre de una secuencia, posiblemente seguido de parametros de la secuencia.

Expresiones

Son similares a las de PASCAL (+,-,*,/,**div**,**mod**,**y**,**o**,**no**,**en**), con la adicion del operador "||" para la concatenacion de tiras de caracteres y de las formas "short circuit" **y entonces** y **o sino**; la prioridad de los operadores esta racionalizada para poder omitir los parentesis en expresiones sencillas; se define un predicado

existe variable **en** iteracion [**talque** condicion]

que puede usarse libremente en las expresiones.

Un programa de ejemplo

El siguiente programa ordena las letras de una frase terminada por un punto, insertandolas en un arbol binario (y eliminando duplicados) y escribiendo luego el contenido de este arbol mediante un recorrido en inorden (que, como se vera, es formalizable muy adecuadamente mediante el concepto de secuencia).

```
programa arboles es
  tipo arbol es nombre
    tupla izq,der: arbol; cont: caracter; fin;
  secuencia inorden(a: arbol): caracter es
    var c: caracter;
  haz
    si a < > nulo entonces
      para c en inorden(a@.izq)  haz produce c; fin;
      produce a@.cont;
      para c en inorden(a@.der)  haz produce c; fin;
    fin si;
  fin inorden;
  accion mete(c: caracter; var a: arbol) haz
    decide
      cuando a = nulo = >
        crea a;
        con a@ haz izq ← nulo; der ← nulo; cont ← c; fin;
      cuando a@.cont < c = > mete c,a@.der;
      cuando a@.cont > c = > mete c,a@.izq;
      cuando otros = > nada;
    fin decide;
  fin mete;
  var a: arbol; c: caracter;
haz
  a ← nulo;
  Escribe_linea "Escribe una frase acabada por un punto:";
  itera lee c; sal cuando c = '.'; mete c,a; fin;
  para c en inorden(a) haz escribe_linea "Caracter: ",c; fin;
fin programa;
```