EL LENGUAJE SIMULA

0. ÍNDICE

EL LENGUAJE SIMULA

1. INTRODUCCIÓN

A mediados de los años 60 se empezó a vislumbrar el uso de las computadoras para la simulación de problemas del mundo real, estos problemas estaban llenos objetos normalmente muy complejos, los cuales eran difícilmente traducidos a los tipos de datos primitivos de los pocos lenguajes de la época.
Así fue como a partir de esta necesidad a dos Noruegos se les ocurrió el concepto de "OBJETO" y sus colecciones CLASES DE OBJETOS.
Nació así el lenguaje SIMULA, un lenguaje que contiene el embrión de lo que hoy se conoce como la PROGRAMACIÓN ORIENTADA A OBJETOS.

2. DESARROLLO Y CREACIÓN DEL LENGUAJE

Sus creadores fueron Kristen Nygaard y Ole-Johan Dahl del Centro Noruego de Computación en Oslo, y su desarrollo se extendió desde 1962 a 1967. El objetivo inicial era definir un lenguaje de propósito específico para aplicaciones de simulación.
De hecho, realizaron una primera versión, bajo contrato con la empresa UNIVAC, que no incluía conceptos novedosos desde el punto de vista de programación -aunque sí desde el punto de vista de simulación- con respecto al lenguaje más avanzado de esos años, Algol 60.
La versión de 1967 tenía como uno de sus objetivos ahorrar esfuerzo de programación. Nygaard y Dahl habían desarrollado grandes programas de simulación con la primera versión, y habían detectado dos deficiencias:

l. Las entidades proceso y estación, útiles en simulación, eran entes dinámicos que se creaban y destruían a lo largo de una ejecución. El Concepto de bloque derivado de **Algol 60**, era insuficiente para reflejar este dinamismo. Por otra parte, cada entidad tenía asociadas un conjunto de variables y un conjunto de operaciones que las manipulaban. Con lo que, el código del programa no reflejaba claramente esta relación.

2. El código de muchas entidades era bastante semejante, pero el lenguaje no proporcionaba un mecanismo que permitiera reutilizar las partes comunes.

El primer hallazgo de Nygaard y Dahl fue la distinción entre una clase de entidades -un texto suministrado por el programador- y los objetos que se derivan de ella -los ejemplares de la misma creados y destruidos dinámicamente a lo largo de una ejecución concreta. Una clase en Simula 67 consiste en una colección de procedimientos, asociados a un conjunto de declaraciones de variable. Cada vez que se crea un objeto de una clase, se asigna memoria para contener una colección de dichas variables. Esta idea, hoy familiar, exigía dos innovaciones con respecto a los lenguajes de la época:

- Escapar de la estructura de bloques. A diferencia de ésta, un objeto ha de "sobrevivir" al procedimiento que lo crea. Varios objetos de una misma clase han de poder coexistir en un mismo ámbito.

- Necesidad de un tipo de datos "referencia a un objeto" que permitiera designar objetos distintos en distintos momentos. Este tipo, llamado *ref* en el lenguaje, no era otra cosa que un ***puntero***.

Con el comenzaron a introducir abstracciones de datos a los lenguajes de programación.
Además pensaron en la posibilidad de comenzar a reutilizar código, con sus respectivas y oportunas modificaciones, así se comenzó a formar la idea de Jerarquías de HERENCIA DE CLASES.

Fueron ellos también los que introdujeron el concepto de POLIFORMISMO introducido vía procedimientos virtuales, idea que derivó en el concepto de PROMOCION NUMERICA y según la cual los objetos pueden ser clasificados o tipificados en una serie de súper clases de forma que se establece la Jerarquía de Clase.

Todas estas ideas revolucionarias fueron plasmadas en una primera versión del lenguaje SIMULA, a principios de los 60, alcanzando su madurez con una versión final en el año 1967, dando lugar a SIMULA67.

Aunque pensado como un lenguaje de propósito general, Simula tuvo su mayor excito en las aplicaciones de simulación discreta, gracias a la clase SIMULATION que facilitaba considerablemente la programación.

3. CARACTERÍSTICAS:

Simula fue, probablemente, el primer lenguaje que presentó las nociones de clase y herencia jerárquica. No se admite la herencia múltiple. El ocultamiento de información se lleva a cabo "protegiendo" una característica, lo cual, a su vez, evita que sea heredada en lo sucesivo.
Admite la sobrecarga de métodos.
La comprobación de tipos se puede realizar, bien estáticamente en el momento de la compilación, para mayor eficiencia, o bien en el momento de la ejecución si se define una característica como "virtual".

Problema:
El sistema necesita llevar a cabo la recolección de basura de vez en cuando, es decir, necesita reutilizar la memoria que fue ocupada por instancias asignadas anteriormente, pero que ya no se están utilizando. (Ralentiza los programas)

Hace más énfasis en los procesos que en los datos, pero Simula es responsable de muchas de las ideas que informaron a sus descendientes, sobre todo la idea de clases, instancias y herencia.
La idea de abstracción está implícita, pero las nociones, como el paso de mensajes, tuvieron que esperar la llegada de Smalltalk.
Simula es un lenguaje orientado a objetos, ***puro*** ya que no permite salirse de la orientación a objetos para realizar programación clásica.

Problema:
Cosa que le perjudicó ya que la comunidad científica dio la espalda a un lenguaje que no permitía ningún modo de programación imperativa clásica.

3.1. SIMULA VS. SMALLTALK

Ahora mostraremos un resumen de las características de Simula en comparación con su sucesor Smalltalk.

A favor de Simula67:

- Tenía una comprobación de tipos en tiempo de compilación, lo que permitía detectar errores de tipos. Smalltalk era un lenguaje sin tipos, por lo que no existía declaraciones de tipo de variables y no se realizaba ninguna comprobación estricta de tipos. Así los errores que surgen cuando un objeto recibe

un mensaje que no pueda resolver, se procesan todos en tiempo de Ejecución.

- Era un lenguaje ampliamente concurrente (debido a sus características innatas como simulador). En cambio Smalltalk permitía una concurrencia escasa.
- Separación Clase-Objeto. En Smalltalk todo eran Objetos, incluso las clases eran también Objetos.

En contra:

- Smalltalk no es solamente un lenguaje, sino todo un entorno completo de programación, que incluye numerosas librerías de clases, contemplándose las abstracciones básicas como colecciones o diccionarios, así como clases gráficas para el manejo de ventanas, barras de desplazamiento, etc. Es, por tanto, un lenguaje casi imposible de separar de su entorno de desarrollo. **En cambio Simula67 carecía de un entorno apropiado con sus características.**

- **Simula, no disponía de depuradores, visores y tampoco disponía de ninguna aplicación que permitiese visualizar su Jerarquía de clases**. Smalltalk si disponía de ello, lo que facilitaba el trabajo de los programadores. Consiguiendo una mejor difusión entre la comunidad de programadores que Simula.

- **La sintaxis en Simula, no era ni intuitiva ni sencilla.** Puesto que era un lenguaje tipado, tanto las variables como los atributos necesitan tener un tipo asociado. Esto unido a la falta de entornos de desarrollo y depuradores hizo aun mas dificultosa la tarea de programar en Simula.

Todo lo contrario pasaba en Smalltalk, su sintaxis era muy simple y uniforme, variables y atributos no necesitaban de tener un tipo y en principio TODO está definido como objeto, incluyendo a las propias clases.

3.2. RESUMEN: SIMULA VS. SMALLTALK

CARACTERISTICAS	SIMULA 67	SMALLTALK 81
Comprobación de tipos / ligadura	Si	Tardía
Polimorfismo	Si	Si
Ocultamiento de la información	Si	Si
Concurrencia	Si	Escasa
Herencia	Si	Si
Herencia múltiple	No	No
Recolección de basura	Si	Si
Persistencia	No	No
Genericidad	No	No
Bibliotecas de objetos	Simulación	Gráficas en su mayoría
Paso de Mensajes	No	Si

3.3. CONCLUSIÓN:

Pese a que Simula es casi 15 años mas antiguo que Smalltalk, observamos que salvo el Paso de Mensajes y la inclusión de algunas bibliotecas gráficas, el núcleo principal de características en Smalltalk es muy parecido al de Simula.

Entonces podemos preguntarnos… si Smalltalk fue un gran éxito, ¿por qué Simula siendo de similares características y con la ventaja de haber salir al mercado 15 años antes, no tuvo apenas éxito ni trascendencia en la comunidad internacional?

La respuesta a esta pregunta la intentamos dar mas adelante, pero una cosa hay que descartar, el poco éxito (o fracaso) de Simula no se debió a sus características como lenguaje en sí.

4. BREVE INTRODUCCIÓN SOBRE LA SINTAXIS EN SIMULA

- Definición de variables;
<tipoDeLaVariable> <nombreVble>;

Ejemplo:

```
array A,B,C;
integer m,n,p;
```

4.1 EJEMPLO DE UN PRODUCTO MATRICIAL

- Estructura de un procedimiento en Simula para multiplicar las matrices A y B y obteniendo el resultado en C:

```
procedure matmult (A,B,C, m,n,p);
        array A, B, C; integer m,n,p;
begin integer i,j,k;
        for i:=1 step 1 until m do
        for j:=1 step 1 until n do
        begin
                C[i,j]:=0;
                for k:=1 step 1 until p do
                C[i,j]:= C[i,j] + A[i,k] × B[k,j];
        end
end;
```

El mecanismo de paso de parámetros en Simula, se desvía un poco del paso de parámetros de ALGOL60. El modo por defecto de intercambio de parámetros es por VALOR para los tipos simples, y por REFERENCIA para los tipos Compuestos (no simples).

Por ello en el ejemplo visto las matrices A, B y C son pasadas por REFERENCIA, mientras que los enteros m, n y p se pasan por VALOR.

Aquellos procedimientos que son capaces de dar lugar a instancias las cuales perduran después de ser llamadas, son conocidas como CLASES; y sus instancias se conocen como OBJETOS de la CLASE.

Una clase puede ser declarada, con o sin parámetros de la siguiente forma:

```
<declaración de la clase> ::= class <identidad de la Clase>
        <Parte de parámetros formales (atributos)> ; <Parte de
especificación>
                <Cuerpo de la Clase>
<Cuerpo de la Clase> ::= < begin
                        ... código ...
                          end>
```

De esta manera:

```
a :- b    para la asignación a <- b.
a = = b        para la comparación.
a = / = b      para la diferencia.
```

- La notación "ref"

Para permitir hacer referencia a los objetos generados, es necesario de almacenar estas referencias en variables. Las variables usadas para este propósito deberían de ser declaradas como variables por referencia;

```
<reference variable declaration> ::=
        ref (<qualification>) <identifier list>
        <qualification> :: = <class identifier>.
```

La notación ref(<qualification>) puede ser también usada para declarar arrays, procedimientos y parámetros por referencia.
Puede ocurrir que la variable referenciada con el "ref" no haga referencia a ningún objeto, en cuyo caso valdrá "none".

NOTA: en realidad "ref" lo que hace es definir un **puntero** hacia un objeto de una clase, y cuando este puntero no apunta a nada su valor es "**none**", es como tener un puntero hacia "**null**".

Ejemplo:

```
class C(...); ... class body for C ... ;
ref (C)X;
        ...
        if X = = none then
                X :- new C(...);
```

Definida la clase C, se declara X como una variable que va a hacer referencia a un objeto de la clase C... si resulta que dicha variable no apunta a nada, entonces se crea un nuevo objeto (new C(...)) y se hace que X apunte hacia él.

Si X hace referencia a una variable de la clase C, y A es un atributo identificador de dicha clase, entonces X.A --> referencia al atributo A del objeto de la Clase C, al que apunta X.

Si cuando se intenta acceder al atributo A de X y resulta X tiene el valor "none", se produce entonces un error.

5. EJEMPLO: LA CLASE HISTOGRAMA

He aquí el problema:
Dada una sucesión de n números ordenados de menor a mayor se construyen n + 1 intervalos que se utilizan para clasificar y contar una sucesión de valores (observaciones) y así se obtiene el histograma de frecuencias relativas.
Desde la Estadística se aconseja que los intervalos sean de igual longitud y según la cantidad de observaciones se construya de 5 a 20
intervalos, aunque en el ejemplo siguiente nos apartamos un poco de este consejo.

Ejemplo: dada la sucesión X = {10,20,30,40} de 4 posiciones se construyen 5 intervalos
[0,10]; (10,20]; (20,30]; (30,40] y (40, + infinito).
Y dada la siguiente sucesión 5, 6, 12,43 (observaciones) se obtienen según X el siguiente histograma:

Intervalo	Frecuencia Relativa
[0,10]	50 %
(10,20]	25 %
(20,30]	0 %
(30,40]	0 %
Mayor a 40	25 %

En general podemos suponer que el límite inferior del primer intervalo es -infinito, por lo que dado un arreglo X de n posiciones digamos
X = {x0, x1, x2, x3, ..., xn-2, xn-1}
se construyen los n + 1 intervalos:
(-infinito, x0], (x0, x1], ..., (xn-2, xn-1] y (xn-1, +infinito).

Nota:
El enunciado original lo podemos encontrar en:
"Structured Programming"O.J.Dahl, E.W Dijkstra, C.A.R. Hoare
(página 181, 3.1. FREQUENCY HISTOGRAM)

Vamos a ofrecer una primera solución al problema en lenguaje Simula, para posteriormente obtener otra solución al mismo problema en un lenguaje más actual (en este caso Visual Basic .NET). En esta última solución se pretende elaborar un Histograma tal cual la formulación original de Dahl y Hoare pero en una plataforma la cual nos va a permitir, compilar y obtener resultados tangibles.

Nota:
(Habría sido interesante encontrar un compilador de Simula, pero no he encontrado ninguno).
Existe un compilador disponible GNU, pero no he conseguido bajarlo.
http://www.gnu.org/software/cim/cim.html
Cim es un compilador Simula, válido para máquinas UNIX.

5.1. Solución Simula

Este es un ejemplo que contiene a la clase Histograma, con sus métodos tabular(y) y frecuencia(i). Cada objeto es inicializado automáticamente cuando el método "new" es ejecutado en el código entre el último método y la frase 'end of histograma'
Los parámetros de entrada son una array 'X', y un entero 'n'. Esta inicialización es especificada justo después del nombre de la clase.

```
class histograma(X,n); array X; integer n;

begin integer N, integer array T[0:n];

procedure tabular(y); real y;
  begin integer i;
   i:=0;
   while ( if i<n then y<x[i+1] else false)
    do i:= i+1;
   T[i] := T[i]+1; N:=N+1;
end of tabular;

procedure frecuencia(i); integer i;
   frequency := T[i]/N;

integer i;
for i:=0 step 1 until n do
       T[i]:=0; N:=0;

end of histograma
```

Después podremos hacer llamadas en el main de la forma:

```
peso :- new histograma(B,12);
```

La variable peso, contiene el histograma generado por el array B, después la ejecución del constructor (new histograma(…)) .
después durante el cuerpo del programa principal puedo hacer llamadas a los métodos asociados a la clase de peso. (histograma).

```
peso.tabular(p);
```

El procedimiento "tabular" debe por lo tanto ser un atributo de la clase histograma. Otro atributo de la clase debe de ser el array T, el cual cuenta el número total de observaciones en casa intervalo; y también la variable N.
Finalmente la función "frecuencia(i)" se utiliza para leer la frecuencia relativa de observaciones en el intervalo i-esimo.

El programa principal:

```
begin ref (histograma) altura, peso, longitud;
      real array A[1:7],B[1:12];
      ... inicializamos A y B ...

      alturas :- new histograma(A,7);
      pesos :- new histograma(B,12);
      longitudes :- new histograma(A,7);

      ...

      pesos.tabular(p);
      alturas.tabular(t);
      longitudes.tabular(l);
      ...

end
```

NOTAS:

(1) En simula67 todos los parámetros simples de una clase o procedimiento son llamados por "VALOR". Arrays y otros parámetros mas complejos son llamados por "REFERENCIA".
(2) En SIMULA67, todas las variables son automáticamente inicializadas a un valor por defecto que depende del tipo. Así: las booleanas inicializan a "false", las numéricas a "0" CERO, y las variables referencias (punteros) a "none" (ó null).
(3) SIMULA no proporciona encapsulación. Los atributos internos de una clase pueden ser accedidos desde fuera de la clase.

5.2. SOLUCIÓN EN VISUAL BASIC .NET

Para ello definiremos primero una clase cHistrograma y luego utilizarla para crear un objeto de esta clase.

5.2.1 declaramos una clase cHistograma

```
Imports System
Imports Microsoft.VisualBasic
Public Class cHistograma
        Private mX(), mT() As Double, mF As Integer
        Public Event FueraDeRango(ByVal d As Double)
           'constructor parametrizado
        Public Sub New(ByVal x() As Double)
            'inicializamos todas las variables
            mX = x
            ReDim mT(x.Length)
            Dim i As Integer
            For i = 0 To UBound(mT)
                mT(i) = 0
            Next
            mF = 0
        End Sub
        Public ReadOnly Property X() As Double()
            Get
                Return mX
            End Get
        End Property
        Public Sub Tabular(ByVal Y As Double)
            If Y < 0 Then
                RaiseEvent FueraDeRango(Y)
                Exit Sub
            End If
            Dim i As Integer = 0
            Do While i <= UBound(mX)
                If Y > mX(i) Then
                    i += 1
                Else
                    Exit Do
                End If
            Loop
            mT(i) += 1
            mF += 1
        End Sub
        Public Function Frequencia(ByVal i As Integer) As Double
            Dim x As Double
            If mF > 0 Then
                Frequencia = mT(i) / mF
            Else
                Frequencia = x.MaxValue
            End If
        End Function
End Class
```

5.2.2 ahora la utilizamos para declarar un objeto de tal clase:

```
Imports System
Imports MicroSoft.VisualBasic
Module ObjetoHistograma
   Dim WithEvents longitudes As cHistograma
   Sub Main()
Dim a() As Double = {10, 20, 30, 40}, h As Double
Dim o() As Double = {5.5, 6, 12, 15, 26, 37, -1, 43}
     longitudes = New cHistograma(a)
           With longitudes
                 Dim i As Integer
                 For i = 0 To UBound(o)
                .Tabular(o(i))
                '.Tabular(-1) provoca el evento FueraDeRango
            Next
            i = 0
            Console.WriteLine("entre " + 0.ToString + " y " _
            + .X(i).ToString + " hay " + .Frequencia(i).ToString)
            For i = 0 To UBound(a) - 1
                Console.WriteLine("entre " + .X(i).ToString + _
                " y " + .X(i + 1).ToString + " hay " _
                 + .Frequencia(i + 1).ToString)
            Next
            Console.WriteLine("entre " + .X(i).ToString + _
            " y "  + h.MaxValue.ToString + " hay " _
            + .Frequencia(i + 1).ToString)
        End With
    End Sub
    Private Sub longitudes_FueraDeRango(ByVal d As Double) _
       Handles longitudes.FueraDeRango
        Console.WriteLine(d.ToString & " esta fuera de rango")
    End Sub
End Module
```

Observar en la salida siguiente la línea que produce el procedimiento de evento FueraDeRango al pretender tabular una observación negativa con la llamada al método cHistograma.Tabular(-1):

```
C:\WINDOWS\system32\cmd.exe
programa o archivo por lotes ejecutable.

C:\Histograma\SolucionSimple>objetohistograma
longitudes.X contiene:
longitudes.X(0)=10
longitudes.X(1)=20
longitudes.X(2)=30
longitudes.X(3)=40
Las observaciones son:
observación 0=5,5
observación 1=6
observación 2=12
observación 3=15
observación 4=26
observación 5=37
observación 6=-1
-1 esta fuera de rango
observación 7=43
entre 0 y 10 hay 0,285714285714286
entre 10 y 20 hay 0,285714285714286
entre 20 y 30 hay 0,142857142857143
entre 30 y 40 hay 0,142857142857143
entre 40 y 1,79769313486232E+308 hay 0,142857142857143
```

NOTAS:

(1) Para poder compilar estos ficheros fuentes, es necesario disponer de un compilador de Visual Basic .NET (fichero vbc.exe). Si estamos en una máquina Windows, lo podemos encontrar en C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727

(2) Localizado el compilador hemos de introducir la siguiente secuencia:

C:\vbc ClassHistograma.vb /reference:System.dll /target:library

Esto nos generar un archivo *.dll que contiene la claseHistograma.

C:\vbc ObjetoHistograma.vb /reference: System.dll, <path> classhistograma.dll /target:exe /verbose

Esto generará nuestro archivo ejecutable, nótese que es necesario pasarle como parámetro el archivo dll, obtenido en el anterior paso.

(3) Tanto el código fuente de los programas como el fichero resultante de la compilación de ambos están aquí.

6. Concurrencia de Simula

6.1 Breve introducción a la Concurrencia:

En muchas ocasiones, un sistema automatizado puede tener que manejar muchos eventos diferentes simultáneamente.
Un solo proceso – llamado hilo (thread) de control- es la raíz a partir de la cual se producen acciones dinámicas independientes dentro del sistema.
Todo programa tiene al menos un hilo de control, pero un sistema que admita concurrencia puede tener muchos de tales hilos (multithread): algunos son transitorios, y otros permanecen durante todo el ciclo de vida de la ejecución del sistema.

6.2 La Concurrencia en Simula:

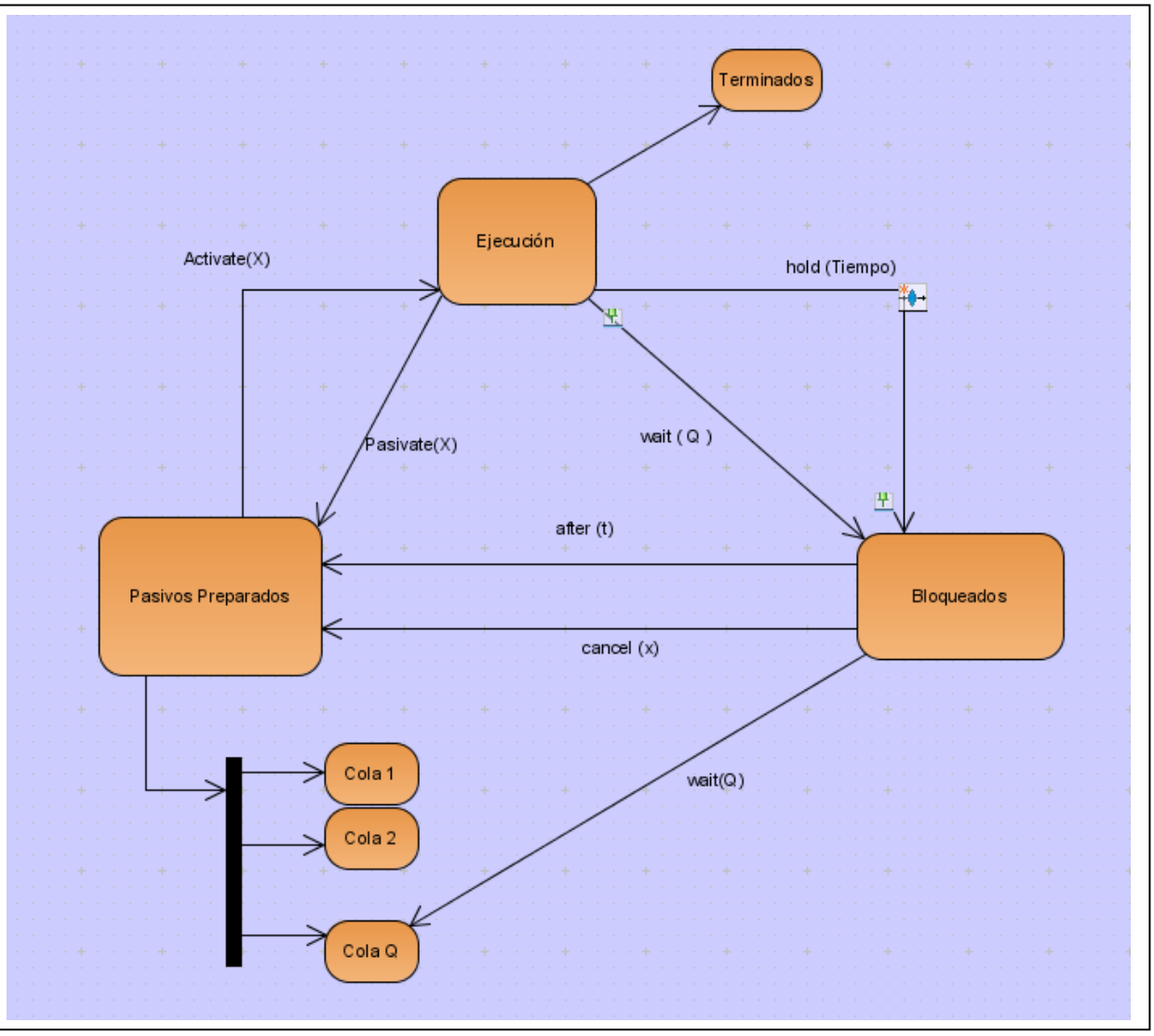
Simula es un lenguaje Concurrente, esto es permite en tiempo de ejecución la interacción simultanea entre varios “objetos”.

Su propósito inicial fue la simulación de sistemas físicos complejos con muchos cientos de componentes.

La simulación es un método para estudiar el comportamiento de objetos de muchos sistemas, y evaluar los efectos provocados de hacer experimentos sobre ellos, los cuales serían muy caros de hacerlos en la vida real.

Así, para representar procesos que ocurren de forma paralela en la vida real, no es necesario que las componentes sean multiprogramadas en el código del programa, pero sí es necesario que el programa pueda SUSPENDERSE a si mismo temporalmente, y que pasado un tiempo pueda DESPERTARSE y volver con su ejecución.
Los estados de un Objeto en Simula pueden ser:

- **En Ejecución** → el objeto esta en ejecución.
- **Preparado** → el objeto esta esperando en una cola para poder tomar el hilo de ejecución.
- **Suspendido** → el objeto ha sido bloqueado en el hijo de ejecución, cuando deje de estar bloqueado pasara a una de las colas de Preparados.
- **Terminado** → el objeto se ha terminado de ejecutar, se pasa a una zona donde el Recolector de Basura cuando lo estime oportuno pasará a destruirlo y liberar memoria.



En Simula, los módulos no se basan en procedimientos como en la programación convencional, sino en los objetos físicos que se modelan en la simulación. Estos objetos podrán ser "suspendidos" de la ejecución y "despertados", según lo necesite el programador.
Podemos decir que la concurrencia en Simula es pseudo-paralela.

En el anterior diagrama podemos ver una serie de comandos Simula utilizados para transitar de un estado a otro del objeto. Estos estados son:

- **Activate (x)** : manda el objeto de situado en la cabecera de la Cola de máxima prioridad en Preparados, hacia la EJECUCIÓN.
- **Passivate (x)** : es el contrario a Activate(x), es decir manda el objeto en ejecución hacia una de las colas en Preparados.
- **Hold (t)** : manda el objeto en Ejecución hacia los objetos bloqueados por espera, pasado un tiempo t, el objeto es enviado hacia las colas de Preparados.
- **Wait(Q)** : el proceso es enviado hacia el final de la cola Q de Preparados.
- **Cancel (X)**: el objeto X, es enviado a la cola de Preparados, pero se tendrá en cuenta que cuando dicho objeto sea llamado para activarse y pasar a Ejecución, el objeto volverá a ser enviado a la cola de Preparados, a la espera de que lo vuelvan a activar.

6.3 CONCLUSION SOBRE LA CONCURRENCIA

Esta característica de Simula, no fue suficientemente valorada por los investigadores de la época que sólo veían en ella no su funcionalidad sino su consecuencia, al tener muchos objetos al mismo tiempo interactuando hace que el sistema sea mas lento, llegando a saturar la memoria, ya que cuando dichos objetos dejan de utilizarse (pasando a Terminados), éstos siguen ocupando memoria… con lo que se limitaba con ello los recursos de la máquina.
La solución a esto pasaba por implementar de manera eficiente mecanismos recolectores de basura, en cambio éstos mecanismo si se ejecutaban con mucha frecuencia más que mejorar la eficiencia del sistema hacia que éste fuera más lento.

7. POLIMORFISMO & PROMOCIÓN NUMÉRICA

Profundizando en el tema los autores de Simula, se dieron cuenta que en ocasiones un método no tiene que ser propiedad exclusiva de una clase, esto es que en el mundo real, diversos objetos pueden responder al mismo mensaje de una forma distinta… nació así el concepto de POLIMORFISMO, palabra que viene del griego y significa "muchas formas".
La idea básica es que cada objeto puede tener una respuesta única al mismo mensaje. A partir surgió otro concepto importante incluido en el lenguaje SIMULA, y fue la "PROMOCION NUMERICA" según la cual los objetos pueden ser clasificados o tipificados en una Jerarquía de clases de forma que se facilitaba el realizar unas mismas operaciones entre objetos distintos que perteneciera a una misma jerarquía.

A veces, una simulación implica solamente un ejemplo de una clase particular de un objeto. Sin embargo es mucho más común, necesitar más de un objeto de cada tipo. Esta posibilidad levanta otra preocupación: seria extremadamente ineficaz redefinir los mismos métodos en cada ocurrencia de ese objeto…
De nuevo los autores de Simula, aplicaron una solución elegante:
LA CLASE, de forma que una clase fuera una plantilla de Software que define los métodos y variables que se incluirán en un tipo particular de objeto. Así los métodos y las variables que hacen el objeto se definen solamente una vez, en la definición de la clase.

8. CONCLUSIÓN FINAL

¿PORQUÉ SIMULA NO GOZÓ DE UNA AMPLIA IMPLANTACIÓN?

A lo largo de todo el trabajo se han ido exponiendo características y detalles del lenguaje, que pese a ser innovadoras producían ciertos efectos laterales no deseados.
Pese a ello no fue esto lo que impidió la amplia difusión de Simula, sino una serie de pequeños detalles:

Kristen Nygaard (KN), fue el padre de Simula se encargó del desarrollo teórico del lenguaje. KN no tenía pensado crear un compilador para su lenguaje hasta que éste estuviese en un alto nivel de desarrollo. Cuando esto ocurrió un experto programador, Ole-Johan Dahl (OJD) se puso en contacto con KN para llevar a cabo un compilador para el lenguaje.

Tanto KN como OJD trabajaban inicialmente en el NDRE (Norwegian Defence Research Establishment) esta institución contaba con máquinas muy antiguas entre ellas una Ferranti MERCURY, sobre las que ambos trabajaron durante años en el desarrollo de simulaciones reales, sin embargo para embarcarse en su nuevo proyecto esta máquina resultaba inadecuada.
Por este y otros motivos a principios de los 60, KNygaard (primero) y luego OJDahl abandonaron la NDRE para pasar a la NCC (Norwegian Computing Center), esta era una institución semi-gubernamental.
En 1963 la NCC incorporó una máquina puntera en tecnología de aquella época era una UNIVAC 1107. El verano anterior Nygaard había estado de gira por Estados Unidos promocionando estas nuevas maquinas, con la doble intención de promocionar el lenguaje SIMULA sobre estas maquinas **(cuando en realidad por aquella fecha aun no habían desarrollado ningún compilador sobre SIMULA)**.
En este tour por EEUU los dirigentes de Univac apalabraron con Nygaard, que si completaban rápidamente un compilador de SIMULA para acoplarlo a su maquina UNIVAC 1107. Ellos podrían ofrecer un importante descuento de esta máquina para el instituto donde trabajaba Nygaard (NCC). El acuerdo se llevo a cabo, con la promesa tanto de Nygaard y de la NCC de completar lo más rápidamente posible este compilador…
No fue hasta pasados 2 años, en Enero de 1965 cuando los acuerdos entre NCC y Univac dieron como resultado el primer compilador para SIMULA I. Durante su desarrollo **el ambiente que existía en el equipo era muy complicado, a menudo agobiante y todo esto dio lugar a heridas entre componentes del NCC de difícil curación.**
Había desaparecido por completo el entusiasmo inicial, y finalmente los dirigentes de la NCC le dijeron a Nygaard y Dahl que:

- ***Allí nunca entraría en uso un lenguaje como SIMULA.***
- ***Allí se podría haber usado SIMULA, si el compilador se hubiese hecho en menos tiempo.***
- ***Sus ideas no eran suficientemente buenas, que ellos (la NCC) carecían en general de la capacidad para emprender tal proyecto, y que nunca podrían terminar tal tarea.***
- ***Un trabajo de esta naturaleza, debería ser hecho en un país con más recursos económicos destinados a tal propósito, y no en un país pequeño como Noruega.***

A pesar de todos estos problemas y contratiempos, el compilador de SIMULA fue completado e incorporado al UNIVAC 1107 aunque debido al retraso de 2 años, los dirigentes de Univac se

vieron obligados a introducir a SIMULA dentro del "paquete B" de software. Siendo el "paquete A" un compilador de ALGOL60 y pasando SIMULA a ser un PRE-procesador para el compilador de Algol.

Pasado un tiempo y después de todo el empeño posible por parte de Nygaard y Dahl… Univac incluyo a Simula dentro de su paquete A (Categoría I), con lo que Univac distribuía el compilador pero era la NCC la encargada de mantenerlo.

SIMULA I, fue originalmente solo considerado como un lenguaje para la descripción y simulación de sistemas, pero no como un lenguaje de programación de propósito general. Por ello pasado un tiempo (1967) de nuevo Nygaard y Dahl, comenzaron a elaborar un nuevo diseño sobre un lenguaje de propósito general: SIMULA 67.
Este nuevo proyecto, no fue visto con buenos ojos desde algunos dirigentes del NCC, sobre todo por cuestiones económicas, el NCC era una entidad no muy grande de unas 120 personas y el departamento que investigaba sobre el SIMULA 67, lo formaban no más de 10 personas.

Se pretendía que este lenguaje fuese implantado en las estaciones de trabajo de la época, sobre todo en las primeras máquinas de IBM, sin embargo competir con el departamento que desarrollaba software de IBM, era una tarea imposible pues estaba formado por más de 500 personas. Con lo que IBM que era por aquella fecha el principal vendedor de Hardware dio de lado a SIMULA 67, con lo que éste quedo relegado a usarse en máquinas poco difundidas, y su trascendencia internacional fue mucho menor de la esperada inicialmente.

No solo fue poca su implantación sino que además los beneficios económicos fueron eran muy bajos, casi insuficiente para mantener los gastos económicos de los desarrolladores del NCC.

Debido a esto un lenguaje que supuso una primera aproximación hacia el "extendido" (hoy día) modelo de objetos, no tuvo la repercusión ni el reconocimiento adecuado de la comunidad científica de la época.

Pese a ello, otro problema que tenia SIMULA 67, era que a la hora de ser interpretado por máquinas que seguían la Arquitectura de Von Neumann, se producía un excesivo manejo dinámico de memoria debido a la constante creación de objetos, así como una excesiva carga de código fuente, causada por la constante invocación de métodos, lo que provocaba la sobrecarga de trabajo y la consiguiente lentitud en éstas máquinas.

No fue hasta los años 80 cuando dichas ideas fueron rescatadas y comenzaron a utilizarse en lenguajes como SMALLTALK, C++…

Pese a ello podemos decir que hoy día el modelo de objetos introducido por primera vez en el lenguaje Simula, esta ampliamente extendido y goza de un gran auge gracias a la popularidad de

lenguajes como JAVA. (Que lo han tenido todo de cara para expandirse, con el nacimiento de Internet….)

9. BIBLIOGRAFIA:

- ✓ "History of Programming Languages" por Richard L. Wexelblat.
- ✓ "Structured Programming" por OJ.Dahl, EW. Dijkstra and CAR. Hoare.
- ✓ "Metodologías orientadas a objetos" por Rafael Barzanallana (Universidad de Murcia)
- ✓ "Introducción al lenguaje Haskell" por Jose.E. Labra G (Universidad de Oviedo)
- ✓ Tesis Doctoral "Computación de Procedimientos vs. Computación Matemática", Javier Fdez Pacheco.
- ✓ "Lenguajes de Programación" por Hanna Oktaba
- ✓ "Lenguajes de Programación: Motivación" por Fco J. Serón Arbeloa, Juan Antonio Magallón, Sandra Baldasarri (Universidad de Zaragoza)
- ✓ "Aplicación del lenguaje Visual Basic .NET a la estadística" por Juan Pedro Esponda.