
 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p>PARADIGMAS DE PROGRAMACIÓN</p>
--	---	-----------------------------------

## PARADIGMAS Y LENGUAJES DE PROGRAMACIÓN

### I - Introducción



La disciplina de programación de computadoras ha sido por muchos años un lugar de oportunidad para el desarrollo del intelecto humano al tratar de describir la realidad en términos computacionales. Para llevar a cabo dicha tarea existen diversos lenguajes, los cuales están basados en una serie de premisas o postulados que definen las características y limitantes de los mismos. Hablar sobre lenguajes de programación requiere de analizar diversos conceptos. En primera instancia, es necesario definir los principios que definen a las diversas formas de programar, esto es hablar de los *paradigmas*. En segundo término es necesario comentar sobre *la forma de instrumentar los diversos lenguajes*. En tercer término, resulta necesario comentar sobre *las aplicaciones, ventajas y desventajas de cada uno de los lenguajes existentes*. Este primer artículo está dedicado a dar una breve presentación de los principios que originan a los diversos lenguajes de programación, usando para ello el concepto de **paradigma**. Adicionalmente, se discuten algunos aspectos de la instrumentación. En el artículo complementario se presentan detalles adicionales de instrumentación, así como un estudio comparativo de las aplicaciones, ventajas y desventajas de los paradigmas y sus lenguajes representativos.

Cabe mencionar que estas notas asumen un conocimiento básico de programación procedural por lo que la misma es poco discutida. Por la novedad del tema, en específico para el caso de la programación orientada a objetos, se ha dedicado una sección especial al tópico. Los paradigmas a mencionar en el artículo son:

- a. Lenguaje de Máquina y ensamblador
- b. Procedural
- c. Orientado a Objetos
- d. Programación Funcional
- e. Programación Paralela
- f. Herramientas para Aplicaciones

El concepto de paradigma fue introducido por Thomas Kuhn, en su famoso estudio sobre la forma como avanza el pensamiento científico. Dentro de este contexto un paradigma es más que un simple modelo de pensamiento, es toda una visión distinta de ver la realidad. En este contexto, José Negrete define en forma sencilla un paradigma como (2): *Un paradigma es una regla cualitativa que agrupa y/o integra a un conjunto de reglas cuantitativas.*

Visto de esta forma un nuevo paradigma es más que un nuevo modelo, es toda una forma distinta de pensar que nos permite resolver un problema en forma diferente. En su libro Kuhn, comenta que las ciencias exactas para avanzar han tenido que cambiar la forma tradicional de pensar de cada época para poder explicar fenómenos antes no explicados o

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

no conocidos; lo anterior significa cambiar los postulados o reglas cuantitativas de las teorías en boga, generando con esto un nuevo paradigma de pensamiento.

Para el caso de la programación, en su muy corta vida (menos de 50 años) ha habido diversas formas de pensar, esto es, formas de concebir un problema y programar su solución.

Por otro lado, es importante mencionar que solo se cambia de forma de pensar, si la forma actual no funciona satisfactoriamente para todas las aplicaciones, tanto las actuales como las nuevas. Casualmente para poder cambiar de forma de pensar se requiere estar dispuestos a eso y para ello a veces es conveniente no estar completamente amarrado a un tipo específico de forma de pensar. De aquí que es común que los cambios de paradigma los den las personas que se encuentran en las fronteras del paradigma y no en el corazón del mismo.

Para el caso de la programación, el paradigma más utilizado y duradero ha sido el de la programación procedural, la cual data de la creación de Fortran en 1957 (3); sin embargo existen algunos problemas no resueltos dentro de este paradigma; de aquí la necesidad de moverse a otros paradigmas. Específicamente estos problemas tienen que ver con:

- Corrección: La habilidad del software de hacer correctamente el propósito para lo que fue creado.
- Robustez: La habilidad del software para trabajar en circunstancias fuera de lo normal.
- Extensión: Capacidad y facilidad de adaptación del software a cambios en las especificaciones.
- Reusabilidad: Habilidad del software para ser usado, total o en partes, en otros diseños y paquetes.
- Compatibilidad: Capacidad de correr bajo diversos ambientes.
- Modularidad: Todo software debe ser diseñado en forma modular.

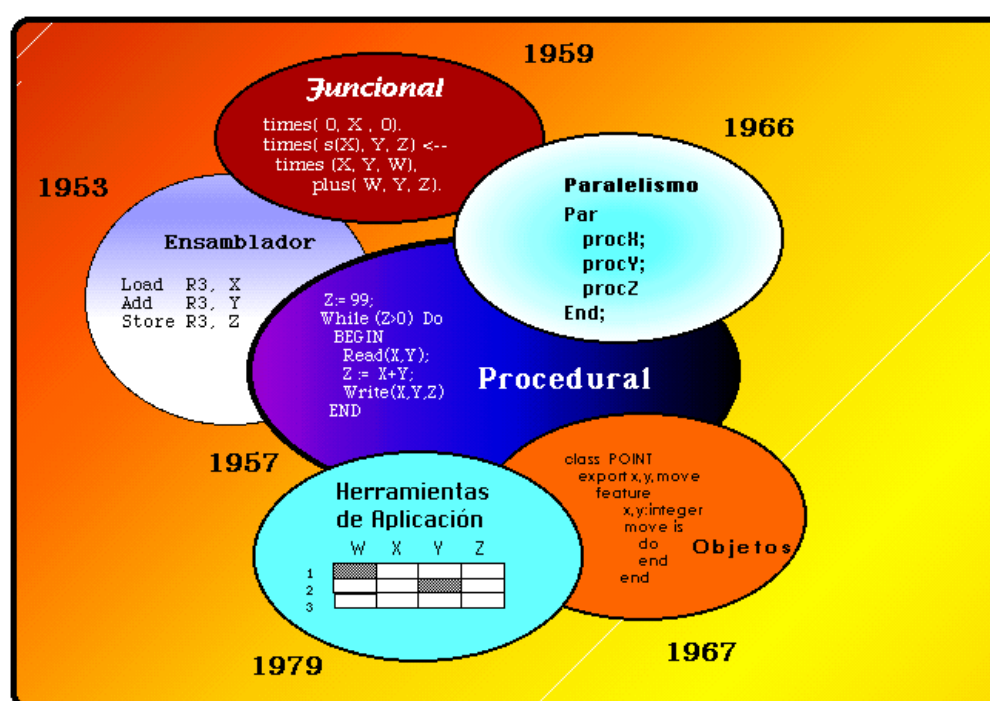
Resulta necesario mencionar que para la instrumentación de los postulados de un paradigma de programación se requiere contar con una infraestructura básica en el hardware que permita la ejecución adecuada (rápida, eficiente, óptima) del mismo. Específicamente las herramientas que permiten la instrumentación de los diversos paradigmas de programación son el uso extensivo de la fase de interrupción de un CPU, el manejo dinámico de la memoria (Heap) y la existencia de los ambientes de programación residentes. Para el caso del paralelismo es necesario contar con varias unidades lógico aritméticas ( ALU's ) para instrumentarlo.

Un punto muy importante es que aunque normalmente se asocia un paradigma de programación a una sintaxis específica, inclusive a un lenguaje en particular; la forma de pensar es en sí ajena a la sintaxis, siendo más bien un problema de semántica. Esta es precisamente la razón por la que en algunos lenguajes típicamente procedurales es posible programar usando un paradigma distinto (este es el caso de PASCAL con objetos ó C con

ensamblador ). Por otro lado, lo anterior no quiere decir que una interfaz adecuada no sea importante; así por ejemplo, para el paradigma orientado a objetos el uso de estaciones gráficas y multimedios es parte importante para la difusión y uso del paradigma:

## II. Postulados

A reserva de comentar algunos detalles más adelante, en las tablas I a VI se presentan en forma resumida la descripción de los postulados de los diversos paradigmas en cuestión. La figura siguiente muestra la interrelación de estos Paradigmas





*Interrelación entre los diversos Paradigmas de Programación*

**Tabla I**

### Postulados de la programación en ENSAMBLADOR

MNEMONICOS, TRANSFERENCIAS, DIRECCIONES = EJECUCION

- 1) 1-1 Una instrucción mnemónica por una de máquina ( Load REG 3, ALFA <==> OFFAD893 )
- 2) DIRECCIONES SIMBÓLICAS El programador indica las posiciones de memoria usando símbolos ( ALFA <==> M( 1AD893 ) )
- 3) SECUENCIAL La siguiente instrucción está en CAR+1
- 4) DOS OPERANDOS El 3er operando es uno de los dos primeros ( A = A + B )

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

- 5) Go To CONDICIONAL Uso de transferencia para modificar la secuencia ( BR ONZ, RUTINA\_DOS )
- 6) ESTATICO Compilación, Ligado y Cargado a priori
- 7) Apuntadores Acceso a estructuras de datos por medio de direccionamientos a memoria ( DATO = M [ CIR(2nd OP) + REG. IND ] )
- 8) NO MODIFICACION DEL CODIGO EN EJECUCION El programa no modifica su código ( instrucciones ) durante su ejecución
- 9) TIPOS BASICOS DE DATOS (Entero, Real, Alfanumérico (ASCII) )

(Ejemplos: IBM360, PDP11, INTEL286, MOTOROLA68020, SPARC)

## Tabla II

### Postulados del Paradigma Procedural

ESTRUCTURAS DE DATOS + ALGORITMOS = PROGRAMAS

- 1) 1- m Una instrucción por muchas de máquina (  $c := a + b \iff$  Load, Add, Store )
- 2) SECUENCIAL Las instrucciones son ejecutadas en forma secuencial
- 3) ASIGNACIÓN Fundamenta su ejecución en asignaciones y fórmulas (  $A := A + B$  )
- 4) MODULAR. La programación se lleva a cabo modularmente, usando para esto procedimientos y funciones.
- 5) ESTRUCTURADO Utiliza estructuras de transferencia definidas ( Do While, If Then Else, Case, Call's )
- 6) ESTATICO Compilación, Ligado y Cargado A Priori
- 7) TIPOS DE DATOS DIVERSOS (Entero, Real, Alfanumérico(ASCII), Enumerables, Matrices, Records)
- 8) NO MODIFICACIÓN DEL CÓDIGO EN EJECUCIÓN El programa no modifica su código (instrucciones ) durante su ejecución
- 9) APUNTADORES LIMITADOS En la medida de lo posible el programador deberá usar Apuntadores cuidadosamente
- 10) NO SE ACONSEJA EL USO DEL Go To



(Ejemplos: FORTRAN, ALGOL, BASIC, PASCAL, C, PSEINT)

## Tabla III

### Postulados del Paradigma Orientado a Objetos

RECORDS + MÉTODOS = OBJETOS

- 1) ORIENTADO A OBJETOS Los sistemas se modularizan en función de los datos.
- 2) OBJETOS COMO DATOS Los objetos son instrumentados como un tipo de dato complejo.
- 3) OBJETOS DINÁMICOS Los Objetos son copias de una clase que dinámicamente usan el HEAP.
- 4) CLASES Todo tipo de dato complejo es un módulo, generando una clase específica.

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

- 5) HERENCIA Toda clase puede ser definida en términos de de otra(s), heredando sus definiciones.
- 6) POLIMORFISMO Los elementos de un programa pueden llamar a diferentes clases con el mismo nombre. Una clase puede tener varias formas y las funciones implementarse en forma distinta
- 7) MENSAJES Los objetos se comunican entre sí, por medio de mensajes, los cuales son interpretados por cada objeto a su manera.
- 8) ENCAPSULAMIENTO En la medida de los posible las variables y métodos de un objeto son ajenos y desconocidos para los demás.
- 9) PERSISTENCIA Los objetos residen en memoria, requiriéndose adicionalmente un almacenamiento en I/O.
- 10) MÉTODOS La programación de los métodos asociados a los objetos, se efectúa usando postulados procedurales
- 11) AMBIENTE DE PROGRAMACION Operan bajo un residente en ejecución de corte interactivo, interpretativo y con ligado retrasado
- 12) INTERFAZ DE MULTIMEDIO Es común que operen en un ambiente gráfico y con uso de medios múltiples de I/O.

(Ejemplos : SIMULA, SMALLTALK, HYPERTALK,C++,EIFFEL, Java )



#### **Tabla IV**

#### **Postulados del Paradigma Funcional**

FORMAS FUNCIONALES + EXPRESIÓN DE EXPRESIONES = APLICACIONES

- 1) EXPRESIONES Una expresión es el resultado del valor de sus subexpresiones y los operadores correspondientes.
- 2) RECURSIVIDAD Uso extensivo de recursividad
- 3) FUNCIONES PRIMITIVAS El lenguaje cuenta con un conjunto poderoso de funciones primitivas
- 4) SINTAXIS SENCILLA Uso de pocas reglas formales para su sintaxis.
- 5) EQUIVALENCIA ENTRE FUNCIONES Y DATOS Las funciones y los datos tienen características similares (valor, argumento de rutinas, estructura)
- 6) AMBIENTE DE PROGRAMACION Operan bajo un residente en ejecución de corte interactivo, interpretativo y con ligado retrasado
- 7) MODIFICACIÓN DEL CÓDIGO EN EJECUCIÓN Por su naturaleza, permite la modificación de datos y expresiones durante la ejecución
- 8) ESTRUCTURA DE PROGRAMACIÓN LIMITADA En general se requieren de pocas estructuras de programación secuencial, condicional y cíclica.
- 9) MEMORIA DINÁMICA El residente maneja dinámicamente la memoria sin intervención del programador ( Garbage Collection )

(Ejemplos : LISP, PROLOG, SNOBOL, ML, SCHEME)

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

## Tabla V

### Postulados del Paradigma de la Programación Paralela

INDIVIDUAL(Instrucción) + MÚLTIPLES(DATOS) = PROGRAMAS

- 1) SIMD Una instrucción puede ser ejecutada al mismo tiempo en varios datos, matricialmente (  $C = A + B \Leftrightarrow c_{ij} = a_{ij} + b_{ij}$  )
- 2) SISD Una instrucción puede ser ejecutada en forma escalar tiempo en varios datos. (  $C = A + B \Leftrightarrow c = a + b$  )
- 3) PARALELISMO Las instrucciones (procesos) pueden ser ejecutadas en forma concurrente o secuencial
- 4) NO DETERMINISMO Por la naturaleza de los procesos paralelos puede generarse una ejecución no determinística.
- 5) PROGRAMACIÓN La programación de los procedimientos y funciones dentro de un bloque, se desarrolla usando postulados procedurales y estructuras de datos adecuadas
- 6) SINCRONIZACIÓN Procesos paralelos deben coordinar sus accesos a los mismos recursos (memoria, disco, I/O)
- 7) ESPERA AND/OR Un bloque paralelo puede terminar cuando todos los subprocesos terminen o bien cuando uno de ellos lo haga.



(Ejemplos : OCCAM, PASCAL CONCURRENTE, ADA)

## Tabla VI

### Postulados de las Herramientas para Aplicaciones

MENÚS, BOTONES, FORMATOS Y/O EJEMPLOS = APLICACIONES

- 1) ORIENTACIÓN A APLICACIONES Existen tantos paquetes como aplicaciones distintas de deseen.
- 2) EJEMPLOS En algunos casos, la entrada de datos y funciones se efectúa por medio de ejemplos
- 3) BOTONES Es común que la selección de funciones se efectúe por medio de ejemplos
- 4) MENÚS En algunos casos , la selección de funciones se efectúa por medio de ejemplos
- 5) FORMAS En algunos casos, la entrada de datos y funciones se efectúa llenando formas predefinidas
- 6) USO DE OBJETOS En general se heredan los postulados del paradigma orientado a objetos
- 7) ACCESO ALEATORIO El direccionamiento de datos y la secuencia de ejecución son efectuados por el usuario en forma aleatorio, y no forzosamente secuencial.
- 8) AMBIENTE DE PROGRAMACIÓN Operan bajo un residente en ejecución de corte interactivo, interpretativo y con una poderosa herramienta de apoyo y asistencia.
- 9) INTERFAZ DE MULTIMEDIO Operan en un ambiente gráfico y con uso de medios múltiples de I/O.

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

(Ejemplos: HOJAS DE CÁLCULO, CAD, DRAW, PAINT, HYPERCARD)

### III. Características de los lenguajes de programación

Los lenguajes de programación constituyen la instrumentación de las reglas cualitativas de un paradigma, estos están sujetos a una serie de características generales que pueden clasificarse como sigue:

1. Tipos de Datos
2. Estructuras de Control
3. Manejo del I/O
4. Operaciones, Operadores y Operandos
5. Sintaxis
6. Esquema de Traducción y Ejecución

#### 1) Tipos de datos

En todo lenguaje existen tres tipos niveles de tipos de datos: Máquina, Lenguaje, Usuario (definidas por el mismo). Entre mayor nivel tenga el tipo del dato, más fácil es su uso. Por otro lado, entre más elemental sea el tipo del dato resulta más eficiente en términos de memoria y tiempo de ejecución.

#### Nivel Máquina

APUNTADOR = Representa una dirección (o zona) en memoria

REGISTRO = Representa un registro o acumulador en el CPU

ENTERO = Número Entero en 2 B (  $2^{15}$  ) ó 4 B (  $2^{32}$  ).

REAL = Número Exponencial en 4, 8 ó 16 Bytes.

ASCII = Caracter alfanumérico 1B ( 256 elementos )

BOOLEANO = Carácter booleano (t,f) en 1bit ó 1Byte

NBCD = Carácter decimal en 4 bits (nibble)

#### Nivel Lenguaje

Varía considerablemente para cada lenguaje, algunos de ellos son:

- CADENA (n) = Cadena de caracteres ASCII de tamaño n, puede ser dinámica o estáticamente asignada
- ARREGLO (n,...,m) = Manejo de vectores y matrices de otros tipos
- TABLA (n) = Tabla de acceso Hash de n elementos
- ENUMERABLES = Asignación de mnemónicos en una secuencia enumerable (256 elementos).
- SUBRANGO = Selección de un rango de números enteros
- SET = Similar al enumerable pero para uso lógico
- RECORD = Estructura formada por varios tipos diversos con un solo nombre (una dirección base en memoria).
- FILE = Similar a CADENA asociado con un manejo de I/O



 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

Relacionado con el tipo de datos, se asocian los conceptos de:

**Coerción:** Los datos son convertidos obligatoriamente al tipo requerido por la operación

**Overloading:** Cuando un tipo de dato, operador o función puede tener distintos significados de acuerdo al contexto en que se encuentre

**Polimorfismo:** Cuando una estructura de datos básica puede tener formas distintas en su definición final (un ejemplo elemental es el Case en un Record, sin embargo puede ser muy complejo). Otros tipos de datos más específicos son:

LISTAS = Secuencia de Apuntadores en forma ligada

PATRONES = Similar a CADENA solo que se usa para comparar valores y contenidos en cada CADENA

FUNCIONES = Asocian una relación o función lógica

CLASES = Similares a RECORD, solo que incluyen en su definición datos, rutinas y comunicación (DB/DP/DC), asociados a una dirección de memoria (normalmente residen en el HEAP)

Nivel Usuario



Adicionalmente el usuario puede definir sus propias estructuras de datos, combinando para esto los diversos tipos de datos anteriormente mencionados. ¿Qué tal un : Apuntador de un Arreglo de Record de un Subrango de enteros con subíndices Enumerables?

## 2) Estructuras de Control

Un aspecto importante en un lenguaje de programación es su estructura de control, esto es la forma como organiza la ejecución de las instrucciones del usuario. Esta organización es independiente de la forma y las operaciones que forman en sí las instrucciones del lenguaje. En realidad es la forma de la estructura de control que identifica al paradigma de programación en cuestión. Así por ejemplo la estructura procedural define al paradigma; en general podemos decir que toman las siguientes formas:

1. Secuencial ==> BEGIN / END
2. Condicional ==> IF THEN, IF THEN ELSE
3. Incondicional ==> GO TO
4. Predicados ==> AND, OR, CUT, NEGATION, BACKTRACKING
5. Cíclica ==> WHILE, REPEAT UNTIL, FOR
6. Modular ==> FUNCTION, PROCEDURE
7. Interrupciones ==> ON CONDITION, EXCEPTION HANDLING
8. No Secuencial ==> CASE OF, RECURSIVE, LISTS
9. Por Eventos ==> MESSAGES, INTERLEAVED
10. Concurrencia ==> PAR/END, PAR/ALT/END



 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

### 3) Manejo del I/O

Otro elemento de los lenguajes de programación es el manejo del Input/Output, el cual se clasifica de acuerdo al medio usado:

Entrada de texto: Stream Oriented, EOF, EOL

Impresión: DISPLAY, TYPE

Manejo del Disco: Virtual Memory, Overlays

Manejo de Archivos: SEQ, RANDOM, INDEX, HASH

Manejo de Multimedia: Mouse, VCR, etc.

Es importante en la medida en que pueda la información se almacene y recupere en una forma compatible con el esquema de operación del lenguaje. Así por ejemplo, para una lengua matemática es importante tener un manejo transparente de matrices en disco y para un manejo de objetos, sería conveniente tener un manejo sencillo de persistencia de los objetos en disco.

### 4) Operaciones, Operadores y Operandos

Se mencionan a continuación las operaciones y su ejecución en lenguajes artificiales (estas pueden requerir de cero, uno, dos ó múltiples operandos):

- 1) Tipo de Máquina Triples de Máquina ( Load Reg3,Alfa)
- 2) Tipo Aritmético - (unario), -, +, \*, /, \*\*, INC, DEC
- 3) Tipo Comparación >, <, >=, <=
- 4) Tipo Lógico AND, OR, NOT
- 5) Tipo Utilería Asignación (=, :=, <-)
- 6) Tipo I/O Open, Close, Get, Put, Read, Write, Seek, Find
- 7) Tipo Texto Concatena ( | |, b, +) , Decatena, SUBSTR, Alterna( | )
- 8) Tipo Misceláneo New, Dispose, Cons, Append, CAR, CDR, CUT, etc.

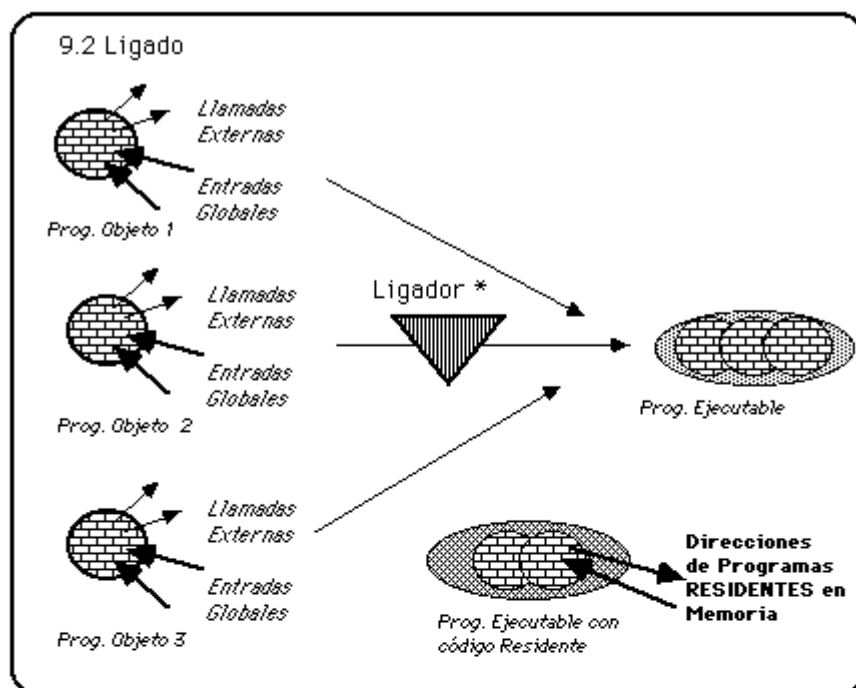
### 5) Sintaxis

Tal vez el componente más característico de un lenguaje de programación es su sintaxis, está invita al uso del paradigma de una manera natural. Así por ejemplo PASCAL, usa una sintaxis estructurada, mientras que LISP tiene una sintaxis más libre pero de tipo funcional. PROLOG por su parte tiene una notación formal que invita a la abstracción y al cálculo de predicados. Finalmente los lenguaje orientados a objetos (como Smalltalk) combinan una sintaxis en texto con un manejo de iconos que representan en croquis ( e inclusive, imágenes completas ) los objetos en cuestión, junto con esto, el uso de multimedia para el pasaje de mensajes de I/O invita el pensamiento en términos del paradigma en forma sencilla (Hypercard). Una vez más es importante recordar que la sintaxis del lenguaje aunque necesaria, no define al paradigma, éste está identificado por los postulados correspondientes. Adicionalmente, se puede argumentar que la programación en términos gramaticales representa un paradigma adicional de programación.

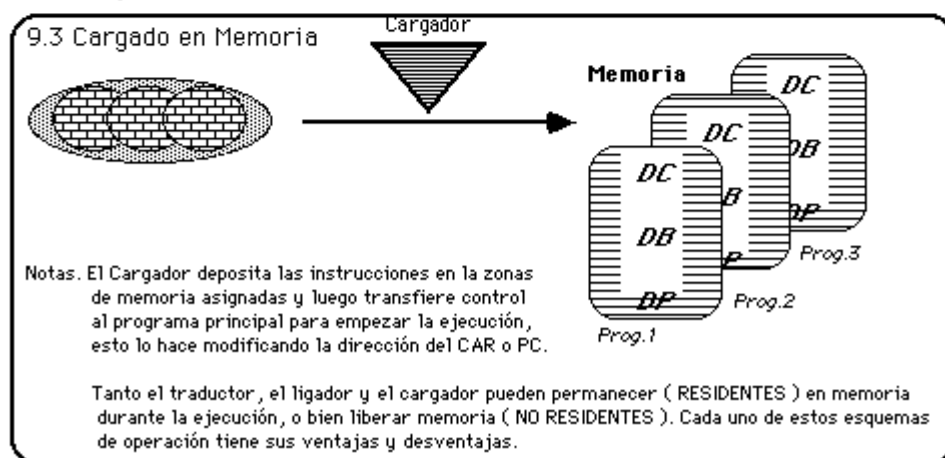


## 6) Esquema de Traducción y Ejecución

Existen un gran número de esquemas para las labores de traducción, ligado, cargado y ejecución de un programa, existen diversos esquemas de Interpretación, Compilación y Residencia en Ejecución:



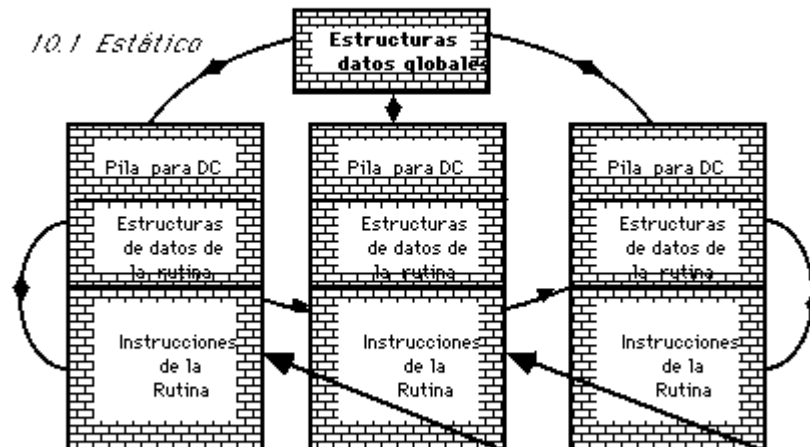
Notas. El ligador toma códigos objeto escritos inicialmente en diferentes lenguajes, pero siguiendo los protocolos estandares entre ellos para los llamados y pasaje de parámetros o mensajes. El programa Ejecutable puede incluir el código de todas las rutinas o programas ligados o bien puede simplemente mantener una lista de apuntadores a direcciones en memoria, donde se asume se encuentran ya RESIDENTES los ejecutables, como se muestra en el diagrama





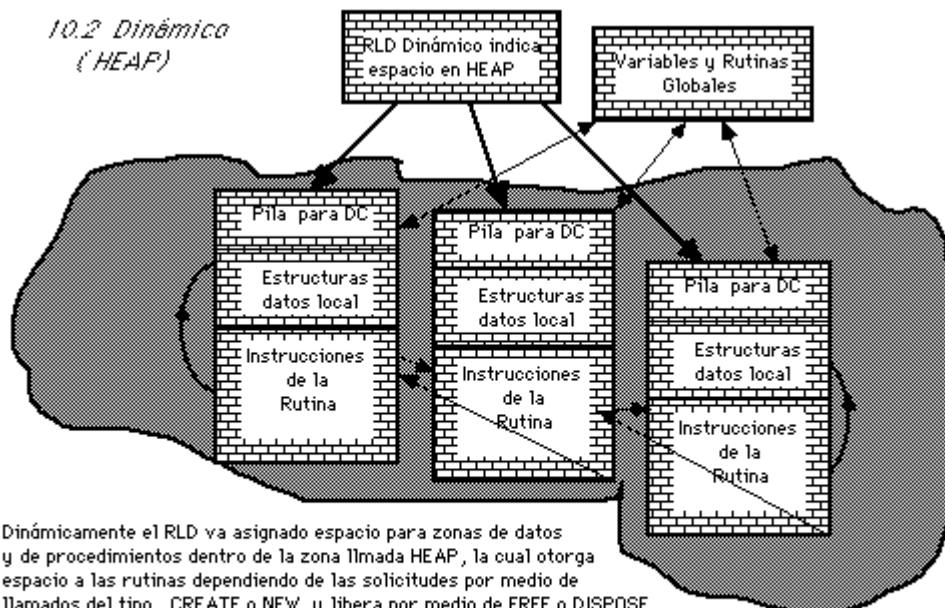
## 10. Linking and Loading

### 10.1 Estática

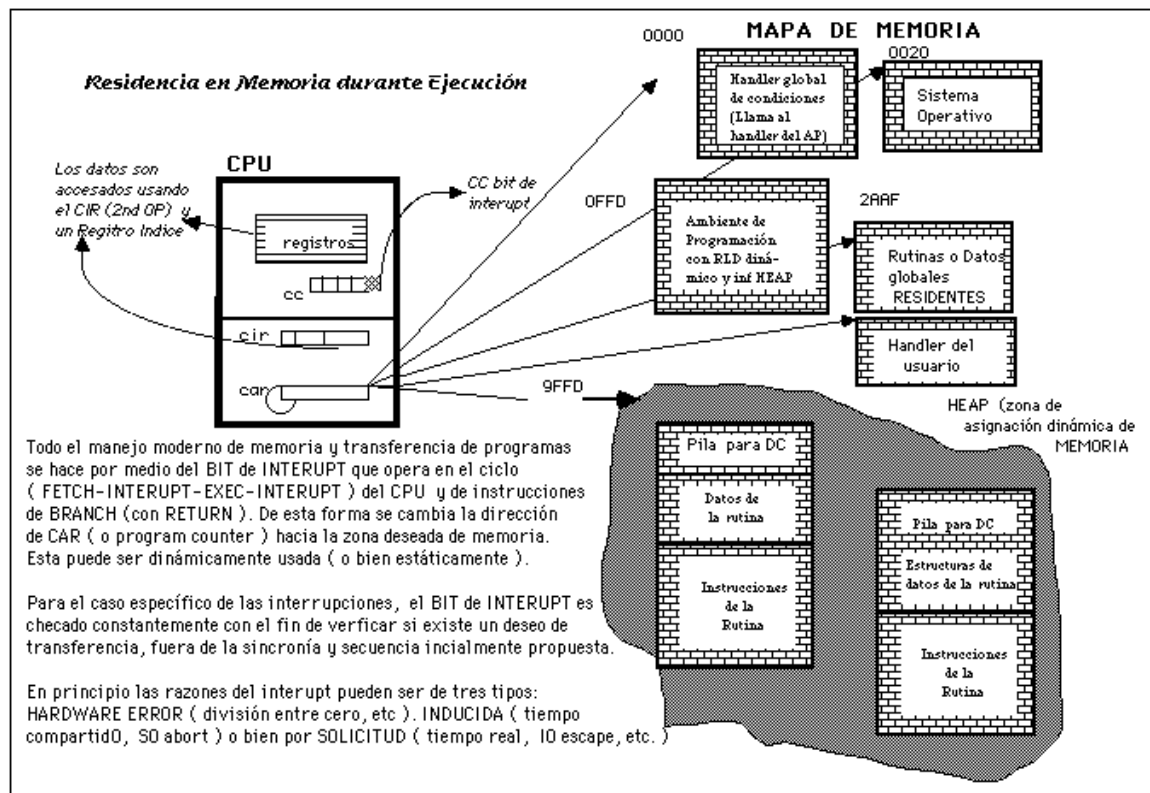


Una vez en memoria, los programas se comunican por medio de dirección estáticamente definidas en los RLD usados por el linker.

### 10.2 Dinámica (HEAP)



Dinómicamente el RLD va asignado espacio para zonas de datos y de procedimientos dentro de la zona llamada HEAP, la cual otorga espacio a las rutinas dependiendo de las solicitudes por medio de llamados del tipo CREATE o NEW y libera por medio de FREE o DISPOSE. Recuerda que la información de la dirección de memoria se accesa por medio de un registro de indexado en el ALU del CPU.



#### IV. Fronteras y Límites del Paradigma Procedural

El paradigma procedural, llamado por algunos autores programación imperativa, incluyendo en esta a la programación a nivel ensamblador, ha sido usado durante más de 40 años, en forma extensiva, sin embargo el software desarrollado no cumple todavía satisfactoriamente con los requisitos para un buen software; por lo anterior han surgido nuevas ideas de programación: estas partiendo de las fronteras mismas del paradigma, específicamente John Backus el creador del primer lenguaje procedural ( FORTRAN ) es uno de los promotores del paradigma funcional(4). Por otro lado, tocando una de las fronteras del paradigma procedural, podemos llegar al siguiente. En este artículo se utiliza el caso de la relación entre el paradigma Procedural y el paradigma Orientado a Objetos. En términos sencillos, partimos del tipo RECORD el cual es ampliado a un nuevo tipo de dato, denominado CLASE, el cual usando un esquema similar al Apuntador del HEAP genera copias dinámicas denominadas OBJETOS. Si notamos, en el caso del paradigma Procedural estas variaciones, aunque permitidas, no se encuentran en el corazón del paradigma, sino que en las fronteras del mismo (históricamente, este nuevo tipo de dato lo crearon los noruegos Dahl y Nygaard en el lenguaje SIMULA 67(5)).

Lo interesante de esto es que esta ruptura genera una concepción distinta de programación, dejando de ser jerárquica (TOP DOWN) y convirtiéndose en sino heterárquica o reticular (INDEPENDENT MODULES). Esta nueva visión plantea los siguientes principios:

El control debe estar distribuido

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

Los procedimientos deben de estar desarrollados alrededor de metas definidas  
 Los procedimientos deben conocer lo mínimo unos de otros  
 Mensajes (avisos, sugerencias, quejas) deben poder ser enviadas entre los diversos elementos o procedimientos  
 Cada elemento debe ser lo suficientemente inteligente para encontrar sus propios errores

Si consideramos las solicitudes de un buen software con las propuestas anteriores, se podría argumentar que esta nueva forma de programación puede ser más eficiente. Solo definir al participante clave de la ejecución: ¿Es el procedimiento o es el dato?. La respuesta aparentemente está en la gramática de cualquier lengua natural: ¿ el verbo o el sujeto ? Mientras que en los lenguajes procedurales la respuesta parece estar dada en el procedimiento, en este nuevo paradigma se pone énfasis en el dato.

## **VII. Programación Funcional**

Un paradigma completamente distinto a la programación procedural y de objetos es el paradigma funcional, el cual parte de una definición sencilla: *una expresión es únicamente el resultado del valor de sus subexpresiones*. Dicha propuesta invita entonces al anidamiento y recursividad de expresiones que utilizan un conjunto poderoso de funciones primitivas. Así la estructura de programación queda sujeta a la forma de evaluar cada subexpresión, partiendo de aquellas expresiones que representan constantes. El siguiente nivel son las variables y finalmente las funciones, las cuales son tratadas en forma idéntica que un dato. Debido a los mencionado anteriormente la sintaxis del lenguaje es particular, normalmente muy sencilla y constituida por una sola regla el caso más popular es el de LISP, donde la sintaxis del lenguaje tanto para datos como para instrucciones y funciones es simplemente una lista de elementos dentro de un paréntesis. Es importante que LISP, fue introducido en 1959 por John McCarthy (8), o sea que el paradigma ha existido casi tanto tiempo como el procedural. Para algunos adicionalmente al paradigma funcional debería mencionarse el paradigma lógico. En este artículo se ha considerado que la programación lógica, es en realidad un ejemplo muy claro del paradigma funcional, en donde el tipo de funciones contempladas son las funciones lógicas. Por esta razón no se describe dicho paradigma en un apartado adicional.

Cada lenguaje funcional define su conjunto básico de instrucciones, que identifican el tipo de función a ejecutar; por la anterior la aplicación del mismo. Así por ejemplo SNOBOL parte de funciones de acoplamiento y búsqueda de patrones o modelos para efectuar sus operaciones. PROLOG por su parte define reglas de producción similares a la lógica de predicados y LISP, se dedica a trabajar con listas de elementos. De cualquier modo, los postulados del paradigma son claramente observados por todos los lenguajes que caen dentro de esta categoría.

Para algunos autores la programación Lógica representa un paradigma distinto al paradigma funcional, esto es ciertamente válido en la medida de una especialización mayor de cada uno de los paradigmas. Para este caso se considera el uso de la lógica formal como la gramática de conceptualización para el modelado de los programas.

Se presentan a continuación ejemplos de programas en LISP y SNOBOL como representativos del paradigma.

LISP (1959)

Algunos ejemplos de operaciones sencillas bajo un intérprete LISP

NOTA. El texto en cursiva indica lo que despliega la terminal

```
( TIMES 9 3 )          27
```

```
( MAX 3 2 6 ) 6
```

```
( SETQ AMIGOS (MANUEL PEDRO LUCIA ) )
AMIGOS
AMIGOS (MANUEL PEDRO LUCIA )
( SETQ ENEMIGOS `(TERESA JAMES HANS ) )
ENEMIGOS
ENEMIGOS (TERESA JAMES HANS )

( DEFINE (AMIGONUEVO NOMBRE )
  (SETQ ENEMIGOS (DELETE NOMBRE ENEMIGOS))
  (SETQ AMIGOS (CONS NOMBRE AMIGOS)) )
AMIGONUEVO

(AMIGONUEVO `TERESA )
AMIGONUEVO

AMIGOS (TERESA MANUEL PEDRO LUCIA )
ENEMIGOS ( JAMES HANS )

(DEFINE (POWER M N )
  (COND (( EQUAL N 0 ) 1 )
        (( EQUAL N 1 ) M )
        (( EQUAL N 2 ) ( TIMES M M ) ) ) )
POWER

(POWER 3 0 )          1
(POWER 3 1 )          3
(POWER 3 2 )          9
```



```
(DEFINE (POWER M N )
      (COND (( EQUAL N 0 ) 1 )
            (( TIMES M ( POWER
              ( M (SUB1 N ) ) ) ) ) ) ) ) POWER
```

```
(POWER 3 3)                27
```

Toda lista tiene la siguiente forma

CAR CDR (recursivo )  
un operador ( + , minus )  
una constante ( 8, `A )  
una variable ( X , A )  
una lista ( ( ) )  
a su vez el CDR es una lista

SNOBOL ( 1962 )

\* Programa para derivar analíticamente

\* Patrones a usar

```
PatBinario = POS(0) "(" BAL . U
ANY("+-*!/") . OPTR BAL . V "
```

```
DEFINE ("D(FUNCION,X) U,V,OPTR")
:(FIN.DER)
```

\* Busca el patrón correspondiente

```
D FUNCION PatBinario : (S$("D"OPTR) )
```

\* Patrones a seleccionar

```
DX D = IDENT(FUNCION,X) 1 : (IMPRIME)
```

```
DC D = 0
```

```
:(IMPRIME)
```

```
D+
```

```
D- D = "(" D(U,X) OPTR D(V,X) )"
:(IMPRIME)
```

```
D* D = "(" V "*" D(U,X) ")+(" U "*" D(V,X) ")" : (IMPRIME)
```

```
D/ D = "(" V "*" D(U,X) "-
(" U "*" D(V,X) ")/(" V "!2 )" : (IMPRIME)
```

```
D! D = "(" V "*" (" U !"V - 1"))" D(U,X) ")" : (IMPRIME)
```

\*

```
IMPRIME TERMINAL = "La derivada es " D : (RETURN)
```





```
FIN.DER
```

```
* Programa Principal
```

```
*           Asume que todas las operaciones
```

```
*           estan entre parentesis
```

```
CICLO  TERMINAL  = "Deme la función a  
                derivar ( entre parentesis) "
```

```
        FUNCION  = TERMINAL
```

```
        FUNCION  "FIN"
```

```
:S(END)
```

```
        TERMINAL = "Deme la variable  
                a considerar "
```

```
        X = TERMINAL
```

```
        FUNCION  = "FIN"
```

```
        TERMINAL = D(FUNCION,X)                :(CICLO)
```

```
END
```

## VIII. Programación Paralela

La programación paralela representa hoy en día un verdadero reto en su instrumentación. En este caso el paradigma correspondiente tiene que ver con la capacidad de concebir y programar tareas paralelas, con el fin de que puedan ser ejecutadas en forma concurrente, esto es al mismo tiempo. Lo anterior invita al programador a buscar algoritmos concurrentes para problemas que tradicionalmente se han atacado en forma secuencial. Por ejemplo, para el caso de los métodos numéricos, se busca el diseño de algoritmos matriciales o vectoriales, en vez de cálculos basados en variables de tipo escalar. Aunque parecería novedoso no lo es, ya que siempre han existido modelos matemáticos que se expresan en forma matricial pero que hasta la fecha se había venido realizando en forma escalar.

Huelga decir que para poder instrumentar adecuadamente el paradigma, es necesario contar con un hardware del tipo SIMD (Single Instrucción/ Múltiple Data). Hoy en día, el costo de este tipo de equipos ha bajado considerablemente por lo resulta factible desarrollar aplicaciones concurrentes a precios razonables. Históricamente, en la definición de PL/I ya se incluían algunos estatutos que invitan al pensamiento concurrente. Como se mencionó la programación tiene aplicabilidad en la medida de que el programador domine el álgebra matricial y los algoritmos paralelos, a continuación se presentan algunos ejemplos de uso del álgebra matricial y su conversión a código paralelo.

La programación en paralelo del algoritmo Gauss Jordan queda como se muestra en el código siguiente:

-- Resolución de un sistema de ecuaciones simultáneas por Gauss Jordan



SEQ

```
SEQ -- Lee matriz B | U | I
en U quedará la solución en y en I la inversa
  For i :=1 to 3
    For j:= 1 to 6
      Read( b( i,j ) )
-- El elemento b ( i, i ) tiene que ser <> 0
END
PAR -- Prepara matriz de apoyo
  m(1,1) := 1/b(1,1)
  m(2,1) := - b(2,1)/b(1,1)
  m(3,1) := - b(3,1)/b(1,1)
END
PAR
-- Eliminación en 1 de 2 y 3 y Normalización de 1
  B := M * B -----> ver *abajo
END
PAR -- Prepara matriz de apoyo
  m(2,3) := - b(2,3)/b(3,3)
  m(3,2) := - b(3,2)/b(2,2)
END
PAR -- Eliminación de 2 en 3 y 3 en 2
  B := M * B -----> ver *abajo
END
PAR -- Prepara matriz de apoyo
  m(2,2) := 1/b(2,2)
  m(3,3) := 1/b(3,3)
  m(1,2) := - b(1,2)/b(2,2)
  m(1,3) := - b(1,3)/b(3,3)
END
PAR
-- Eliminación de 2 y 3 en 1 y Normalización de 2 y 3
  B := M * B -----> ver *abajo
END
SEQ -- Escribe Resultado
  For i := 1 to 3
    For j := 4 to 7
      Write( b( i,j ) )
END
```



END

\* Nota  $B := M * B$

es calculado por medio de la multiplicación de  $M * B$  se efectúa en forma paralela equivalentes a 21 procesos y la reasignación de  $B := B$  prima

PAR

$b'(1,1) := \text{SUM } m(1,k)*b(k,1) \quad k=1 \text{ to } 3$

$b'(1,2) := \text{SUM } m(1,k)*b(k,2) \quad k=1 \text{ to } 3$

.....

$b'(i,j) := \text{SUM } m(i,k)*b(k,j) \quad k=1 \text{ to } 3$

$b'(3,6) := \text{SUM } m(3,k)*b(k,6) \quad k=1 \text{ to } 3$

$b'(3,7) := \text{SUM } m(3,k)*b(k,7) \quad k=1 \text{ to } 3$

END

A su vez cada sumatoria se puede llevar a cabo en forma paralela o bien a través de un pipeline , el código completo de  $B := M * B ; B := B$  prima es entonces

SEQ

PAR

SEQ

PAR

$mp(1,1,1) := m(1,1)*b(1,1)$

$mp(1,1,2) := m(1,2)*b(2,1)$

$mp(1,1,3) := m(1,3)*b(3,1)$

END

$b'(1,1) := mp(1,1,1)+mp(1,1,2)+mp(1,1,3)$

END

. . . . .

SEQ

PAR

$mp(i,j,1) := m(i,1)*b(1,j)$

$mp(i,j,2) := m(1,2)*b(2,j)$

$mp(i,j,3) := m(1,3)*b(3,j)$

END

$b'(i,j) := mp(i,j,1)+mp(i,j,2)+mp(i,j,3)$

END





```

. . . . .
SEQ
  PAR
    mp(3,7,1) := m(3,1)*b(7,j)
    mp(3,7,2) := m(3,2)*b(7,j)
    mp(3,7,3) := m(3,3)*b(7,j)
  END
  b'(3,7) := mp(3,7,1)+mp(3,7,2)+mp(3,7,3)
END
END
PAR
  b(1,1) := b'(1,1)
  b(1,2) := b'(1,2)
  . . . . .
  b(i,j) := b'(i,j)
  . . . . .
  b(3,6) := b'(3,6)
  b(3,7) := b'(3,7)
END
END
```

## IX. Herramientas para Aplicaciones

El desarrollo de software para aplicaciones sencillas para el comercio y la oficina durante mucho tiempo estuvo limitado por el costo de contratar a un experto en programación administrativa, por esta razón la automatización no se desarrolló más que en las grandes empresas. Con la llegada de la microcomputadora en el 1977, se abrió una puerta adicional, esto es una revolución dentro de la revolución de información que ya se vivía en esos años. El costo de estos equipos invitó al desarrollo de software económico para todo tipo de usuario. Durante unos años el paradigma de procedural, e inclusive la programación en ensamblador era la única forma de programar las microcomputadoras. Por la anterior aunque el equipo de hardware resultaba económico, el usuario requería aprender a programar en BASIC, o bien contratar a un experto.

Una nueva forma de pensar era requerida, esta debería estar dirigida al desarrollo de lenguajes o paquetes de corte comercial y económico que permitieran que usuarios con poco conocimiento desarrollarán aplicaciones fácilmente; igualmente tendrían que permitir la modificación de sus programas sin mucha dificultad. A este paradigma de programación se le ha denominado en este artículo, el paradigma de las Herramientas para Aplicaciones. El ejemplo por excelencia es la hoja de cálculo, la cual surge en 1979 bajo el nombre VisiCALC. En primer término se conciben tantos paquetes como aplicaciones distintas; sin embargo,

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

todos los paquetes utilizan conceptos similares, que en si forman los postulados de este pseudo paradigma. En segundo término con el fin de dar presentar una interfaz amigable, estos paquetes requieren de la existencia de un residente poderoso y una manejo gráfico y/o de multimedia. Por lo anterior es común incluir en este paradigma muchas de las características del lenguaje orientado a objetos. Lo anterior por que se sugiere que la forma más sencilla de comunicación Hombre-Máquina es a través de objetos y no de funciones.

Las características comunes de estos lenguajes se refieren a la forma efectuar la entrada de datos y la selección de funciones, la cual se lleva de algunas de las siguientes formas: Menus, Ejemplos o Llenado de formatos predefinidos. La estructura y secuencia de un programa se define en forma aleatoria, esto es, no es necesario definir las actividades en forma secuencial. Por esta razón el usuario puede llevar a cabo una tarea de formas múltiples dando la libertad de desarrollar su estilo personal de programar. La premisa fundamental es que no se pretende desarrollar sistemas de software muy complejos, sino mas bien proporcionar al usuario una herramienta que le permita desarrollar programas rápidamente. Cabe mencionar, que hoy en día, algunas de estas herramientas se han sofisticado tanto (por ejemplo: la hoja de cálculo), que se pueden efectuar aplicaciones muy complejas en forma eficiente y rápida, transformando a la herramienta, en un verdadero lenguaje de programación.

Excel es un buen ejemplo de herramienta de aplicación como se muestra en la figura a continuación



## **PROGRAMACIÓN ORIENTADA A OBJETOS**

### **1) INTRODUCCIÓN**

Actualmente una de las áreas más candentes en la industria y en el ámbito académico es la orientación a objetos. La orientación a objetos promete mejoras de amplio alcance en la forma de diseño, desarrollo y mantenimiento del software ofreciendo una solución a largo plazo a los problemas y preocupaciones que han existido desde el comienzo en el desarrollo de software: la falta de portabilidad del código y reusabilidad, código que es difícil de modificar, ciclos de desarrollo largos y técnicas de codificación no intuitivas.

Un lenguaje orientado a objetos ataca estos problemas. Tiene tres características básicas: debe estar basado en objetos, basado en clases y capaz de tener herencia de clases. Muchos lenguajes cumplen uno o dos de estos puntos; muchos menos cumplen los tres. La barrera más difícil de sortear es usualmente la herencia.

El concepto de programación orientada a objetos (OOP) no es nuevo, lenguajes clásicos como SmallTalk se basan en ella. Dado que la OOP se basa en la idea natural de la existencia de un mundo lleno de objetos y que la resolución del problema se realiza en

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

términos de objetos, un lenguaje se dice que está basado en objetos si soporta objetos como una característica fundamental del mismo.

El elemento fundamental de la OOP es, como su nombre lo indica, el objeto. Podemos definir un objeto como un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.

Esta definición especifica varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

## **2) ESTRUCTURA DE UN OBJETO**

Un objeto puede considerarse como una especie de cápsula dividida en tres partes:

- 1 - RELACIONES
- 2 - PROPIEDADES
- 3 - MÉTODOS

Cada uno de estos componentes desempeña un papel totalmente independiente: Las relaciones permiten que el objeto se inserte en la organización y están formadas esencialmente por punteros a otros objetos.

Las propiedades distinguen un objeto determinado de los restantes que forman parte de la misma organización y tiene valores que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización.

Los métodos son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (código) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia.

## **Encapsulamiento y ocultación**

Como hemos visto, cada objeto es una estructura compleja en cuyo interior hay datos y programas, todos ellos relacionados entre sí, como si estuvieran encerrados conjuntamente en una cápsula. Esta propiedad (encapsulamiento), es una de las características fundamentales en la OOP.

Los objetos son inaccesibles, e impiden que otros objetos, los usuarios, o incluso los programadores conozcan cómo está distribuida la información o qué información hay disponible. Esta propiedad de los objetos se denomina ocultación de la información. Esto no quiere decir, sin embargo, que sea imposible conocer lo necesario respecto a un objeto y a lo que contiene. Si así fuera no se podría hacer gran cosa con él. Lo que sucede es que las peticiones de información a un objeto. Deben realizarse a través de mensajes dirigidos a él, con la orden de realizar la operación pertinente. La respuesta a estas órdenes será la

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

información requerida, siempre que el objeto considere que quien envía el mensaje está autorizado para obtenerla.

El hecho de que cada objeto sea una cápsula facilita enormemente que un objeto determinado pueda ser transportado a otro punto de la organización, o incluso a otra organización totalmente diferente que precise de él. Si el objeto ha sido bien construido, sus métodos seguirán funcionando en el nuevo entorno sin problemas. Esta cualidad hace que la OOP sea muy apta para la reutilización de programas.

## Organización de los objetos

En principio, los objetos forman siempre una organización jerárquica, en el sentido de que ciertos objetos son superiores a otros de cierto modo.

Existen varios tipos de jerarquías: serán simples cuando su estructura pueda ser representada por medio de un "árbol". En otros casos puede ser más compleja.

En cualquier caso, sea la estructura simple o compleja, podrán distinguirse en ella tres niveles de objetos.

*-La raíz de la jerarquía.* Se trata de un objeto único y especial. Este se caracteriza por estar en el nivel más alto de la estructura y suele recibir un nombre muy genérico, que indica su categoría especial, como por ejemplo objeto madre, Raíz o Entidad.



*-Los objetos intermedios.* Son aquellos que descienden directamente de la raíz y que a su vez tienen descendientes. Representan conjuntos o clases de objetos, que pueden ser muy generales o muy especializados, según la aplicación. Normalmente reciben nombres genéricos que denotan al conjunto de objetos que representan, por ejemplo, VENTANA, CUENTA, FICHERO. En un conjunto reciben el nombre de clases o tipos si descienden de otra clase o subclase.

*-Los objetos terminales.* Son todos aquellos que descienden de una clase o subclase y no tienen descendientes. Suelen llamarse casos particulares, instancias o ítems porque representan los elementos del conjunto representado por la clase o subclase a la que pertenecen.

Veamos ahora en detalle los tres elementos mencionados en "Estructura de un Objeto".

### a. Relaciones



 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p>PARADIGMAS DE PROGRAMACIÓN</p>
--	---	-----------------------------------

Las relaciones entre objetos son, precisamente, los enlaces que permiten a un objeto relacionarse con aquellos que forman parte de la misma organización.

Las hay de dos tipos fundamentales:

*-Relaciones jerárquicas.* Son esenciales para la existencia misma de la aplicación porque la construyen. Son bidireccionales, es decir, un objeto es padre de otro cuando el primer objeto se encuentra situado inmediatamente encima del segundo en la organización en la que ambos forman parte; asimismo, si un objeto es padre de otro, el segundo es hijo del primero (en la fig. 2, B es padre de D,E y F, es decir, D,E y F son hijos de B; en la fig. 3, los objetos B y C son padres de F, que a su vez es hijo de ambos).

Una organización jerárquica simple puede definirse como aquella en la que un objeto puede tener un solo padre, mientras que en una organización jerárquica compleja un hijo puede tener varios padres).

*-Relaciones semánticas.* Se refieren a las relaciones que no tienen nada que ver con la organización de la que forman parte los objetos que las establecen. Sus propiedades y consecuencia sólo dependen de los objetos en sí mismos (de su significado) y no de su posición en la organización.

Se puede ver mejor con un ejemplo: supongamos que vamos a construir un diccionario informatizado que permita al usuario obtener la definición de una palabra cualquiera. Supongamos que, en dicho diccionario, las palabras son objetos y que la organización jerárquica es la que proviene de forma natural de la estructura de nuestros conocimientos sobre el mundo.

La raíz del diccionario podría llamarse TEMAS. De éste término genérico descenderán tres grandes ramas de objetos llamadas VIDA, MUNDO y HOMBRE. El primero (vida) comprenderá las ciencias biológicas: Biología y Medicina. El segundo (mundo), las ciencias de la naturaleza inerte: las Matemáticas, la Física, la Química y la Geología. El tercero (hombre) comprenderá las ciencias humanas: la Geografía, la Historia, etc.

Veamos un ejemplo: estableceremos la relación *trabajo* entre los objetos NEWTON y OPTICA y la interpretaremos diciendo que significa que Newton *trabajó* en óptica (véase la fig. 4). La relación es, evidentemente, semántica, pues no establece ninguna connotación jerárquica entre NEWTON y OPTICA y su interpretación depende exclusivamente del significado de ambos objetos.



La existencia de esta relación nos permitirá responder a preguntas como:

¿Quién *trabajó* en óptica?

¿En qué *trabajó* Newton?

¿Quien *trabajó* en Física?

Las dos primeras se deducen inmediatamente de la existencia de la relación *trabajo*. Para la tercera observamos que si Newton *trabajó* en óptica automáticamente sabemos que

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

trabajó en Física, por ser óptica una rama de la Física (en nuestro diccionario, el objeto OPTICA es hijo del objeto FISICA). Entonces gracias a la OOP podemos responder a la tercera pregunta sin necesidad de establecer una relación entre NEWTON y FISICA, apoyándonos sólo en la relación definida entre NEWTON y OPTICA y en que OPTICA es hijo de FISICA. De este modo se elimina toda redundancia innecesaria y la cantidad de información que tendremos que definir para todo el diccionario será mínima.

## **b. Propiedades**

Todo objeto puede tener cierto número de propiedades, cada una de las cuales tendrá, a su vez, uno o varios valores. En OOP, las propiedades corresponden a las clásicas "variables" de la programación estructurada. Son, por lo tanto, datos encapsulados dentro del objeto, junto con los métodos (programas) y las relaciones (punteros a otros objetos). Las propiedades de un objeto pueden tener un valor único o pueden contener un conjunto de valores más o menos estructurados (matrices, vectores, listas, etc.). Además, los valores pueden ser de cualquier tipo (numérico, alfabético, etc.) si el sistema de programación lo permite.



Pero existe una diferencia con las "variables", y es que las propiedades se pueden heredar de unos objetos a otros. En consecuencia, un objeto puede tener una propiedad de maneras diferentes:

- Propiedades propias*. Están formadas dentro de la cápsula del objeto.
- Propiedades heredadas*. Están definidas en un objeto diferente, antepasado de éste (padre, "abuelo", etc.). A veces estas propiedades se llaman propiedades miembro porque el objeto las posee por el mero hecho de ser miembro de una clase.

## **c. Métodos**

Una operación que realiza acceso a los datos. Podemos definir método como un programa procedimental o procedural escrito en cualquier lenguaje, que está asociado a un objeto determinado y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por éste o por sus descendientes.

Son sinónimos de 'método' todos aquellos términos que se han aplicado tradicionalmente a los programas, como procedimiento, función, rutina, etc. Sin embargo, es conveniente utilizar el término 'método' para que se distingan claramente las propiedades especiales que adquiere un programa en el entorno OOP, que afectan fundamentalmente a la forma de invocarlo (únicamente a través de un mensaje) y a su campo de acción, limitado a un objeto y a sus descendientes, aunque posiblemente no a todos.

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

Si los métodos son programas, se deduce que podrían tener argumentos, o parámetros. Puesto que los métodos pueden heredarse de unos objetos a otros, un objeto puede disponer de un método de dos maneras diferentes:

-*Métodos propios*. Están incluidos dentro de la cápsula del objeto.

-*Métodos heredados*. Están definidos en un objeto diferente, antepasado de éste (padre, "abuelo", etc.). A veces estos métodos se llaman métodos miembro porque el objeto los posee por el mero hecho de ser miembro de una clase.

#### **d. Polimorfismo**

Una de las características fundamentales de la OOP es el polimorfismo, que no es otra cosa que la posibilidad de construir varios métodos con el mismo nombre, pero con relación a la clase a la que pertenece cada uno, con comportamientos diferentes. Esto conlleva la habilidad de enviar un mismo mensaje a objetos de clases diferentes. Estos objetos recibirían el mismo mensaje global pero responderían a él de formas diferentes; por ejemplo, un mensaje "+" a un objeto ENTERO significaría suma, mientras que para un objeto STRING significaría concatenación ("pegar" strings uno seguido al otro)

#### **e. Demonios**

Es un tipo especial de métodos, relativamente poco frecuente en los sistemas de OOP, que se activa automáticamente cuando sucede algo especial. Es decir, es un programa, como los métodos ordinarios, pero se diferencia de estos porque su ejecución no se activa con un mensaje, sino que se desencadena automáticamente cuando ocurre un suceso determinado: la asignación de un valor a una propiedad de un objeto, la lectura de un valor determinado, etc.

Los demonios, cuando existen, se diferencian de otros métodos por que no son heredables y porque a veces están ligados a una de las propiedades de un objeto, mas que al objeto entero.

### **3) CONSIDERACIONES FINALES**

#### **3.1) Beneficios que se obtienen del desarrollo con OOP**

 <p>FACULTAD DE <b>INGENIERIA</b> UNIVERSIDAD NACIONAL DE JUJUY</p>		<p><b>PARADIGMAS DE PROGRAMACIÓN</b></p>
--	---	--

Día a día los costos del Hardware decrecen. Así surgen nuevas áreas de aplicación cotidianamente: procesamiento de imágenes y sonido, bases de datos multimediales, automatización de oficinas, ambientes de ingeniería de software, etc. Aún en las aplicaciones *tradicionales* encontramos que definir interfases hombre-máquina "a-la-Windows" suele ser bastante conveniente.

Lamentablemente, los costos de producción de software siguen aumentando; el mantenimiento y la modificación de sistemas complejos suele ser una tarea trabajosa; cada aplicación, (aunque tenga aspectos similares a otra) suele encararse como un proyecto nuevo, etc.

Todos estos problemas aún no han sido solucionados en forma completa. Pero como los objetos son portables (teóricamente) mientras que la herencia permite la reusabilidad del código orientado a objetos, es más sencillo modificar código existente porque los objetos no interaccionan excepto a través de mensajes; en consecuencia un cambio en la codificación de un objeto no afectará la operación con otro objeto siempre que los métodos respectivos permanezcan intactos. La introducción de tecnología de objetos como una herramienta conceptual para analizar, diseñar e implementar aplicaciones permite obtener aplicaciones más modificables, fácilmente extendibles y a partir de componentes reusables. Esta reusabilidad del código disminuye el tiempo que se utiliza en el desarrollo y hace que el desarrollo del software sea más intuitivo porque la gente piensa naturalmente en términos de objetos más que en términos de algoritmos de software.