Entorno Lope
Sistema de Visualización

Juan José Amor Iglesias

jjamor@ls.fi.upm.es

16 de octubre de 1997

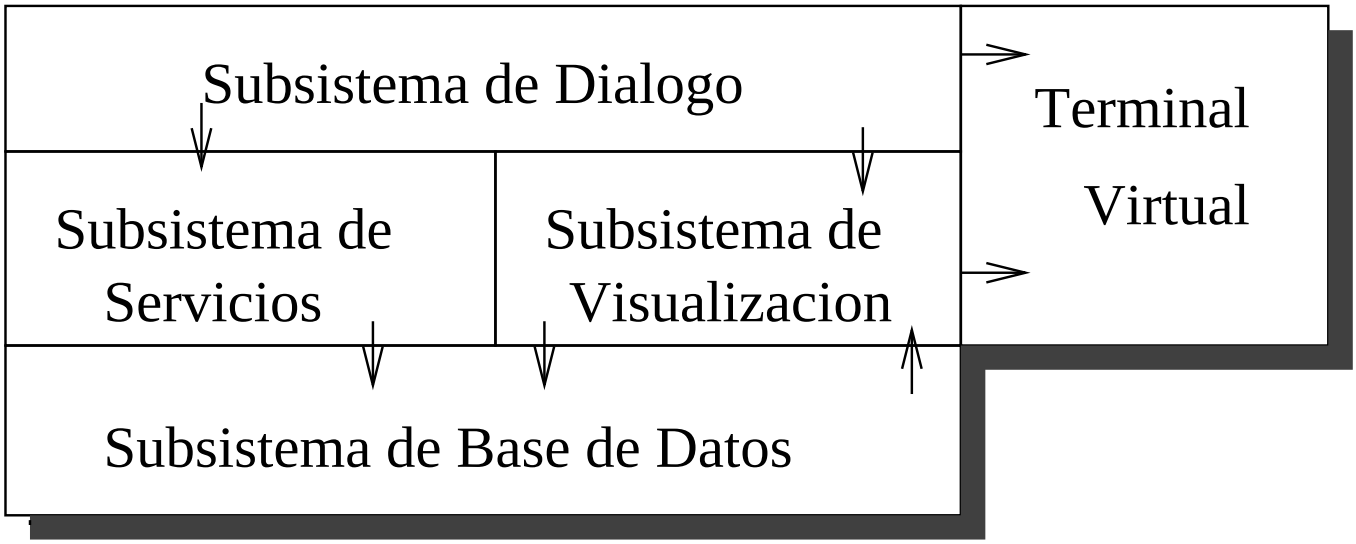
Contenidos

- Entorno LOPE: Contexto
- LOPE: Breve descripción
- Sistema de Visualización: Conceptos
- Los Formatos
- Ejemplos de Formatos
- Notas sobre Implementación

Entorno LOPE: Contexto

- Entorno de Desarrollo de Software
- Orientado a la Estructura
- Orientado a Metodología. Pero flexible
- NO ES: Entorno de Programación, ni Orientado a ningún Lenguaje
- ES: Metaentorno. Integrado.

Entorno LOPE



LOPE: Terminal Virtual

- Simplificación del Sistema de Ventanas
- Servicios: Crear/destruir "displays", ventanas, elementos de ventana (botones, paneles de texto, paneles gráficos, barras de "scroll" . . .)
- Aquí utilizamos solo los paneles de texto

LOPE: Base de Datos

- Contiene: objetos de diversos tipos *y* definición de nuevos tipos

- Tipos Simples: **Enteros, Textos, Símbolos, Identificadores**

- Tipos Definidos (estructuras):
  - **Tupla** (+)
  - **Alternativa** (|)
  - **Iteración** (*)
  - **Sinónimo**
  - **Directorio** (%)

- Los tipos definidos permiten construir nuevos tipos o esquemas de datos

Ejemplo: Esquema Documento

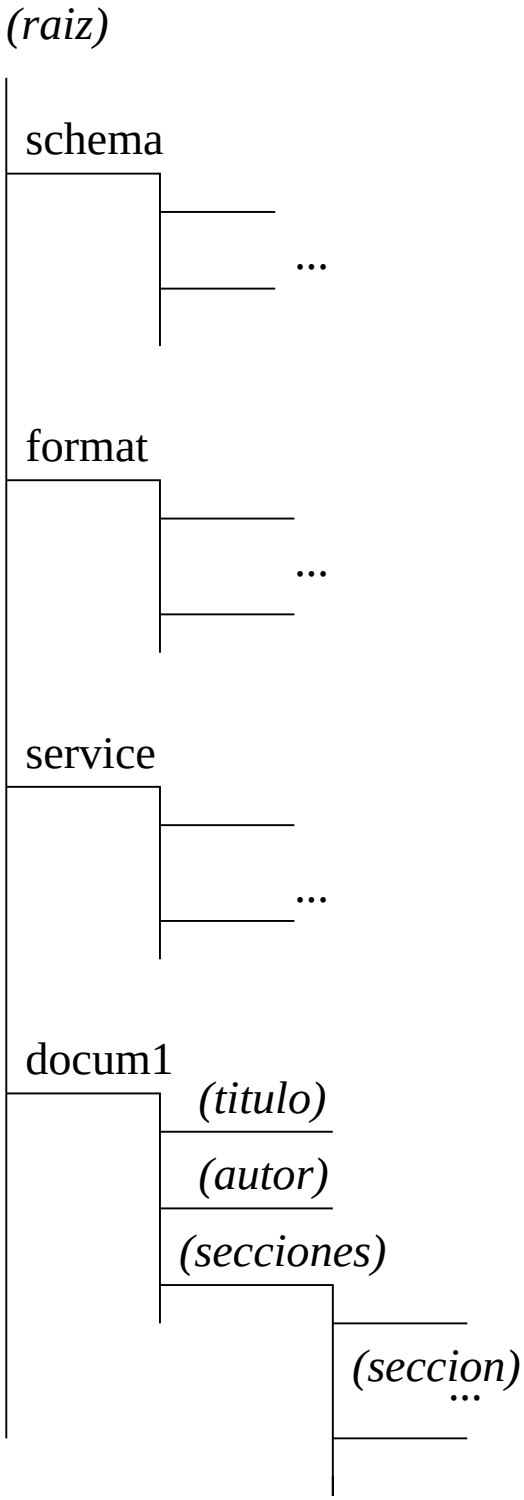
```
documento ::= + titulo + autor + secciones ;

titulo ::= .text. ;

autor ::= .text. ;

secciones ::= * seccion ;

seccion ::= + titulo + cuerpo ;

cuerpo ::= | secciones | parrafos ;

parrafos ::= * parrafo ;

parrafo ::= * linea ;

linea ::= .text. ;
```

LOPE: Base de Datos (II)

- **Jerárquica**: Es un directorio *raíz* con elementos de diversos tipos en él

- Entre ellos, algunos directorios predefinidos
  - **schema**
  - **service**
  - **format**

LOPE: Base de Datos (III)

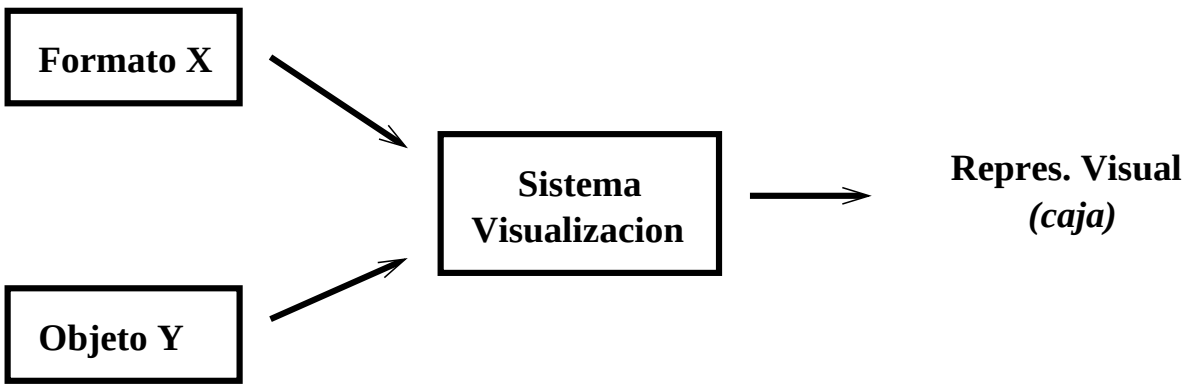


LOPE: Sistema de Visualización

- Sistema capaz de representar cualquier objeto como se desee

- Basado en los "*formatos*": objetos que especifican cómo representar otros objetos

- Es un **intérprete de formatos**:



Cajas

- Caja: Rectángulo con contenido
- Contenido de caja:
  - Vacío
  - Texto
  - 1..N cajas hijas
- Representación interna de la visualización de un objeto
- Atributos destacables: tamaño (ancho+alto), posición relativa

Formatos

- Especifica cómo representar un objeto

- Se construye combinando formatos simples:

  – conversión
  – alternativa
  – composición
  – nombrado
  – alineación tupla
  – alineación lista

  Esquema:

```
formato ::= | f_conversion | f_alternativa
            | f_composicion | f_alintupla
            | f_alinlista ;
```

- Además: formato por defecto, formato universal

Formato Conversión

- Genera líneas de texto utilizando: Textos fijos o Atributos del objeto representado

- Sintaxis: Similar a `printf` en C

- Atributos:

  `%v` : Valor
  `%n` : Nombre (etiqueta)
  `%s` : Esquema (tipo)
  `%t` : Tipo de valor (ej: .integer.)
  `%*` : Posición (índice) en objeto compuesto
  `%#` : Número de componentes
  `%%` : Carácter "%"

  Ejemplo: "`hoy es %v`"

- Valores por defecto

- Esquema: `f_conversion ::= .text.`

Formato Alternativa

- Elige, a partir de un "*discriminante*" (atributo), un nuevo formato

- Cada formato es un elemento de directorio

- Comparaciones:
  Para valor: texto o número como texto
  Para esquema, etiqueta. . . : símbolo
  Para cantidades (índice, cardinal): número como texto

- Opcional: formato por defecto. Si no existe genera caja vacía.

- Esquema:

```
f_alternativa ::= + atributo + % + formato;
atributo ::= $ ;
```

Formato Composición

- Elige nuevo objeto mediante "*selectores*"

- Un selector puede ser:
  Número: Selección de componente i-ésima
  Nombre: Selección de tipo o etiqueta
  Símbolos especiales: "<<", "$\wedge\wedge$"

- Esquema:

```
f_composicion ::= + selectores + formato ;
selectores ::= * selector ;
selector ::= $ ;
```

Formato Nombrado

- Acceso a nuevo formato a partir de otro

- Esquema: `f_nombrado ::= $ ;`

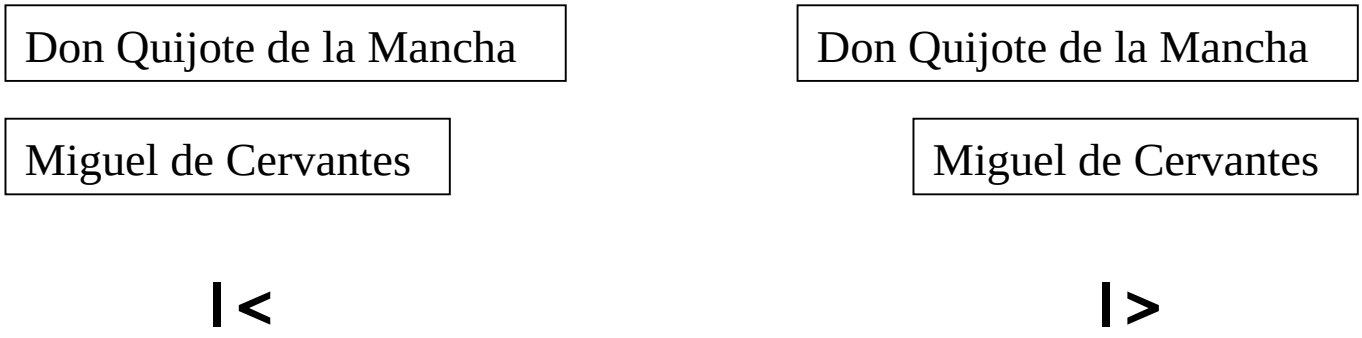
Formato Alineación Tupla

- Genera caja compuesta de N hijas
- Las N hijas se colocan entre sí según "códigos de alineación"
- Códigos de alineación:
  - Alineación vertical centrada ($||$), izquierda ($|<$) o derecha ($|>$)
  - Alineación horizontal centrada ($--$), superior ($-\wedge$) o inferior ($-\vee$)
- Posición: primera caja hija siempre en (0,0)
- Esquema:

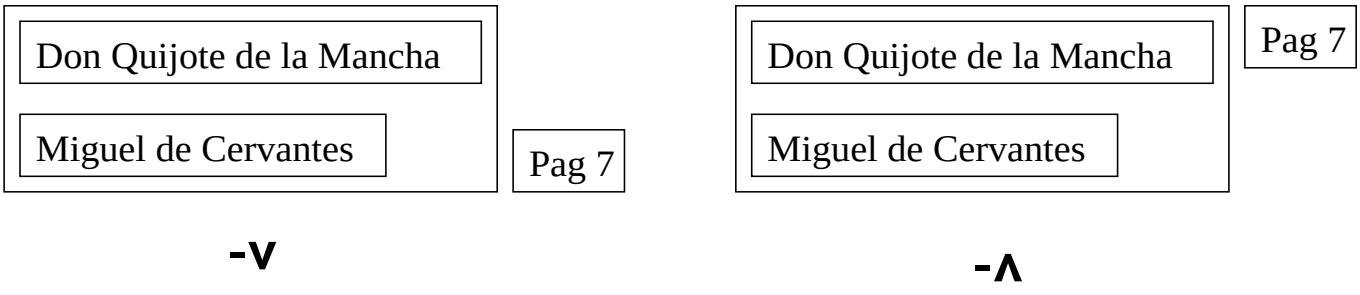
```
f_alintupla ::= + alineaciones + formatos ;
alineaciones ::= * alineacion ;
formatos ::= * formato ;
alineacion ::= $
```

Formas de Alineación

- Alineación vertical



- Alineación Horizontal



Formato Alineación Lista

- Genera caja compuesta de N hijas
- Se aplica un nuevo formato a cada componente del objeto
- Opcionalmente hay caja "*separadora*"
- Hay dos códigos de alineación:
  - Entre caja hija y separador
  - Entre separador y siguiente caja hija
- Esquema:

```
f_alinlista ::= + formato + alineacion
                + separador + alineacion ;
alineacion ::= $ ;
separador ::= .text. ;
```

Formatos por defecto y universal

- **Formato por defecto**: Definido por cada tipo de objeto, se aplica cuando no se especifica otro

- **Formato universal**: Definido con nombre "universal", se aplica si en el caso anterior, no existe formato por defecto

- El formato universal representa los objetos de forma genérica:

```
quijote (documento) [+
  (titulo) 'Don Quijote de la Mancha'
  (autor) 'Miguel de Cervantes'
  (secciones) [*
     (seccion) [+
        (titulo) 'Prologo'
     ...
     ]
  ...
  ]
]
```

Ejemplos de Formatos: Bloques de Pascal

- Representación interna:

  - Esquema:

```
bloque ::= * sentencia ;
sentencia ::= .text. ;
```

  - Un ejemplo:

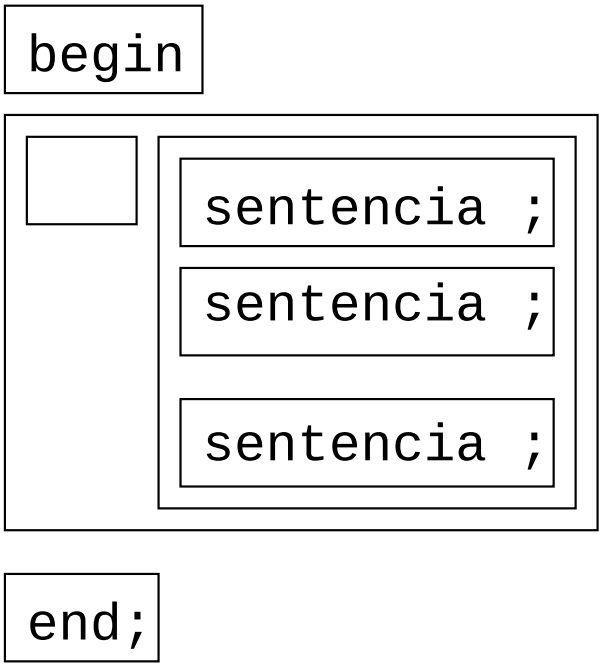
```
bloque1 (bloque) [*
  (sentencia) 'sentencia1'
  (sentencia) 'sentencia2'
  (sentencia) 'sentencia3'
]
```

Bloques de Pascal (II)

- Cómo queremos verlo:

```
begin
  sentencia1;
  sentencia2;
  sentencia3;
end;
```

- Solución:



Bloques de Pascal (III)

- El formato (notación simplificada):

```
FORMAT f_bloque =
  ALIGN
    'begin' |<
    f_bsentencias |<
    'end ;'
END;

FORMAT f_bsentencias =
  ALIGN
    '  ' _^
    LIST ALIGN
      '%v ;'
      |<
    END;
END;
```

Bloques de Pascal (IV)

- El formato (código interno):

```
f_bloque (formato) [|
  (f_alintupla) [+
    (alineaciones) [*
      (alineacion) |<
      (alineacion) |<
    ]
    (formatos) [*
       (formato) [|
          (conversion) 'begin'
       ]
       (formato) [|
          (f_nombrado) f_bsentencias
       ]
       (formato) [|
          (conversion) 'end ;'
       ]
    ]
  ]
]
```

Documentos

- Queremos este aspecto para los documentos:

```
LOPE: Visualizacion
Juan Jose Amor

1. Entornos de Desarrollo
  1.1. Conceptos Basicos
    ...
  1.2. Clasificacion de los Entornos
       1.2.1. Por ciclo de vida
         ...
2. El Entorno Lope
  2.1. Introduccion
    ...

  ...
```

Documentos (II)

```
memoria (documento) [+
 (titulo) 'LOPE: Visualizacion'
 (autor) 'Juan Jose Amor'
 (secciones) [*
   (seccion) [+
     (titulo) 'Entornos de Desarrollo'
       (cuerpo) [|
         (parrafos) [*
           ...
         ]
       ]
   ]
   (seccion) [+
     (titulo) 'Clasificacion de los entornos'
     (cuerpo) [|
       (secciones) [*
         (seccion) [+
           (titulo) 'Por ciclo de vida'
           (cuerpo) [|
            ...
```

Documentos (III)

- Hace falta indentación

- Hace falta numerar secciones

- Formato para las secciones

```
FORMAT f_secciones =
  ALIGN
    f_titulosec |<
    f_cuerposec
  END;
```

- Formato para cuerpo de sección

```
FORMAT f_cuerposec =
  CASE schema OF
    secciones: f_secciones
    parrafos: f_parrafos
END;
```

Documentos (IV)

- Formato para título

```
FORMAT f_titulosec =
  ALIGN
    f_numerosec -- ' ' -- f_titulo
  END;
```

- Formato para número de sección

```
FORMAT f_numerosec =
  ALIGN
    <<.<<: CASE schema OF
                    cuerpo: <<: f_numerosec
           ELSE
                ' '
           END; --
    '.%*'
  END;
```

Implementación (I)

- Realizada en cuatro módulos

- Servicios principales módulo `VisView`

```
VisView.Create -> Crea cajas
                  Visualiza en panel texto
                  Otras

VisView.Delete -> Destruye cajas
                  Borra panel texto
```

- Servicios principales módulo `VisBox`

```
VisBox.Build   -> Crea jerarquía cajas
                  Alineación

VisBox.Destroy -> Destruye jerarquía
                   eliminando referencias
```

Implementación (II)

- El usuario crea y destruye vistas

- El sistema mantiene referencias internas que permiten:

  - Mantener coherencia entre objeto y representación, cambiando esta última si cambia el objeto.

- La reconstrucción de vistas ante modificaciones minimiza el trabajo, a dos niveles:

  - Reconstrucción óptima de árbol de cajas

  - Optimización de número operaciones E/S

Bibliografía

[1] A. Fugetta, *A Classification of CASE Technology*. IEEE Computer, Vol 26, NO. 12, Diciembre, 1993.

[2] S.A.Dart et al, *Software Development Environments*. IEEE Computer, Vol. 20, No. 11, Noviembre 1987.

[3] M. Collado, *Implementación del Entorno* Lope. Documento Interno F.I.M. (1993)

[4] M. Collado, *Nuevo Terminal Virtual para el Prototipo 2 de* Lope. Documento Interno F.I.M. (1996)

[5] M. Collado y D. Villén: *El Lenguaje* Lope. Documento Interno F.I.M. (1995)