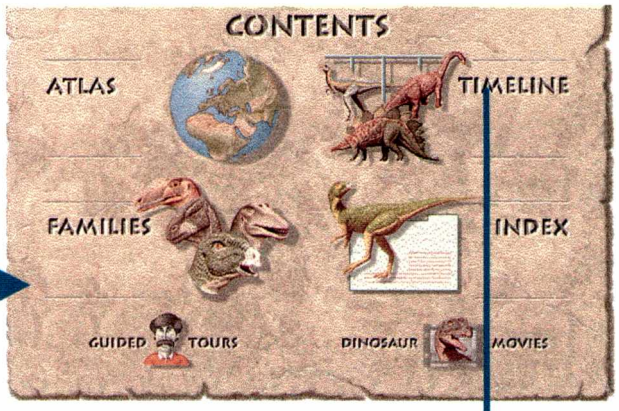
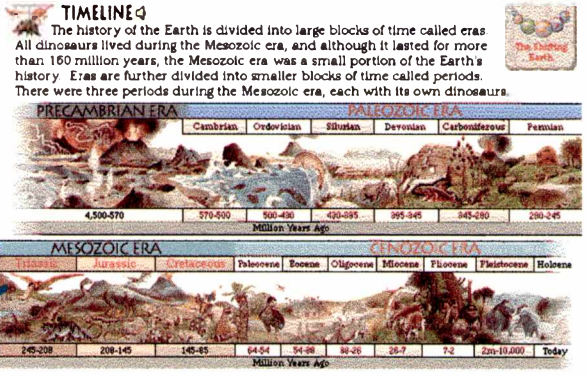
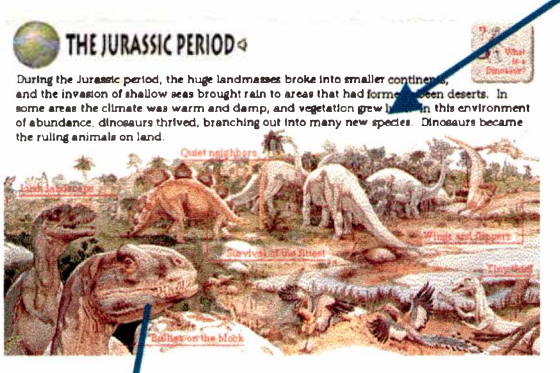


Soporte y Consultas

a

Hipertextos





Verónica Sequeiros

Inés Suárez Cabretón

Directores: Silvia Gordillo - Alicia Díaz

INDICE

CAPITULO IV

Presentación del Proyecto

INTRODUCCIÓN

Universidades, gobiernos, empresas y usuarios de todas las edades utilizan con mayor frecuencia aplicaciones hipermedia. Esto se debe, a que las mismas permiten acceder a una amplia variedad de información proveniente de distintos medios, a través de una interfaz "amigable".
Este tipo de aplicaciones presentan a los lectores diferentes opciones para recorrer la información y cada uno de ellos determinará cual elegir, en el momento de la lectura del texto.

Nuestro objetivo, es facilitar la recuperación de información en aplicaciones hipermedia donde su recorrido sería una tarea muy tediosa debido al volumen de la misma. Para esto desarrollamos un modelo para el diseño de aplicaciones hipermedia y un lenguaje de consultas que combina los beneficios del paradigma orientado a objetos con los de hipermedia.

Hemos organizado nuestro trabajo en cuatro capítulos y dos apéndices.

En el capítulo I, se describen los conceptos del paradigma orientados a objetos y algunos lenguajes de consultas existentes.

Durante el desarrollo del capítulo II, se explican los conceptos básicos de hipertextos y el modo de acceder a la información.

En el capítulo III, presentamos los beneficios y problemas de hipertextos, y veremos como algunos autores intentan mejorar la navegación con otras formas de acceso.

En el capítulo IV, se describen los conceptos de nuestro proyecto, en el cual se detallan los elementos necesarios para la representación de una aplicación hipermedia, su presentación y el modo de acceder a la información.

Por último, en el apéndice A se presenta un ejemplo utilizado durante el capítulo I para la aclaración de los conceptos descriptos y en el Apéndice B se desarrolla una aplicación hipermedia existente (Microsoft Dinosaurs) utilizando como soporte el modelo que hemos desarrollado.

CAPITULO I

Bases de Datos Orientadas a Objetos

1- BASES DE DATOS ORIENTADAS A OBJETOS

1.1- Introducción:

Los modelos de Bases de Datos tradicionales, Jerárquico ,en Red y fundamentalmente el Relacional, son ampliamente usados, y con gran éxito, en respuesta a las necesidades de las aplicaciones, especialmente de tipo comercial. En estos modelos encontramos limitaciones que dificultan su utilización en áreas específicas que adquirieron gran impulso en estos últimos años, como por ejemplo, procesamiento de imágenes, multimedia, hipertexto, etc.. Estas aplicaciones tienen la particularidad de manejar gran cantidad de datos y tratar con tipos de datos complejos.

Entre los inconvenientes presentados en los modelos tradicionales podríamos destacar: la imposibilidad de representar objetos compuestos, ya que los modelos tradicionales están orientados a registros, donde el contenido de un ítem es un dato atómico. Otro punto conflictivo es que en estos modelos sólo se utilizan tipos de datos primitivos, no permitiendo tipos definidos por el usuario. También sería deseable poder aumentar la semántica de la Base de Datos, mediante la posibilidad de incorporar el comportamiento de los datos dentro de la base, en vez de reflejarlo en las aplicaciones como ocurre en los modelos tradicionales. Observamos que en los modelos convencionales existen dos (2) formas de acceder a los datos: a través de un lenguaje Ad-Hoc, o a través de un lenguaje embebido en un lenguaje Host. En el primer caso muy efectivos para el acceso a la información, pero son computacionalmente incompletos. En el segundo caso hay distintos tipos de incompatibilidades entre el lenguaje de Base de Datos y el Lenguaje de programación: *Incompatibilidad estructural*, porque las B.D. poseen sus propios tipos de datos y los lenguajes de programación también. *Incompatibilidad operacional* causada porque los operadores relacionales son orientados a conjuntos y los operadores de los lenguajes de programación son orientados a registros.

Para poder desarrollar el modelo de nuestro proyecto en los capítulos posteriores, presentamos en primer lugar un resumen con las características principales de los Sistemas Manejadores de Bases de Datos Orientadas a Objetos, especialmente sobre los lenguajes de consulta a B.D.O.O..

1.2 - (S.M.B.D.O.O). Sistemas Manejadores de Bases de Datos Orientadas a Objetos:

Los S.M.B.D.O.O. cuentan con facilidades para un excelente manejo de los mecanismos de abstracción (clasificación, generalización/especialización y agregación) y características para describir datos y relaciones entre ellos (por ejemplo: propiedades, relaciones y objetos complejos).

Por otro lado, los S.M.B.D.O.O resuelven los problemas de incompatibilidad descriptos anteriormente, entre el Lenguaje de Programación y el de Administración de la Base de Datos, porque ambos poseen el mismo paradigma orientado a objetos.

No es nuestra intención hacer un análisis exhaustivo de B.D.O.O., ya que no es el tema específico, sino mencionar las características necesarias para describir el proyecto realizado.

Algunas de las características más significativas son:

- ✓ Objetos e Identificadores de Objetos:

  El principio fundamental del modelo es que cualquier entidad del mundo real, concreta o abstracta, se representa con un único **objeto.** Un Objeto tiene un estado y un comportamiento asociado.
  Cada objeto es unívocamente identificado por un identificador, en la mayoría de los S.M.B.D.O.O., se denomina **OID**, y no puede ser modificado en la aplicación Esto nos sugiere una independencia entre la forma de modelar un objeto y la forma de accederlo.
  Se dice que dos objetos son **iguales** si tienen el mismo estado (misma estructura y mismos valores de atributos). Se dice que dos objetos son **idénticos** si tienen el mismo OID.

- ✓ Atributos y Métodos :

  El estado de un objeto se define por los valores de sus propiedades, llamadas **atributos**. El valor de un atributo de un objeto es también un objeto. Por lo tanto, los objetos pueden definirse recursivamente en términos de otros objetos. Un atributo de un objeto puede tomar un sólo valor o un conjunto de valores.
  Los métodos definen el comportamiento de los objetos. Un objeto puede tener uno o más métodos que operan sobre su estado, es decir con los valores de sus atributos.

- ✓ Clases:

  Los objetos con las mismas características son agrupados en clases. Un objeto pertenece a una única clase como una instancia de esa clase y recibe de ella su funcionalidad.
  Una clase contiene un "molde" de las características que contienen los objetos de la misma, así como los mensajes que pueden recibir y la manera que responden a ellos.
  Las clases son consideradas como objetos, por lo tanto pueden tener un conjunto de atributos y métodos asociados a ellas. Estos atributos y métodos son llamados *atributos* y

métodos de clase. Son utilizados para capturar propiedades comunes a las instancias que pertenecen a la clase. Por ejemplo: teniendo en cuenta la definición de la clase de Automóvil; del Apéndice A, podríamos definir el siguiente atributo de clase: *PesoPromedio* de todos las instancias Automóviles. Estos atributos son 'lógicamente' parte de cada una de las instancias de la clase, pero sus valores se mantienen en el objeto clase.

✓ Jerarquía de clases y herencia :

Como lo define [KIM95] las clases en un sistema forman una jerarquía o un grafo acíclico dirigido con una raíz (DAG), llamado **jerarquía de clases,** tal que, para una clase C y un conjunto de clases {Si} de menor nivel, conectadas a C, una clase en el conjunto {Si} es una especialización de la clase C. Podemos decir también, que la clase C es una generalización de las clases en el conjunto {Si}. Las clases en {Si} son subclases de la clase C; y la clase C es una superclase de las clases en {Si}.
La jerarquía de clases captura la relación **Es-Un** entre una clase y sus superclases.
Cualquier clase en {Si} hereda todos los atributos y métodos de la clase C, y puede tener atributos y métodos adicionales. Todos los métodos y atributos definidos para una clase C son heredados en todas sus subclases recursivamente.
La jerarquía de clases (DAG) tiene una única raíz y es una clase definida por el sistema. Cada clase sobre la jerarquía tiene un nombre distinto y cada atributo y método de una clase, definido o heredado, tienen nombres distintos. Aunque la mayoría de los lenguajes orientados a objetos soporten una única raíz en la jerarquía, existen algunos como C++ que no lo hacen.
En sistemas orientados a objetos una clase puede tener cualquier número de subclases. Sin embargo, algunos sistemas permiten tener una sola superclase, mientras que en otros pueden definirse varias superclases (como ORION) .
Si una clase hereda atributos y métodos de una sola superclase; se denomina **Herencia Simple.** Si, en cambio, una clase hereda atributos y métodos de más de una superclase lo llamamos **Herencia Múltiple**.
El concepto de Herencia Múltiple presenta algunos problemas, por ejemplo, la herencia repetitiva y el conflicto de nombres. En el caso de conflicto de nombres entre atributos o métodos se produce cuando dos superclases de una clase C tienen atributos o métodos con igual nombre, nos encontramos al momento de la herencia sin saber cual de los dos atributos hereda la clase C. Por ejemplo si vemos la clase ayudante-alumno del Apéndice A, la cual hereda FechaDeIngreso de la clase Empleado y de la clase Alumno. Algunas soluciones a este conflicto podrían ser:

- Una solución muy común es seleccionar una superclase de la cual heredar los atributos y métodos en conflicto, sobre la base de algún orden de precedencia. El orden de precedencia se podría especificar en cada clase o determinarlo el usuario en tiempo de ejecución.
- Otra solución sería renombrar al menos uno de los atributos en conflicto, para heredar ambos.

Otro problema surge al heredar atributos con el mismo nombre (como en el caso de arriba) pero el dominio de uno se esos atributos es una especificación del otro. Una solución a esto, sería heredar el atributo con el dominio más específico.

Una clase hereda el comportamiento (métodos) desde sus superclases. Los métodos heredados pueden ser modificados aún si no cambiamos su nombre. Cuando un mensaje es enviado a un objeto, primero lo busca en la clase del objeto. Si el método no esta asociado a esa clase, se buscan en las superclases, una vez que el método es encontrado, el programa asociado a él es ejecutado. El uso del mismo nombre para más de un programa es conocido como **overloading**.

✓ Objetos Compuestos :

En sistemas orientados a objetos, un objeto tiene un conjunto de atributos, y el valor de un atributo es otro objeto, el cual es una instancia del dominio del atributo.
Si el dominio del atributo es una clase primitiva (entero, carácter, etc.), el valor almacenado para el atributo es una instancia simple o un conjunto de instancias simples, si tuviéramos constructores, tales como conjuntos o listas .
Si el dominio de un atributo es una clase no-primitiva, el valor almacenado para el atributo es un identificador de objeto de las instancias del dominio, o un conjunto de identificadores (en el caso de contar con constructores) . En este caso, decimos que el objeto referencia a otros objetos. Podemos distinguir dos (2) tipos de referencias:
1- Referencias sin una semántica específica, a los que llamaremos objetos complejos.
2- Referencias que hacen explícita la relación parte-de, entre un par de objetos, los cuales llamaremos objetos compuestos. La semántica de una referencia compuesta puede ser más refinada dependiendo si un objeto es una parte de un único objeto o de más de un objeto. Esto conduce a dos tipos de referencias compuestas: exclusivas y compartidas. Una referencia exclusiva desde un objeto X a otro Y, significa que Y es parte únicamente de X; mientras que una referencia compuesta compartida desde X a Y significa que Y es parte de X y posiblemente de otros objetos.
Podríamos refinar la semántica anterior (exclusiva y compartida), sobre la base de si la existencia de un objeto depende de la existencia de su objeto padre, esto es, que las referencias podrán ser dependientes o independientes.

1.3.- Lenguajes de Consulta a B.D.O.O.

El **Modelo Orientado a Objetos** es más rico que el relacional, ya que provee mecanismos de abstracción (encapsulamiento, identidad y métodos) para la definición de datos e incorpora algunas características (datos derivados, relaciones, propiedades multivaluadas, navegación interna , jerarquía de clases) de otros modelos.

Una de las diferencias entre los lenguajes de consultas relacionales y los lenguajes de consultas orientados a objetos yace en la expresividad de los operandos de cada uno.
Un Lenguaje de Consultas Orientado a Objetos aporta operadores relacionales para colecciones, recursión lineal, el grado de enriquecimiento que proveen los métodos de cada clase para obtener información derivada y operadores para consultar el esquema .

El modelo relacional no soporta la noción de identidad, la forma de identificar una entidad es a través de la clave primaria. Las propiedades multivaluadas, en el modelo relacional se recuperan utilizando relaciones adicionales, con numerosos joins. En Orientación a Objetos, estos recorridos se hacen naturalmente a través del grafo de agregación implícito en el esquema.

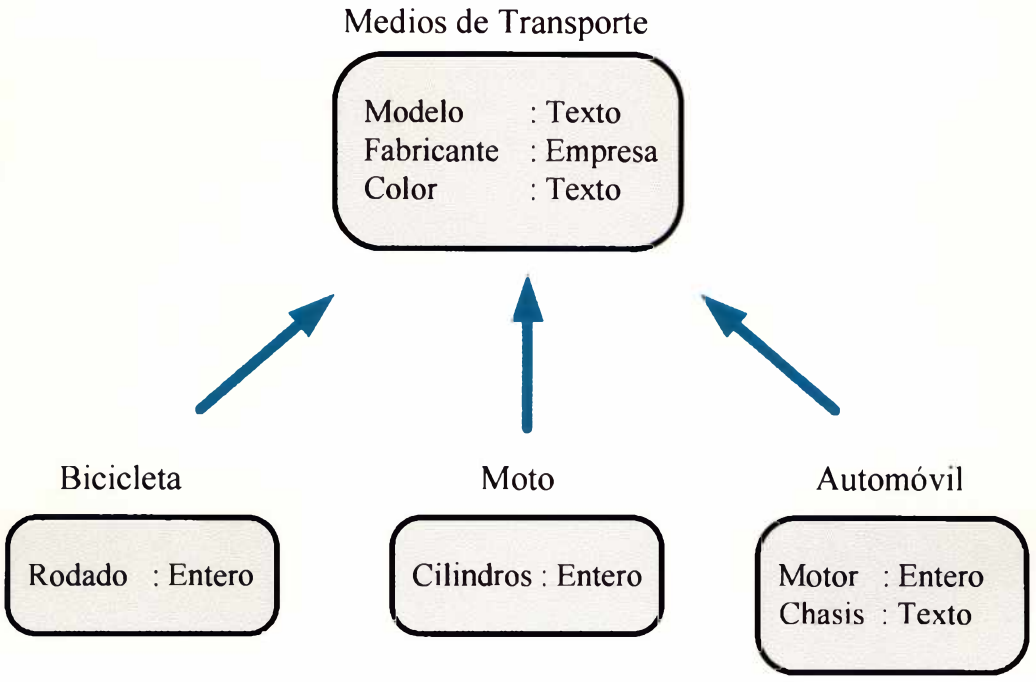
El modo de representar la información en el Modelo Orientado a Objetos es más natural , esto implica que sea muy diferente la forma de consultarla y accederla.

Veamos el siguiente ejemplo:
Supongamos que una Base de Datos incluye información sobre los distintos medios de transporte (Automóviles, Colectivos, Bicicletas, etc.)

• En una B.D. Relacional, podríamos tener un **atributo** MediosDeTransporte teniendo los distintos medios como sus posibles valores.

El registro estará compuesto por los siguientes campos:
Medios de transporte :string(15).Cuyos posibles valores son: Automóvil, Moto, Bicicleta, Colectivo, Barco y Tren.
Modelo :string(20)
Color : string(15)
Fabricante :string(20)

• En una B.D.O.O., tendría una **clase** MediosDeTransporte y los distintos medios serían sus **subclases**



Esta es una diferencia fundamental, porque la información sobre los Medios de Transporte cambia de un modelo a otro, en el primero es parte de los datos, en el segundo forma parte del esquema.

Si queremos consultar todos los tipos de Medios de Transporte :

- En el modelo relacional, proyectamos sobre el atributo MediosDeTransporte.
- En el modelo Orientado a Objetos, tenemos que interrogar al esquema y no a los datos.

Un **esquema** orientado a objetos posee información estructural y semántica sobre los objetos y su comportamiento. Está formado por la Jerarquía de Clases y por el Grafo de Agregación que ellas componen, vemos entonces que contiene mayor información que un esquema relacional, en el cual sólo definimos la estructura de los datos; la parte semántica la proveen los programas de aplicación que corran sobre esos datos.

El esquema 1 (del Apéndice A) muestra la definición de un esquema orientado a objetos.

1.3.1 - Ejemplos de Lenguajes de Consulta Orientados a Objetos:

A continuación se describen las características más importantes sobre investigaciones, prototipos y proyectos de Lenguajes de Consulta Orientado a Objetos , que nos serán de utilidad para evaluar las ventajas y desventajas de este tipo de consultas.

1.3.1.1- XSQL [KIF92]

El esquema de una B.D.O.O. es un aspecto muy importante a tener en cuenta en un lenguaje de consultas y se necesita, por lo tanto, tratar fácilmente con estructuras anidadas. XSQL provee ésta facilidad basado en " expresiones de camino ".

- Características del modelo utilizado:

El modelo de datos en el que se basa el XSQL tiene las siguientes características:

- ✓ Los **objetos** son entidades del mundo real y se hace referencia a ellos por medio de OID's (Identificadores de Objetos).
- ✓ Los **atributos** pueden ser escalares (un solo id. de objeto) o multivaluado (un conjunto de id. de objetos).
- ✓ Las **clases** organizan a los objetos en conjuntos de entidades relacionadas, formando una jerarquía de clases. Soporta **herencia múltiple**, por lo tanto una clase puede tener varias superclases. El usuario resuelve el conflicto de herencia explícitamente (explicado en Jerarquía de clases).
- ✓ Los **métodos** tienen un nombre y una función (la implementación del método). Pueden ser escalares o multivaluados (dependiendo del tipo de resultado que retornen).
- ✓ Los **tipos** son usados para clasificar a los objetos por su estructura, y los objetos son agrupados en clases basado en un criterio semántico.
- ✓ Soporta **herencia de comportamiento** (Las subclases y sus instancias heredan todos los métodos y atributos definidos en sus superclases) y **estructural**

- Características de las Expresiones de Camino:

La expresión de camino es una forma de acceder a los datos a través de estructuras anidadas (Grafo de Agregación). Pueden tener "selectores" que seleccionen datos o parte del esquema del cual los datos serán recuperados. El objetivo de estos selectores es limitar el espacio de búsqueda.

- Estructura y Manejo de las "expresión de camino"

Sel0 . ExprAtri1{[Sel1]} . ExprAtri2{[2]}....

Sel : Selector, puede ser un OID (identificador de un objeto) o una variable, sus posibles valores son OID's de objetos individuales. Sel0 es obligatorio.

ExprAtri : Expresiones Atributos, son nombres de atributos o "variables de atributos", cuyo rengo de valores posibles son los nombres de atributos

Un **"Camino de Base de Datos"** es una secuencia finita de objetos de la Base de Datos. Una instancia de una expresión de camino se obtiene sustituyendo un identificador de objeto por cada selector variable, y un nombre de atributo, por cada "variable de atributo".

Una **"expresión de camino"** es usada como un predicado booleano, y una instancia de una expresión de camino es verdadera o falsa, dependiendo si es o no satisfecha por algún "Camino de Base de Datos", es decir que exista una secuencia de objetos en la Base de Datos que la cumpla.

El usuario puede realizar consultas al esquema de Base de Datos sin tener un conocimiento completo de la representación interna:

```
SELECT Y FROM Persona X
WHERE X.Y.Ciudad [ ' La Plata ']
```

Y = atributo

La respuesta es el conjunto de todos los atributos Y, tal que para algunos objetos X de clase Persona la instancia X.Y.Ciudad['La Plata'] es verdadera.

Las expresiones de camino se pueden comparar usando los operadores =, !=, >, < ,etc.. Pero como las expresiones representan conjuntos, pueden ser restringidos con los cuantificadores Existe o Todos. Por ejemplo teniendo en cuenta el esquema 1:

-Juan13. MiembrosFamilia.Edad *Existe* > 20
(-Juan13 es un OID)

es verdad cuando algún miembro de la familia del objeto -Juan13 son mayores a 20 años. Las expresiones de camino pueden ser comparados usando los comparadores de conjuntos : *contiene*, *subconjunto*, *contienelgual*,y *subconjuntolgual* . Para todos los ejemplos tener en cuenta el esquema de Base de Datos del ejemplo 1:

```
SELECT X FROM Automóvil Y
WHERE Y.Fabricante[X] y X.Gerente.Vehículos.Color contienelgual { 'Azul' , 'Rojo' }
```

No es necesario definir el rango de X, puede ser inferido de la expresión de camino, que X es de tipo Empresa. La consulta retorna las empresas fabricantes de vehículos cuyos gerentes posean vehículos de color azul y rojo, fabricados por la empresa en donde trabajan.

Las expresiones de camino pueden ser argumento de funciones definidas por el sistema, tales como: *Sumar*, *Cantidad* o *Promedio*. Por ejemplo: Encontrar todos los empleados que tengan un salario menor a $15.000 y tengan más de cuatro (4) miembros de familia y que todos vivan en la misma casa.

```
SELECT X  FROM  Empleado X
WHERE  Cantidad ( X.MiembrosFamilia) > 4  y  X.Salario < $15.000
       y  X.residencia=  Todos  X.MiembrosFamilia.Residencia.
```

Las cláusulas SELECT pueden contener una lista de expresiones path y la cláusula WHERE puede poseer varias combinaciones de condiciones booleanas. Podemos especificar JOINS.

```
SELECT X,Y   FROM  Empresa X
WHERE  X.Nombre =  Existe   X.Grupos.Empleados(Y).Nombre
```

Produce tuplas consistiendo de un objeto Empresa y un objeto Empleado tal que los empleados tienen el mismo nombre que la empresa en la que trabajan. Esto es un Join, el cual involucra la comparación de dos (2) atributos que comparten un dominio común.

1.3.1.1.1.- Creando Nuevos Objetos.

Las consultas consideradas retornan relaciones, conjuntos de tuplas de identificadores de objetos (Oid's). Las tuplas en sí mismas no poseen Oid's.

Podríamos ver a las nuevas tuplas, resultado de alguna consulta, como nuevos objetos; para ello debería asignarle un nuevo Oid a las nuevas tuplas producidos por las consultas. Por ejemplo:

```
SELECT  EmpSalario = W.Salario   FROM  Empresa X
OID FUNCTION OF X,W
WHERE  X.Grupo.Empleados(W)
```

La cláusula SELECT da nombres explícitos de atributos para la relación de salida (respuesta), en este caso es el atributo EmpSalario. La cláusula OID FUNCTION OF determina un identificador de objeto para cada tupla en el resultado.

1.3.1.2- O.O.P.C. [BER92]

En este trabajo se pone de manifiesto la importancia de la identidad de objetos, estructuras complejas, métodos y jerarquía de clases en el momento de diseñar un lenguaje de consultas. Define las características de un Lenguaje Orientado a Objetos basado en el

Cálculo de Predicados Orientado a Objetos (O.O.P.C.). Los autores basan el O.O.P.C. en el cálculo de predicados relacional, enriquecido con características orientadas a objetos.

Una expresión O.O.P.C. tiene la siguiente forma:

{ cláusula destino , cláusula de rango , cláusula de calificación }

Las cláusulas destino y de rango son obligatorias, mientras que la de calificación es opcional.

cláusula destino : especifica lo que debe ser retornado por la consulta. Tiene una de las siguientes formas:

- ✓ **v** , denota una variable de objeto, limitada a alguna clase en la cláusula de rango.
- ✓ **v.Ai** , donde **Ai** denota el nombre de una propiedad (atributo) de la clase sobre la cual **v** tiene rango.

cláusula de rango : define el 'binding' de una variable de objeto. La estructura de la cláusula es:

- ✓ **v / C** , donde **v** es una variable de objeto y **C** es el nombre de una sola clase .

El binding define :
- el tipo de la variable v.
- el rango de la variable : los objetos para la variable se tomarán de la colección de instancias correspondientes a la clase C .

cláusula de calificación : es una fórmula bien formada, definida como una combinación booleana de predicados usando los conectivos lógicos $\wedge$, $\vee$, $\neg$. Los predicados tienen la siguiente estructura:

a)- Predicados Atómicos

i) VERDADERO
ii) FALSO
iii) T1 θ T2 , donde T1 puede ser constante, una variable de objeto 'v' o un término v.Aj (Aj propiedad de la clase ligada a v); y θ es un operador de comparación que depende del tipo de los operandos

b)- Predicados Cuantificados

i) ϕ cláusula de rango (cláusula de calificación), donde ϕ es el cuantificador universal ($\forall$) o el existencial ($\exists$).

Ejemplos: A partir del esquema definido como ejemplo1,

Retornar el nombre de todos los documentos que contienen en el resumen el objetivo del proyecto "Migración de Bases":

{ v.nombre ; v/Documento ; ∃ p/Proyecto
(v.resumen CONTAINS p.objetivo ∧ p.nombre = "Migración de Bases")

En los Manejadores de Bases de Datos las consultas generalmente son usadas para restringir el conjunto de objetos de una clase dada.

En B.D.O.O. hay distintos tipos de igualdad, este modelo maneja ambos, igualdad idéntica de objetos e igualdad por valor de objetos.

En O.O.P.C. se define = = como operador de igualdad idéntica
= como operador de igualdad por valor

- Predicados sobre propiedades de los objetos:

Las estructuras pueden ser complejas, por lo tanto permite que sean anidadas en profundidad.

1) Predicados dependientes del constructor:

Los predicados para propiedades multivaluadas en O.O.P.C. son pertenece (∈) e incluido (⊂) .

• Pertenece: y ∈ x.P , x e y variables de objeto ,y P propiedad multivaluada del objeto denotado por x.

• Incluido: y.P ⊆ x.P' , x e y variables de objeto, y P y P' son propiedades multivaluadas.

Ejemplo teniendo en cuenta el esquema 1: Retornar los nombres de todos los empleados que son miembros del staff del equipo llamado Base de Datos y son investigadores :

{ x.nombre ; x/Empleado ; ∃ e/Equipo
(x.categoría = 'Investigador' ∧ x ∈ e.staff ∧ e.nombre='Base de Datos') }

2) Cuantificadores restringidos:

Al haber propiedades multivaluadas, las consultas pueden permitir la restricción de objetos, especificando que los objetos verifiquen un cierto predicado. Para manejar este tipo de restricciones introducen en O.O.P.C. un cuantificador de restricción, que es un

cuantificador universal o existencial con una cláusula de rengo asociada, en la cual una variable de objeto es acotada a los objetos que son miembros de una propiedad multivaluada. Los cuantificadores de restricción se definen:

- $\forall$ x / y.P , cuantificador universal de restricción, y

- $\exists$ x / y.P , cuantificador existencial de restricción.

Con x e y variables de objeto, P propiedad multivaluada. Estos cuantificadores ligan el objeto denotado por x a la clase que es dominio de la propiedad P, expresando el requerimiento de que el objeto sea un elemento de un conjunto de valores de otro objeto.

Ejemplo: Retornar los nombres de todos los equipos tal que todos los empleados del staff tienen un salario > $500:

$$\{y.nombre\ ;\ y/Equipo\ ;\ \forall\ x/y.staff\ (x.salario > 500)\ \}$$

Para garantizar la seguridad de la cláusula restringida, la variable y debe estar acotada a una clase en la cláusula de rango o a una propiedad multivaluada en alguna cláusula de restricción.

Las cláusulas de restricción no son construcciones básicas:

- ✓ $\forall$ x/y.P pred(x) $\equiv \forall$ x/C' (~ (x $\in$ y.P) $\vee$ pred(x)), y
- ✓ $\exists$ x/y.P pred(x) $\equiv \exists$ x/C' ((x $\in$ y.P) $\wedge$ pred(x)), con pred(x) un predicado sobre la variable de objeto x, C clase a la cual y está acotada y C' clase de dominio de la propiedad P.

- Navegación:

El modelo permite la navegación a través del Grafo de Agregación mediante las expresiones de camino. Esto corresponde a la noción de **Join** en lenguajes relacionales.
Para este ejemplo ver en el esquema 1 (Apéndice A) el Grafo de Agregación de clases: Retornar los nombres de todos los proyectos que tienen como reporte un documento escrito el día 5:

$$\{\ p.nombre\ ;\ p/Proyecto\ ;\ \exists\ u/Documento\ (p.reporte == u)$$
$$\wedge\ \exists\ w/Fecha\ (u.fecha == w.día = 5)\}$$

Se incorporan dos variables existenciales (u y w) y dos predicados de join (p.reporte = = u $\wedge$ u.fecha = = w $\wedge$ w = 5). La expresión de la consulta puede simplificarse usando la **función punto:** dada una variable x y una propiedad simple p, x.p retorna el objeto que es valor de la propiedad p de x. La especificación de un objeto a través del uso de funciones punto anidadas se denomina **expresión de camino**.

Ejemplo:

{ p.nombre ; p/Proyecto ; p.reporte.fecha.día=5 }

Una expresión de camino puede usarse en cualquier predicado donde se espere la especificación de un objeto.

La navegación a través del grafo de agregación permite retornar propiedades multivaluadas en el resultado de una consulta.

- Métodos:

Una clase puede encapsular un procedimiento o función que especifica un cálculo de datos complejo. La invocación de un método incluye parámetros (una propiedad puede verse como un método sin parámetros) que pueden ser constantes, variables de objeto o expresiones de camino. La invocación de un método retorna un objeto como parámetro de salida, con lo que la invocación de un método puede usarse en cualquier lugar donde se espera un objeto. Inclusive, si el resultado es un valor booleano, el método puede utilizarse como predicado. Si el método retorna un objeto primitivo puede ser usado al final de una expresión de camino, caso contrario en cualquier lugar de la misma.

Un aspecto a tener en cuenta en el uso de los métodos para la formulación de consultas, es si los métodos producen efectos colaterales o si sólo recuperan valores. Esto no es algo sencillo de determinar, requiere un análisis del código de la implementación. Otro tema a tener en cuenta es la terminación de los métodos, ya que en B.D.O.O. es posible la invocación de un método al realizar una consulta, esto puede llevar a una evaluación de una consulta sin fin.

- Jerarquía de clases:

Se extiende el O.O.P.C. para tratar con la unión de clases en la cláusula de rango.

En este lenguaje de consultas cuando se especifica una clase en la cláusula de rango, el rango de la variable ligada consiste de todas las clases de la jerarquía que la tienen a ella como raíz. En el otro caso, que queremos restringir sólo a la clase especificada en la cláusula de rango, el nombre de la clase debe estar seguido por el operador " # ".

{ p.DNI; p/Persona #;p.edad >18}

recupera el DNI de todas las instancias de la clase Persona que **no** son instancias de la clase Empleado o Estudiantes y tienen más de 18 años.

Esta extensión requiere la capacidad de binding dinámico del procesador de consultas, porque el binding de las variables a la unión de las clases de la jerarquía requiere la determinación del tipo de la nueva cláusula de rango en tiempo de ejecución.

Posee el operador denominado **predicado alternativo** (Class_of), reconoce la clase a la que pertenece un objeto en tiempo de ejecución. Es posible aplicar diferentes predicados a diferentes miembros de una jerarquía ligados a una variable de objeto:

Class_Of (x) = [C1:P1 ; C2:P2 ; ....; Cn:Pn],

donde cada Ci:Pi se denomina **dominio alternativo** de x.

Ejemplo: Retornan el documento de todas las personas con salario > a 20 si son empleados, o con año de ingreso igual a 1987 si son estudiantes:

{x.dni ; x/Persona; Class_Of (x) = [Empleado:x.salario>20 ; Estudiante:x.añoInscr=1987]}

Al tratar con objetos que corren sobre distintas clases pueden surgir errores. Depende del orden en que se evalúen los predicados de la consulta, puede intentarse aplicar un predicado a una propiedad que el objeto no tiene o puede dar error en ejecución. Sin una jerarquía de clases esta situación podría ser descubierta durante el chequeo estático, antes que la consulta sea ejecutada, avisando que esa propiedad (por ejemplo Año de Inscripción) no está definida para la clase.

Para ver el ejemplo tomar en cuenta el esquema del Apéndice A:

{x.dni ; x/Persona ; x.edad>20 $\wedge$ x.añoInsc=1987}

En este caso sería correcto la consulta si todas las personas mayores a 20 años fueran estudiantes. Como no es el caso, entonces aparece un error en ejecución cuando el predicado "Año de Inscripción= 1987" es aplicado a un objeto que no es estudiante.

Hay tres posibles soluciones :

- ✓ Suspender la evaluación de la consulta y tratarlo como un error,
- ✓ Imponer que las restricciones sobre jerarquías de clases sólo se hagan con predicados alternativos,o
- ✓ devolver cierto valor indefinido (significado **no aplicable**)

La restricción de que sólo una clase puede especificarse en la cláusula de rango a sido flexibilizada, introduciendo la posibilidad de consultar la unión de clases que forman la jerarquía. Esto restringe la consulta a un subconjunto de clases de tal jerarquía.

Ejemplo: Retornar los documentos de empleados y personas que viven en la ciudad de La Plata; los empleados deben haber sido contratados desde 1988 y los estudiantes haberse inscripto en 1988, del esquema de B.D. del esquema 1:

{ p.dni ; p/(Empleados# $\vee$ Estudiantes#) ; p.residencia.ciudad = "La Plata" $\wedge$

Class_Of (p) = [Empleado# : p.AñoInContratado = 1988 , Estudiante# : p.inscripto = 1988]}

Otro rasgo importante al tratar con jerarquías es la posibilidad de **sobreescribir** (overriding) las definiciones de los métodos y propiedades heredados. Esto puede producir problemas si la redefinición pierde sentido de herencia (en términos de sustitución). Pueden usarse los predicados alternativos (Class_of) para situaciones de sobreescritura. Un predicado alternativo para cada dominio de la propiedad en la jerarquía de clases debe ser formulado para construir consultas. Un problema similar al de las propiedades surge al modificar las implementaciones de los métodos bajando en la jerarquía, produciendo cambios en las subclases.

Si se trabaja con **herencia múltiple**, como en ORION, pueden surgir problemas si una clase hereda propiedades o métodos que se denominan igual, en dos o más superclases. Puede ocurrir que tengan distintos dominios en las superclases, y traer conflictos si se formula consultas sobre la jerarquía que comienza en una de sus superclases , involucrando la propiedad con el mismo nombre, la misma solución anterior (con predicados alternativos) puede ser aplicada.

Los mismos inconvenientes surgen si en la invocación de un método O.M(O1,O2,....On) O u Oi son expresiones de caminos denotando una propiedad con dominios alternativos. Se resolvería de manera similar, por ejemplo, con la invocación de un método alternativo. En la invocación de métodos alternativos hay una invocación a métodos por cada una de las clases al cual el objeto denotado por O es ligado.

- Joins y proyección genérica:

Una consulta puede involucrar una o más clases mediante el uso de relaciones predefinidas descriptas explícitamente en el grafo de agregación del esquema. Dicho grafo describe Joins implícitos entre clases. Una expresión de camino es un Join.

Considerar el esquema de B.D. del Apéndice A y la siguiente consulta: recuperar todos los nombres de los empleados que vivan en la misma ciudad de la empresa en donde trabajan. Esta consulta no puede ser expresada a través de una expresión de camino porque no hay un camino que conecte el domicilio de la propiedad de la clase *Empresa* con la propiedad residencia de la clase *Empleado.* Por esta razón en algunos modelos de datos Orientados a Objetos son admitidos Joins explícitos entre dos clases con propiedades con el mismo dominio. Para permitir esto en O.O.P.C. se flexibiliza la restricción sobre las cláusulas de rango y fuente para que admitan más de una variable definida en cada uno, con la restricción que las variables de la cláusula fuente debe ser un subconjunto de las variables de la cláusula de rango.

{ e.nombre ; e/Empleado ;m/Empresa; e ∈m.staff ∧ e.ciudad = = m.ciudad }

La igualdad por valor o la igualdad idéntica pueden usarse para definir los Joins. Dos propiedades o variables de objeto pueden usarse en un join, únicamente si sus dominios son compatibles (son el mismo dominio o uno está incluido dentro del otro).

Otra extensión necesaria para tener el poder expresivo de un lenguaje relacional es la **proyección** de más de una propiedad.

Ejemplo : Retornar nombre, ciudad y categoría de todos los empleados que cobren más de $200.

{ x.nombre, x.ciudad, x.categoría ; x/Empleado ; x.salario>200 }

- Consultas recursivas

Como consecuencia de que la base de datos puede contener ciclos, se introducen las consultas recursivas. Se presenta una forma de recursión que corresponde a la noción de recursión lineal.

El operador de recursión en O.O.P.C. es ρ. Dado un objeto O, instancia de una clase C, y una propiedad p de O, O.ρ(p) define un conjunto de objetos que, o son valores de la propiedad p del objeto O, o son valores de la propiedad p de algún objeto O' el cual es un valor de la propiedad p del objeto O, etc.

Ejemplo: Ver esquema 1 para observar la recursión en la clase Empleado. Retornar todos los empleados tal que alguno de sus jefes cobre más de $100:

{x.nombre ; x/Empleado ; ∃ y/Empleado (y ∈ x.ρ(jefe) ∧ y.salario>$100) }

también : {x.nombre ; x/Empleado ; ∃ y/ x.ρ(jefe) ∧ y.salario>$100) }

Si una de las propiedades involucradas en el operador ρ tiene subclases, el resultado de la consulta puede ser un conjunto heterogéneo de objetos, y pueden surgir los problemas presentados para las jerarquías de clases.

- Resultados de consultas

En la definición del O.O.P.C. se restringió a una sola clase en la cláusula de rango y a lo sumo una de las propiedades de la clase especificada, en la cláusula de rango. La razón para esta restricción es que permite definir el resultado de una cláusula como una colección de objetos que son instancias de la misma clase base. Por ejemplo: Recuperar las categorías de todos los empleados que ganen más de $10.000, el resultado es una colección de objetos primitivos, instancias de la clase string.

La ventaja de esta primera aproximación del O.O.P.C. es que es un lenguaje cerrado. Las extensiones definidas posteriormente no permiten esta definición, para resultados de consultas, ya que se requiere una definición más compleja.

Para tener un lenguaje clausurado, se debe permitir que el nombre de una consulta Q pueda aparecer en el lugar de una clase básica en una consulta, por ejemplo, en la cláusula de rango. Como se dijo antes, la cláusula de rango define el tipo y el rango de la variable ligada, por lo tanto el resultado de la consulta debe ser **una clase** con una estructura de tipo y una extensión bien especificada.

Por lo tanto el resultado de una consulta podría formar una clase nueva donde el nombre de la consulta sería el nombre de la clase y :

1. Un nuevo identificador de objeto (OID) es provisto para cada objeto del resultado; el conjunto de estos objetos es la extensión de la nueva clase.

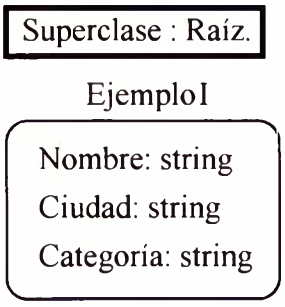
2. La estructura del tipo de la nueva clase es determinado por las siguientes reglas:

   - Para cada propiedad especificada en la cláusula destino (x.Ai) existe una propiedad en la nueva clase, con el mismo dominio.
   - Para cada variable especificada en la cláusula destino sin propiedad asociada, se genera una propiedad que tiene como dominio el rango de la variable; si el rango es especificado como la unión de subclases, el dominio es especificado como la superclase común más específica (la raíz del subarbol considerado).
   - la superclase de la nueva clase es la raíz de la jerarquía (Clase Tope).

Ejemplo I: Recuperar nombre, ciudad de residencia y categoría de todos los empleados que ganan más de $10.000.

{ e.nombre, e.residencia.ciudad, e.categoría; e/ Empleado; e.salario > 10.000 }

El resultado de esta consulta tiene tres (3) propiedades (una por cada atributo proyectado). El dominio de cada propiedad en la nueva clase es el mismo que el de la clase base. Los nombres de las propiedades en la nueva clase están dados por las expresiones de camino que especifica la propiedad en la cláusula fuente de la consulta.



Ejemplo II : Recuperar para cada empleado el grupo del cual él es miembro:

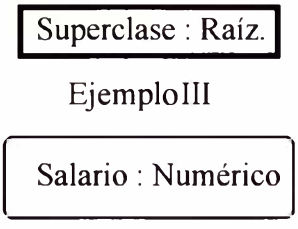
{ x, t ; x /Empleado, t/ Equipo ; x ∈ t. staff }

Se puede utilizar el resultado de una consulta en otra consulta. A la nueva clase, definida por una consulta, no se el pueden asociar métodos definidos por el usuario. Los únicos métodos disponibles son los métodos definidos por el sistema que permiten recuperar y modificar propiedades de la clase.

El resultado de una consulta es guardado, por lo tanto la definición de la clase y los objetos pertenecientes a ella son hechos persistentes.

La ventaja de esto es que las consultas son únicos y no hay ambigüedad para todo tipo de consultas. La gran desventaja es que para consultas simples se producen subconjuntos de clases básicas. Por ejemplo:

{ x; x/ Empleado; x.salario > $10.000 }. Crea la clase Ejemplo III:



Sería útil que para tipos particulares de consultas, los cuales retornen subconjuntos de objetos de clases existentes, admita que el resultado de la consulta sea una colección de objetos existentes en vez de una nueva clase. De ésta forma no generaría nuevos OID's para sus miembros. El tipo de la colección correspondería al dominio de la propiedad o a al variable especificada en la cláusula Fuente, y la extensión sería un subconjunto del dominio de la clase de su tipo.

Otras soluciones que no requieren nuevos OID's son adoptados por otros sistemas a expensas de perder la clausura del lenguaje.

CAPITULO II

Hipertextos: Esquemas de representación y acceso a los datos

2.- HIPERTEXTOS: ESQUEMA DE REPRESENTACION Y ACCESO A LOS DATOS

2.1.- Introducción

La manera más simple de comprender un hipertexto es contrastarlo con un texto tradicional como podría ser un libro. La forma de todos los textos tradicionales impresos es secuencial, por lo tanto tiene una única manera de leerlo; primero la página uno, luego la página dos y así sucesivamente. Pero recordemos que mientras leemos un texto tradicional, es común encontrar notas de pie de página, referencias entre secciones, referencias bibliográficas, por ejemplo, podríamos encontrar la siguiente referencia: "ver Capítulo 2" y esto nos lleva a leer el texto de una manera que no es completamente secuencial.

Los hipertextos son no secuenciales, es decir, no existe un orden que predetermine la secuencia en la cual el texto será leído, no hay una única secuencia. Además en hipertextos se permite al escritor hacer tales referencias y al lector se le permite tomar sus propias decisiones acerca de qué referencia seguir y en qué orden. Tales referencias están representadas por links las cuales veremos con más detalle en el punto 2.2.2. Debido a este motivo Ted Nelson, pionero de los hipertextos, los define [CON87] como "una combinación de texto en lenguaje natural con la capacidad de las computadoras para realizar saltos interactivos o displays dinámicos de un texto no lineal, los cuales no pueden ser impresos convenientemente en una página convencional.

Supongamos que la Figura 2.1 es un hipertexto con información acerca de artistas y sus obras y nosotros estamos viendo sólo parte del mismo, donde el rectángulo con el título Leonardo da Vinci provee la siguiente información:

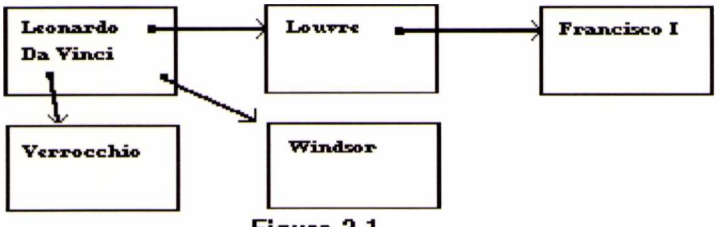


Figura 2.1

Leonardo Da Vinci

Pintor, dibujante, escultor, arquitecto, ingeniero e inventor ital. Su vida artística se divide en cuatro períodos: florentino (1452-82), de (1467-78) frecuentó el taller de ***Verrocchio*** *y la escuela de Pollaiuelo, milanés (1483-1499), de vida errante (1500-16) y fr. en la corte de Francisco I (1516-19). En Florencia pintó dos obras incompletas: San Jerónimo (pinacoteca vaticana) y adoración de los magos (Uffizi) .... En Milán pintó las dos Virgen de las Rocas (****Louvre*** *y National Gallery)... En 1502 en Florencia realizó Santa Ana con la Virgen y el Niño (Louvre) , cartas topográficas y proyectó el desvío del curso del Arno para construir un canal que lo hiciera navegable hasta el mar. En 1503 pintó la batalla de Anghiari y la Gioconda (Louvre)... En Francia terminó su Anatomía y realizó los dibujos del Fin del Mundo (****Windsor****). Su gran innovación pictórica fue el sfumato y la asociación de colores contiguos.*

Al leer este texto el lector podría interesarse por los temas que indican las palabras resaltadas, los cuales se comportan como botones en pantalla que proveen más información. Por ejemplo si el lector seleccionara el botón Louvre se verá la información de Louvre.

Louvre

Museo, antigua residencia real iniciada por ***Francisco I*** *y que finalizó durante el II imperio. Como Museo data de 1791 por decreto de la Asamblea Constituyente. Su contenido se puede dividir en seis departamentos: antigüedades or.(-iv milenio al s. xiv), egipcias or.(-iii milenio al s. vi), Grecia y Roma (-ii milenio al s. iv), pintura donde se hallan representadas todas las escuelas europeas desde la EM hasta la contemporánea (excepto los impresionistas) y escultura de la EM a la moderna. Posee además una biblioteca de más de 80.000 vol, archivo, laboratorio (1931), escuela (1881), etc.*

El lector podría elegir en este momento más información acerca de Francisco I o bien volver al punto anterior e interiorizarse en Windsor o Verrocchio.

Francisco I

(1708-65). Emperador de Alemania (1745-65). Casada desde 1736 con la futura emperatriz MA. Teresa....

Windsor

Castillo de Gran Bretaña (Inglaterra), consta de 94608 habitaciones. Durante el reinado de Victoria I se convirtió en la residencia preferida de los monarcas británicos. Su castillo es panteón real y guarda una vasta colección de arte británico.

Verrocchio

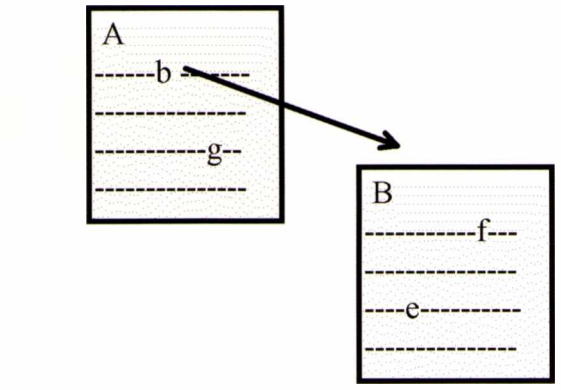
llamado II. Escultor y pintor ital . Discípulo de Donatello y maestro de Boticcelli y Leonardo. En escultura se destacan su David (Bargello, Florencia) y la estatua ecuestre de Bartolomeo Calleoni(h 1483), en Venecia como pintor es notable su tabla Bautismo de Cristo (h. 1470, Uffizi, Florencia) y la Madonna con el Niño y Los Angeles (National Gallery, Londres).-

Los hipertextos son cada vez más populares y su aceptación se incrementa día a día más allá del mundo informático. Un ejemplo de esto es la Word Wide Web, uno de los servicios más populares de información hipertexto / hipermedia de Internet, que permite a una amplia variedad de usuarios, desde investigadores, abogados, médicos hasta filatelistas acceder a información situada en cualquier lugar del mundo.

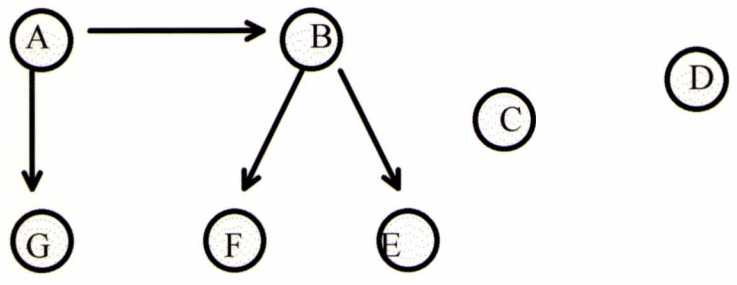
2.2.- Definición de Hipertextos

Con el avance de la tecnología, la noción de texto ha ido avanzando hasta llegar a organizaciones más complejas. Los nuevos mecanismos permiten relacionar grandes volúmenes de información en diferentes sentidos, para luego poder consultarlos en la forma en que al lector / usuario le resulte más conveniente.

Definición:
Un Hipertexto según [CON87] y en el que también se basaron [NIE95] [AMA92] [BRA92] es un conjunto de porciones (chunks) de información llamadas Nodos que se relacionan entre sí mediante referencias llamadas Links, lo que en conjunto forma una red de nodos y links, la cual no puede ser convenientemente impresa en una página convencional. La capacidad de tener links entre diferentes nodos permite una organización no lineal del texto, que podrá ser navegada en la forma en que el lector desee.



Representación de un hipertexto en Pantalla



Red Hipertexto

Figura 2.2

Primitivamente los hipertextos representaban información sólo en forma de texto. Luego, diversas aplicaciones fueron incorporando información en forma de gráficos, animación, video y audio, a esto se lo llamó **Hipermedia.** Actualmente en los Hipertextos puede almacenarse todo este tipo de información.

Con frecuencia se confunde sistemas hipermedia con sistemas que incluyen efectos multimedia. Muchos sistemas multimedia se basan en desplegar video clips a un usuario pasivo que no puede navegar un espacio de información. Sólo cuando el usuario siente que puede moverse libremente a través de la información acorde a sus necesidades será un hipertexto.
Según J. Nielsen [NIE95] la diferencia entre sistemas multimedia e hipermedia es comparable con ver el video de un viaje (usuario pasivo) y ser un turista (usuario activo).

A diferencia de los textos tradicionales donde el usuario debe leer la información secuencialmente, en hipertextos el usuario tienen la posibilidad de elegir como leerá el mismo, es decir, el orden en el que irán seleccionando la información relacionada. Esta actividad es llamada navegación del hipertexto.

Las bases fundamentales de hipertextos no son complicadas, la información está organizada en nodos (punto 2.2.1) y estos pueden estar relacionados a través de links (punto 2.2.2). Sin embargo, la naturaleza de los hipertextos tienen dos problemas conocidos:
- ✓ la desorientación del usuario en la red de hipertexto y
- ✓ el overhead cognoscitivo para crear, manipular y seleccionar datos.

Estos temas serán tratados con más detalle a lo largo del capítulo.

2.2.1.- Nodos

Un nodo es una porción de información en la que se define un concepto o idea del dominio de la aplicación que se está modelando.
Los nodos son uno de los puntos fundamentales en la definición de un hipertexto, ya que estos permiten modularizarlo, lo que resulta en un punto crítico pues impacta directamente en la comprensión del lector.

Un ejemplo de nodo podría ser la definición de artista:

Artista: Persona que se dedica al arte...

Las ideas deben modularizarse en unidades de manera que :

1) una idea individual pueda ser referenciada desde cualquier lado.

2) ofrecerle al lector sucesivas alternativas de una unidad, por ejemplo más detalle, un ejemplo, referencias bibliográficas, sucesores lógicos o una relación semántica.

2.2.1.1.- Nodos Tipados

La mayoría de los sistemas hipermedia han sido diseñados para soportar tareas específicas, aún los llamados sistemas genéricos, fueron diseñados con una tarea destino en mente. Y por tal motivo enfatizan capacidades que reflejan los requerimientos de la tarea destino.
Cada autor provee clasificaciones de nodos dependientes de las aplicaciones destino del sistema definido, por ejemplo aplicaciones orientadas a la educación, ingeniería de soft, etc..
Por ejemplo, en Notecards [HAL88], un sistema hipertexto de propósito general, se define un número limitado de tipos de nodos, diferenciados por la naturaleza del contenido (texto, gráfico, animación, etc).
Conklin [CON87], un sistema orientado al diseño de soft, ofrece en su prototipo nodos tipados que ayudan a manejar interfaces y aspectos navegacionales. Por ejemplo notas, restricciones artifacts, decisiones, etc.

2.2.2.- Links

Los textos tradicionales nos obligan a escribir y leer los párrafos en una manera mayormente lineal. existen ciertas marcas que indican saltos en el flujo cuando es necesario, por ejemplo, notas de pie de página, referencias entre secciones (por ej. Ver cap.4). Esto le permite al autor decir "aquí hay información relacionada, en caso que usted esté interesado". Esta idea está representada en hipertextos por medio de links. Un link conecta dos nodos y por lo general es dirigido.

Un link tiene su origen en un punto particular del nodo, llamado anchor origen, y el destino puede ser un punto particular del nodo, llamado anchor destino o el nodo en su totalidad como puede verse en la figura 2.1 o 2.2.

Los links pueden ser con una sola dirección o bidireccionales. Casi todos los sistemas hipertexto actuales se ven limitados por proveer links con una sola dirección como se ve en la figura 2.1. Esto significa que el sistema puede mostrar al usuario los links que tiene el nodo corriente como su punto de salida pero no los que tiene como punto de arribo, es decir, el sistema le informará dónde puede ir pero no cómo llegó.

2.2.2.1.- Links tipados

Los tipos de links representan diferentes formas de relación entre los nodos. Al igual que en los nodos tipados cada autor clasifica los links en función de las tareas destino del sistema:

Sistemas hipertextos de propósito general como Notecards [HAL88], Intersect [WAN91] definen dos tipos de links: de anotación y referenciales:
Un link de tipo anotación le permite tanto al lector como al autor relacionar un nodo a una pequeña cantidad de información. Estos links se encuentran principalmente en sistemas orientados a la educación, los lectores desean tomar notas acerca de los nodos que están leyendo.
Un link referencial permite conectar una porción de información de un nodo origen con una porción de información o la totalidad del nodo destino. Este es el tipo más común de link y es el que se utiliza para recorrer el hipertexto.

Algunos sistemas hipertextos definen "superlinks" que representan relaciones con cardinalidad 1:N conectando un nodo con varios nodos destino. En este tipo de links el autor debe definir cuál será el destino al momento de navegar el link. Algunos autores como [NIE95] proponen elegir un destino al azar, otros como Intermedia presentan un menú con los posibles destinos y es el usuario el encargado de decidir cual tomará, otros autores, en cambio, presentan todos los destinos en pantalla.

2.2.3.- Navegación

Navegación es el proceso de arribar desde un nodo origen a un nodo destino mediante la activación de links. En hipertextos, es el método primario para acceder a la información.
Hablamos de navegación y no de lectura para enfatizar que el usuario puede determinar el orden en que leerá los nodos.
Por ejemplo:

Pintor, dibujante, escultor, arquitecto, ingeniero e inventor ital. Su vida artística se divide en cuatro períodos: florentino (1452-82), de (1467-78) frecuentó el taller de ***Verrocchio*** *y la escuela de Pollaiuelo, milanés (1483-1499), de vida errante (1500-16) y fr. en la corte de Francisco I (1516-19). En Florencia pintó dos obras incompletas: San Jerónimo (pinacoteca vaticana) y adoración de los magos (Uffizi) .... En Milán pintó las dos Virgen de las Rocas (****Louvre*** *y National Gallery)... En 1502 en Florencia realizó Santa Ana con la Virgen y el Niño (Louvre) , cartas topográficas y proyectó el desvío del curso del Arno para construir un canal que lo hiciera navegable hasta el mar. En 1503 pintó la batalla de Anghiari y la Gioconda (Louvre)... En Francia terminó su Anatomía y realizó los dibujos del Fin del Mundo (****Windsor****). Su gran innovación pictórica fue el sfumato y la asociación de colores contiguos.*

las palabras resaltadas representan un anchor origen que al ser activado nos permite acceder a algún otro documento que describe su significado.

Cuando los usuarios navegan el hipertexto a través de los links, con frecuencia tienen necesidad de volver al nodo previamente visitado. La mayoría de los sistemas hipertexto cuentan con un elemento que asiste a la navegación, que es la facilidad de backtrack.

2.2.4.- Representación visual de nodos y links

Los nodos pueden ser representados en pantalla de dos formas diferentes, en frames o en ventanas (scrolling windows).
En frames la información está representada en un espacio fijo del monitor, que por lo general es la pantalla entera, existiendo una correspondencia 1 a 1 entre un nodo y su representación en la pantalla. Entonces podría suceder que el autor del hipertexto tenga que dividir su información en varios frames, tal es el caso de KMS y las "cards" de HyperCard .
En cambio, en ventanas el nodo puede ser tan grande como sea necesario, pero sólo podrá ser mostrado en pantalla una porción del mismo, es por eso que el usuario requiere además de los mecanismos de hipertextos , usar un mecanismo de scrolling para poder desplegar en pantalla la parte deseada del nodo, tal es el caso de Guide e Intermedia [NIE95].
Por lo general la distinción de representación de nodos en frames o en ventanas no es tan clara. Hypercard, por ejemplo, es mayormente basado en frame, pero incluye facilidades de "scrolling" de campos de texto. Hyperties, en cambio, usa la pantalla completa sin la facilidad para hacer "scrolling", pero permite pasar a la página anterior y a la página siguiente, en caso de tratarse de un nodo muy grande.

La representación visual del link es, por lo general, en el anchor origen que sirve para que el usuario detecte su presencia (botones), estos pueden ser: palabras resaltadas, un botón, un dibujo, un icono, un cambio en el formato del cursor, etc.
Los links son activados cuando el lector "clickea" un botón en pantalla que representa el origen del links.

CAPITULO III

Análisis del Estado del Arte

3.- ANALISIS DEL ESTADO DEL ARTE

3.1.- Introducción

En este capítulo haremos un análisis de la situación actual de los hipertextos. En particular, trataremos algunos de los puntos críticos como la desorientación que tomamos como motivación para la elaboración de nuestro proyecto.

En la primer sección presentamos los beneficios y problemas del uso de hipertextos, en la segunda sección veremos como algunos autores intentaron mejorar la navegación, incorporando entre otras cosas, conocimiento a la estructura y lenguajes de consultas.

3.2.- Beneficios y problemas del uso de hipertextos

En la actualidad los hipertextos son ampliamente difundidos, esto se debe a las ventajas que presentan los mismos con respecto a la forma de acceder a la información. Entre las ventajas podríamos mencionar :

- ✓ Los hipertextos hacen explícitas las relaciones implícitas que tienen los textos.
- ✓ Ofrece la posibilidad de acceder de una forma más natural a fuentes de información grandes y/o complejas.
- ✓ Las referencias proveen consistencia ya que si la información se cambia de lugar, aún en otro documento, el link aún provee acceso directo a la referencia.

Pero además las aplicaciones hipertextos traen aparejados dos grandes problemas:

a) overhead cognoscitivo
b) problema de desorientación

a) Overhead cognoscitivo
Según [WAN91] el problema del overhead cognoscitivo es el esfuerzo que debe realizar el lector para alcanzar el punto donde desea navegar. Esta dificultad se debe en gran parte a que los links no pueden expresar correctamente la semántica existente entre los nodos.
Como resultado de esto es muy posible que los usuarios se pierdan en el hiperespacio. Esto puede provenir de la confusión producida cuando el lector le da a la estructura del hiperdocumento una interpretación diferente de la que intentó darle el autor.

b) Desorientación
Tradicionalmente el método de consulta en un hipertexto es el **navegacional,** el cual se logra siguiendo links de nodo a nodo. Normalmente el usuario ve en pantalla los posibles lugares a navegar, luego elige el link que le provee mayor información acerca del tema a consultar.

Existen aplicaciones donde el acceso a la información a través de la navegación es adecuado. Por ejemplo, podríamos mencionar:

- ✓ Aplicaciones pequeñas con pocos usuarios, usando intensivamente una red. Como la red es pequeña y familiar los usuarios tienen poco problema en localizar la información.

- ✓ Aplicaciones donde el autor provee una guía para la exploración de la misma. Si no incluye instrucciones navegacionales entonces la red debe ser explorada en forma no directa, por ejemplo un juego donde trata de ocultarse información.

A diferencia de estas, existe una amplia variedad de aplicaciones donde el acceso a la información mediante la navegación no es tarea trivial. Las mismas se caracterizan, por lo general, por el gran tamaño de sus redes, por no ser familiar para los usuarios que la navegan y por estar heterogéneamente estructurada.
En estos casos el acceso navegacional resulta problemático, pues los usuarios tienden a perderse en la red, buscando alguna información particular. Con frecuencia los usuarios pueden describir exactamente lo que están buscando, pero simplemente no pueden encontrarlo.

Estos problemas se deben básicamente a que la semántica de los modelos no aporta demasiada información. En la forma en que se definieron los hipertextos anteriormente, sólo sabemos que hay uno o más links para navegar entre nodos, o que tenemos una anotación, etc. Es decir, la información que podemos obtener se ve limitada por los tipos de nodos y links definidos en el modelo.
Por ejemplo, si un artículo habla sobre pintores que realizaron ciertas obras, entonces podríamos crear un link entre pintores y obras, pero luego podríamos erróneamente pensar que esta relación representa los pintores que compraron obras. Esto se debe a que los nodos y links tal cual se han descripto hasta ahora son incapaces de expresar la semántica existente.

3.3.- Intentos por solucionar los problemas presentados

Son varios los autores que intentaron solucionar estos problemas, por ejemplo, en Notecards [HAL88] existen nodos browser que permiten ver la red en su totalidad. Estas vistas permiten al usuario buscar visualmente la información deseada y moverse directamente a áreas de la red donde se encuentra. Pero esto no siempre da resultado, porque en el caso en que la red es suficientemente grande, no es posible localizar la información, aún sabiendo exactamente lo que está buscando.

Sin embargo, [NAN91] al igual que [WAN91] sugieren que los problemas de desorientación y overhead cognoscitivo pueden disminuir aumentando las capacidades de navegación existentes a través de un mecanismo de acceso directo basado en consultas.

Existen dos tipos de mecanismos de búsqueda / query que son necesarios para sistemas hipermedia. Estos son búsqueda al contenido y búsqueda a la estructura.
En el primer mecanismo (al contenido) todos los nodos y links en la red son examinados independientemente, para seleccionar aquellos cuyo contenido o propiedades satisfagan una consulta dada. La búsqueda al contenido ignora la estructura de la red hipermedia. A diferencia en la búsqueda a la estructura se examina la red hipermedia buscando aquellas subredes que matcheen con un pattern dado.
Por ejemplo una consulta simple a la estructura podría ser:
- Todas las subredes que contienen 2 nodos conectadas por un tipo de link determinado donde el nodo destino contiene la palabra "hipertexto".

Además de su rol para localizar información, los mecanismos de búsqueda y query son utilizados para filtrar información en la interfase hipermedia. esto significa que el usuario especifica una consulta para tener acceso sólo a información que es de su interés.
Los browsers presentado en Notecards [HAL88] trabajan de esta manera, con la desventaja que el tipo de consultas a realizar está limitado a localizar un texto particular.

Luego del análisis de diferentes hipertextos notamos que la principal deficiencia se encuentra en consultas al conocimiento, ya que el conocimiento existente es muy reducido debido a la ausencia de tipos definidos según el dominio de la aplicación. En la mayoría de los hipertextos sólo puede localizarse un texto particular.
Por ejemplo, [WAN91] al igual que [NAN91] propone usar nodos y links tipados según el dominio de la aplicación para incorporar mayor conocimiento al hipertexto y así poder realizar consultas semánticamente más completas.

3.3.1.- Incorporación de semántica a través de tipos

Sabemos que los hipertextos contienen información y conocimiento. Información en el contenido de los nodos y conocimiento en la estructura.
Cuanto mayor es el poder expresivo de los links, mayor será el conocimiento embebido en la estructura y así podrá realizarse un acceso más eficiente a la información.

Hasta el momento sólo hemos visto nodos y links tipados según la tarea destino del sistema hipertexto utilizado, en la que sólo podíamos decir que dos nodos estaban relacionados, como una anotación , para navegar, etc, pero la semántica de la relación era muy limitada.
De ahí que surge la necesidad de definir nodos y links tipados que capturen la semántica del dominio de la aplicación.

- Nodos tipados

Según [WAN91] un **tipo de nodo** es un tipo de datos abstracto que describe la estructura interna de un documento. Cada nodo tiene características (propiedades / atributos) que lo identifican y son los que definirán su propia estructura interna.
Una instancia individual de un nodo está asociada a un tipo de nodo particular.
Estos atributos pueden ser vistos como filtros para seleccionar todos los nodos con una propiedad particular.

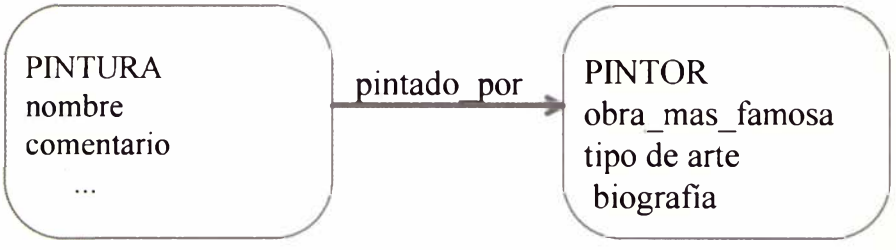
Un ejemplo de un tipo artista:

Artista:
obra mas famosa
tipo de arte
biografía

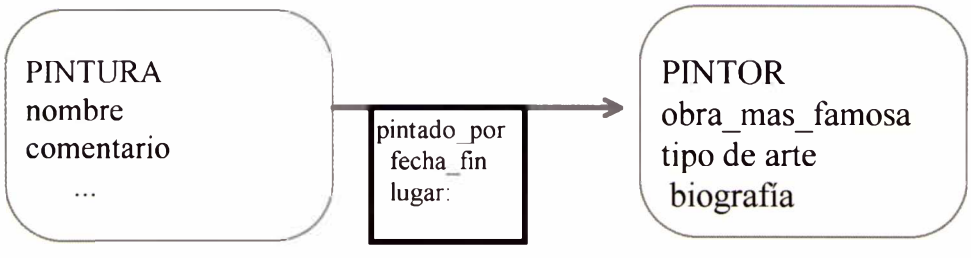
- Links tipados

La mayoría de los sistemas hipertextos presentan links planos, es decir, sólo se utilizan para conectar dos nodos y poder navegarlos. Sin embargo, los links pueden contener información adicional , tales como una fecha, un nombre, etc., la que es representada por medio de atributos en la definición del link. Podríamos decir que un link es de tipo A si su estructura interna es igual a la definición del link de tipo A.

Por ejemplo, supongamos que deseamos representar la relación que represente quién pintó una pintura determinada, entonces definiríamos un link con la siguiente estructura:



pero, supongamos que varios pintores representaron la misma obra, sólo definiremos un nodo que identifique esa pintura y un nodo para cada pintor. Si deseamos saber en qué fecha cada pintor finalizó la obra debemos representar esa información en el link de la siguiente manera:



[NAN91] al igual que [WAN91], propone incorporar conocimiento a través de tipos, representando información en el contenido de los nodos y conocimiento en la estructura.
Por ejemplo supongamos la red representada en la figura 3.1.

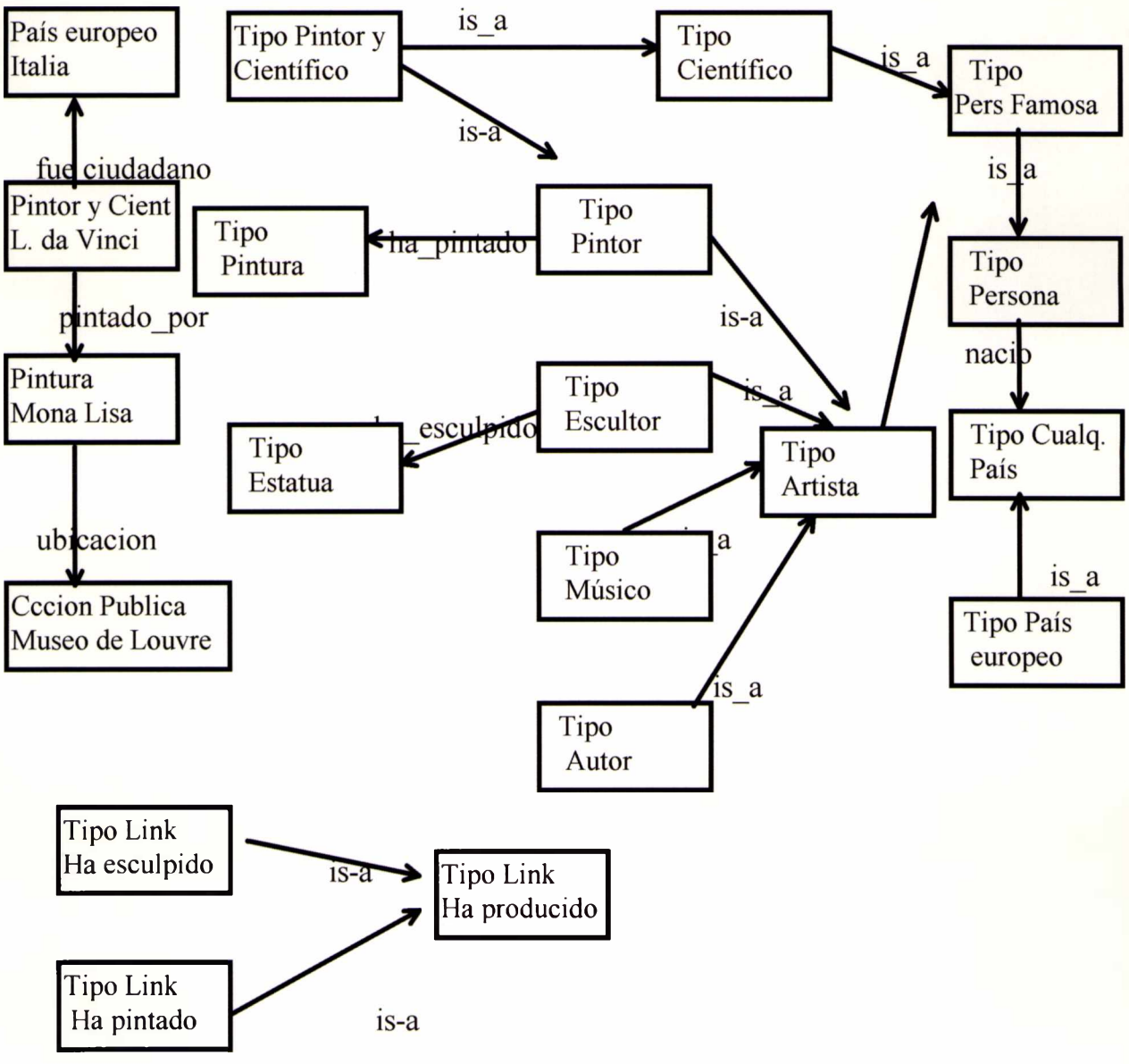


Figura 3.1

Por ejemplo:
La <Pintura> de <Mona Lisa> es sólo información. Su link de tipo <pintado por> al nodo de tipo <persona famosa> llamado <Leonado da Vinci> y su link <ubicación> a la <Colección pública> <Museo de Louvre> representa algún conocimiento.
Además con esta información podría recuperarse ubicaciones dónde los <retrato>s <pintado por> <pintores italianos> que <vivían en> el <siglo 16>.

Los modelos orientados a objetos son los más adecuados para representar conocimiento, pues permiten modelar entidades del mundo real de una forma más natural. Además gracias a su mecanismo de herencia permite simplificar la creación de tipos y con las entidades complejas provee mayor nivel de abstracción. Por ejemplo, un <Pintor Italiano> es un concepto más específico que <Pintor>, el cual es más específico que <Artista> y que <Persona Famosa>. Así es como los <Pintores Italianos> heredan todas las propiedades definidas para <Artista>s.

[NAN91] al igual que [WAN91] presentan un prototipo que utiliza nodos y links tipados para modelar hipertextos, pero no aportan información acerca de lenguajes que permitan consultarlos.

3.3.2.- Lenguajes de Consultas existentes a hipertextos

Como ya hemos mencionado cuando tratamos con una base de datos con gran cantidad de nodos surge el problema de desorientación y encontrar navegando un nodo particular no es tarea trivial. Por tal motivo se ha considerado el uso de **queries** conjuntamente con navegación para obtener más rápidamente la información buscada.

Los hipertextos tienen una gran ventaja que puede ser aprovechada en el momento de realizar consultas. Esta ventaja es su estructura.
Los presencia de links representan las relaciones conceptuales y lógicas que existen entre los nodos, con lo cual las consultas pueden ser extendidas usando información conceptual y contextual. El dominio y rango de la consulta puede ser restringida o extendida considerando la naturaleza y tipos de los links.

Como vimos anteriormente [WAN91],[NAN91] algunos autores sólo presentan un modelo de datos basados en nodos y links tipados según el dominio de la aplicación, pero no proveen información acerca de los lenguajes para realizar consultas.

En cambio otros autores [HAL88], sólo permiten realizar búsquedas de un texto o una subred particular. Este tipo de consultas además de ser muy limitadas tiene la desventaja que las mismas se realizan sobre todo el hipertexto y si la red es suficientemente grande, obtener una respuesta podría demorar mucho tiempo.

Hydesign [MAR92] presenta un modelo de datos que tiene como ventajoso el uso de nodos compuestos (SBL) y utiliza, para acceder a la información, un lenguaje de consultas orientado a objetos llamado OPAL que fue desarrollado para el OODBMS Gemstone. La desventaja es que como el lenguaje no fue diseñado para este modelo de datos, no hace uso de las ventajas del mismo.

[FUL91] presenta un modelo de datos que utiliza nodos y links tipados y un lenguaje de consultas basado en funciones.

El autor define dos tipos básicos para la definición del lenguaje

- node_list: una lista de nodos
- link_list: una lista de links.

Las siguientes constantes son utilizadas:

- this_node: el nodo corrientemente activo.
- this_link: el link en cual el usuario está interesado.
- specified_node_id: un nodo particular, especificado a alto nivel.
- this_data: el dato asociado con este nodo.
- history_list: una lista de los nodos previamente visitados.
- all_nodes: todos los nodos en la base de datos.

Las funciones son:

- node_kind(node_list, Tipo) →node_list, desde una lista de nodos, extrae aquellos que son de un tipo particular.
- link_kind(link_list, Tipo)→link_list, desde una lista de links, extrae los links de un tipo particular.
- links_from(node_list) → link_list, desde una lista de nodos extrae los links.
- destination(link_list) →node_list, determina el destino de esos links.
- rank(node_list, condición) → node_list, realiza el ranking de los nodos contra una condición específica.
- match(node_list, condición) → node_list, matchea una lista de nodos contra la condición especificada
- top(N, node_list) → node_list, selecciona los primeros N nodos de una lista.
- union(node_list, node_list) → node_list, devuelve la unión de la primera y segunda lista.
- intersect(node_list, node_list) → node_list, devuelve la intersección de la primera y segunda lista.
- subtract(node_list, node_list) → node_list, devuelve los elementos de la primera lista que no están en la segunda.

A partir de estas funciones es posible resolver cualquier consulta. Además permite definir funciones combinando las funciones ya existentes.
El principal problema de este lenguaje de consultas es que por cada consulta realiza un recorrido de toda la red, esto se debe a que una función no puede ser aplicada a un tipo de nodo particular y en redes muy grandes podría resultar un proceso muy lento.

Ahora veremos Gram [AMA92], un modelo de datos organizado como grafo, el cual podemos consultar mediante expresiones regulares sobre los tipos de los nodos y labels de las aristas.
La estructura del hipertexto es básicamente un multigrafo con labels dirigido, donde los nodos son documentos tipados y las aristas representan los links tipados entre documentos.

Además los links pueden contener atributos que le agregan información al link, tal como una fecha, dirección o distancia.

El lenguaje de consultas presentado está basado en un álgebra query donde expresiones regulares sobre los tipos de datos son utilizados para seleccionar secuencias de nodos y links que cumplan determinadas condiciones.

Entre las posibles operaciones definidas por el lenguaje encontramos básicamente las mismas que definen en SQL pero adaptadas a hipertextos:

- Renombramiento: Permite renombrar algún tipo en la secuencia de nodos y links para trabajar con mayor claridad.
- Selección: Permite seleccionar todas las secuencias de nodos y links que satisfagan las condiciones definidas, las que pueden ser aplicadas tanto a nodos como a links. Pudiendo por ejemplo obtener información sobre todos los "pintores" que "pintaron" "La Monalisa".
- Proyección: Permite proyectar sólo una parte de una secuencia de nodos y links. Pudiendo así, considerar sólo la parte de la respuesta en que está interesado y evitar de esta manera confusiones que podrían ser causadas por manejar mucha información.
- Join: Este tipo de operación permite cruzar información de dos o más secuencias de nodos y links. Permitiendo por ejemplo, resolver una consulta por partes y luego "joinearlo".
- Concatenación: Permite concatenar dos secuencias de nodos y links, si el nodo final del primero es igual al nodo comienzo del segundo.

También es posible utilizar funciones tales como Unión, Intersección y Diferencia entre conjuntos de hyperwalks con el mismo formato.

Entre las desventajas de este lenguaje encontramos que no es posible realizar consultas a la estructura de la información y como el modelo no define nodos compuestos tampoco es posible consultarlos.

Además de permitir localizar determinada información de una forma más directa, las consultas asisten a la navegación permitiendo:

- ✓ realizar múltiples pasos de navegación: en lugar de navegar a través del hipertexto seleccionando links paso por paso, el usuario podría alcanzar directamente un nodo siguiendo una secuencia de nodos y links que satisfagan un query Q.
- ✓ restringir el espacio de navegación: el usuario especifica un query y luego, sólo puede navegar a través de los links involucrados en la respuesta.

3.4.- Conclusión

A lo largo de este capítulo describimos los problemas de los hipertextos, básicamente la desorientación.
Vimos que una de las maneras de disminuir el problema es accediendo directamente a la información mediante consultas al conocimiento. Y para tal fin es necesario incorporar conocimiento al modelo mediante el uso de tipos definidos según el dominio de la aplicación.
Y por último vimos que es necesario definir un lenguaje de consultas que se adapte completamente al modelo definido.

En base a estos puntos se desarrollará el proyecto que a continuación presentamos, en el que se definirá un modelo de datos para hipertextos y un lenguaje de consultas acorde al mismo, que permita consultar tanto el modelo existente como la estructura de la información.

CAPITULO IV

Presentación del Proyecto

4.- PRESENTACIÓN DEL PROYECTO

4.1.- Introducción:

La naturaleza de las aplicaciones de Base de Datos ha ido evolucionando y en el transcurso de muy poco tiempo se las ha generado cada vez más sofisticadas. Actualmente permiten involucrar datos complejos, datos compuestos y tipos de datos provenientes de una amplia variedad de medios (imágen, audio, animación, video, texto), con estos nuevos avances se origina la representación de las denominadas *aplicaciones hipermedia*.

El objetivo del proyecto es desarrollar un modelo para diseñar aplicaciones hipermedia, conbinando los beneficios del paradigma *Orientado a Objetos* con los de hipermedia, y definir un lenguaje de consultas acorde al mismo. Durante su desarrollo, se mantendrán vigentes las características más destacables de estas aplicaciones, tales como la naturalidad que se ofrece para el acceso a la información utilizando la navegación, o la participación del lector, que a través de la selección de uno de los caminos alternativos que se le presentan, determinará el orden en que se navegará el espacio de información.

Son varios los estudios realizados sobre el problema de desorientación en una hiperbase y sus posibles soluciones, en los mismos se hace mención de diferentes formas de representar y acceder a los datos, pero observamos que en la mayoría de estos, las consultas a la información y a la estructura son muy limitadas.
A medida que fuimos avanzando en el conocimiento de los diferentes modelos para diseñar hipertextos, notamos que el aumento de la semántica es directamente proporcional a la estructuración de la información. Es así que utilizando esta premisa, desarrollaremos un modelo donde los nodos y los links sean modelados mediante clase que estructuren la información y se obtenga de esta manera mayor semántica.
Una vez que logremos este modelo de datos para el diseño de hipertextos, en el cual se pueda representar la estructura de los documentos hipermedia, su contenido y presentación, estaremos en condiciones de definir un lenguaje de consultas que atenue los problemas detectados.

4.2.- Descripción del modelo

Como lo hemos expresado anteriormente, definiremos un modelo para el diseño de aplicaciones hipermedias, utilizando el paradigma Orientado a Objetos del que podemos destacar las siguientes ventajas:

- ✓ soporta datos de cualquier tipo y complejidad,
- ✓ existe una correspondencia entre objetos del modelo y conceptos del dominio de la aplicación,
- ✓ el encapsulamiento de objetos permite mayor abstracción de la información,
- ✓ un objeto tiene una única identidad independientemente de su estado,
- ✓ la información es organizada en forma jerárquica otorgando de esta manera mayor semántica,

haciendo uso de estas características estamos en condiciones de representar información de la complejidad existente en Hipertextos.

Los dominios de las aplicaciones hipermedias estan formados por un conjunto de objetos del mundo real. Entre estos objetos podemos definir entidades, y relaciones entre ellos. Sería deseable ver a una entidad desde diferentes aspectos, para que cada lector pueda acceder a la misma información desde el/los matices que le interesan. Este modelo permite definir estas vistas de las entidades, su interfáz y relaciones entre ellas.

La construcción de aplicaciones Hipermedia en nuestro modelo consta de dos niveles de diseño:

a) Diseño de Alto Nivel, en el cual serán modeladas las entidades que definen la aplicación y las relaciones entre ellas.

b) Diseño de Bajo Nivel, donde serán definidas las diferentes formas de ver la información, la interfaz con el lector y las relaciones, utilizadas para definir la estructura navegacional.

El esquema final del modelo es el siguiente:

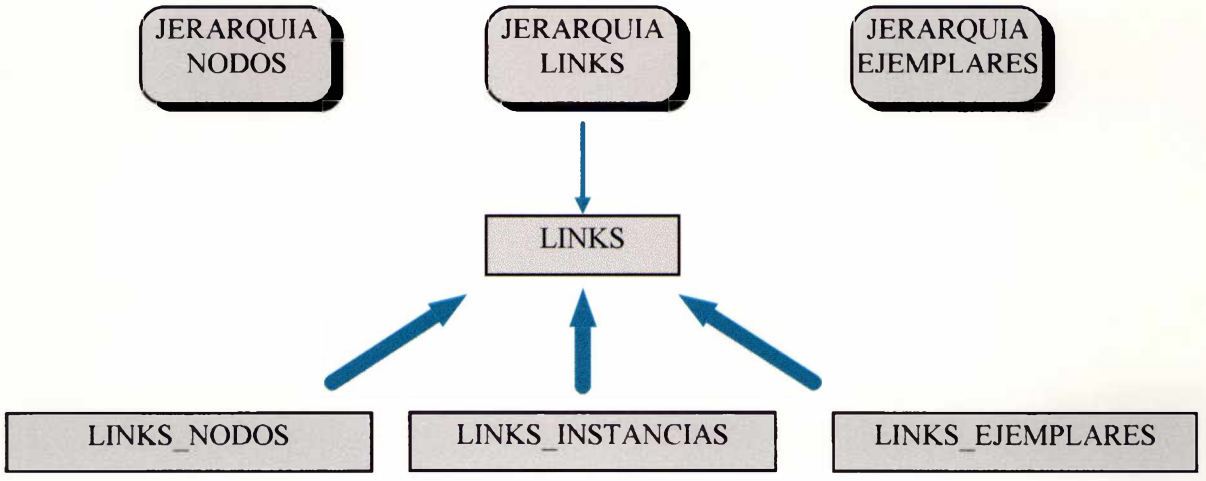


Figura 4.1

4.2.1.- Diseño de alto nivel

En esta etapa construiremos un modelo del dominio de la aplicación utilizando conceptos de la modelización orientada a objetos.

4.2.1.1.- Entidades

El concepto de entidad corresponde a un objeto del dominio de la aplicación y se describe por un conjunto de propiedades (atributos) y comportamiento (métodos). Cada atributo estará definido por un nombre y un dominio, donde el dominio puede ser primitivo, tales como : carácter, string, fecha, numérico, lógico, enumerativo, sonido, texto y video, o definido por el usuario.
Las entidades que compartan los mismos atributos e igual comportamiento serán agrupados en una clase (punto 1.2) , las que denominaremos en nuestro modelo clases **NODOS**.

- Atributos:

Los atributos que describen una clase nodo los clasificamos de la siguiente manera:

- ✓ Un ***atributo atómico*** es aquel instanciado con un solo valor proveniente de un dominio primitivo.
- ✓ Un ***atributo compuesto*** es aquel cuyo dominio es una clase no primitiva y establecen la semántica de la relación parte-de.
- ✓ Un ***atributo multidominio*** es aquel que puede ser representado desde diferentes dominios, por ejemplo un Comentario podría ser descripto en este nivel de diseño tanto

como un texto, como por sonido. Su utilización la veremos en el momento de definir la interfaz en el Diseño de Bajo Nivel.

- ✓ Un ***atributo derivado*** es aquel que se define mediante una función.
- ✓ Un ***atributo multivaluado*** es aquel que puede instanciarse con un conjunto de valores de su dominio, y son definidos mediante constructores.

El modelo presenta los **constructores** SET OF y LIST OF, que son utilizados para definir un atributo multivaluado como un conjunto o una lista de objetos de una clase definida y su sintaxis es la siguiente:

SET OF (C1): define un conjunto de objetos asociados a la clase C1.
LIST OF (C1) : define una lista de objetos asociada a la clase C1.

Veamos la definición de clases nodos basado en el ejemplo del Apendice B.

clase CONTINENTES

nombre	: string
titulo_fon	: sonido
descripción	: texto
descr_imag	: imagen
ubicación	: texto
superficie-enKm²	: numérico

clase ERAS

nombre	: ['Cenozoica', 'Paleozoica',...]
título_fon	: sonido
descripc	: texto
descrip_imag	: imagen
año_desde	: string
año_hasta	: string
períodos	: set of (PERIODOS)

Ejemplo 4.A

En este ejemplo vemos que el atributo "nombre" de la clase ERAS es definido como un enumerativo y el atributo "períodos" como un atributo compuesto

4.2.1.2.- Relaciones

El concepto de entidad definido sólo describe una parte del dominio de la aplicación, para completar esta descripción es necesario incorporar objetos que permitan relacionar las entidades, los que denominaremos relaciones. El modelo permite tener diferentes tipos de relaciones entre clases nodos, a través del concepto de subclase (relación Es-un), de atributos compuestos (relación Parte-de) y relaciones definidas por el usuario.

El concepto de subclase modeliza la semántica de la relación *Es-Un* entre clases, podemos decir que una subclase es una especialización de una o más clases existentes, llamadas superclases. Este modo de organización nos permite crear una jerarquía de clases nodos, que denominaremos **Jerarquía de nodos**, cuya raíz es una clase llamada NODOS que define el comportamiento común a todas sus subclases.
Los atributos compuestos definen relaciones implícitas entre nodos y definen [KIM95] la semántica de la relación *Parte-de*, que permite ver a una entidad compuesta como un todo o acceder a cada una de sus partes independientemente. A partir de estas relaciones entre nodos obtenemos una **Jerarquía de composición** de clases de nodos (ver ejemplo en el Apéndice B).
En nuestro modelo, a diferencia de algunos modelos de bases de datos los atributos complejos (punto 1.2) serán modelados como relaciones entre clases nodos, esto se debe a que en Hipermedia las relaciones son objetos de primera clase, al igual que las entidades, por lo tanto no deben oscurecerse definiendolas a través de un atributo complejo.

Las relaciones *definidas por el usuario,* en nuestro modelo se definen como clases, las que forman una jerarquía cuya raíz es la clase **LINKS**.
La clase LINKS tiene tres atributos fijos **From**, **To** y **Cardinalidad** . El dominio de los atributos From y To es cualquier clase existente, estos indican la clase fuente y destino del link respectivamente. En esta clase se define el comportamiento común a las relaciones definidas por el usuario.
El modelo permite representar relaciones con cardinalidad 1-1 o 1-n. Además, como los links estan definidos por medio de clases es posible tener información específica de la relación, incorporando nuevos atributos tales como una fecha, un dibujo, etc.

Veamos un ejemplo: supongamos que deseamos relacionar especies animales y el continente donde habitaron y además deseamos agregar información acerca del hallazgo de los restos fósiles del animal en el continente. Estos datos podrían ser representados por medio de la siguiente relación :

clase HABITABAN

superclase	: {LINKS}
from	: ESPECIES_ANIMALES
to	: CONTINENTES
cardinalidad	: 1-n
comentario	: multidominio(sonido, texto)

A continuación se presenta un esquema conceptual del CDROM Dinosaurs de Microsoft, mostrando algunas de las entidades y sus relaciones.
Esta aplicación es la utilizada para ejemplificar los conceptos definidos.

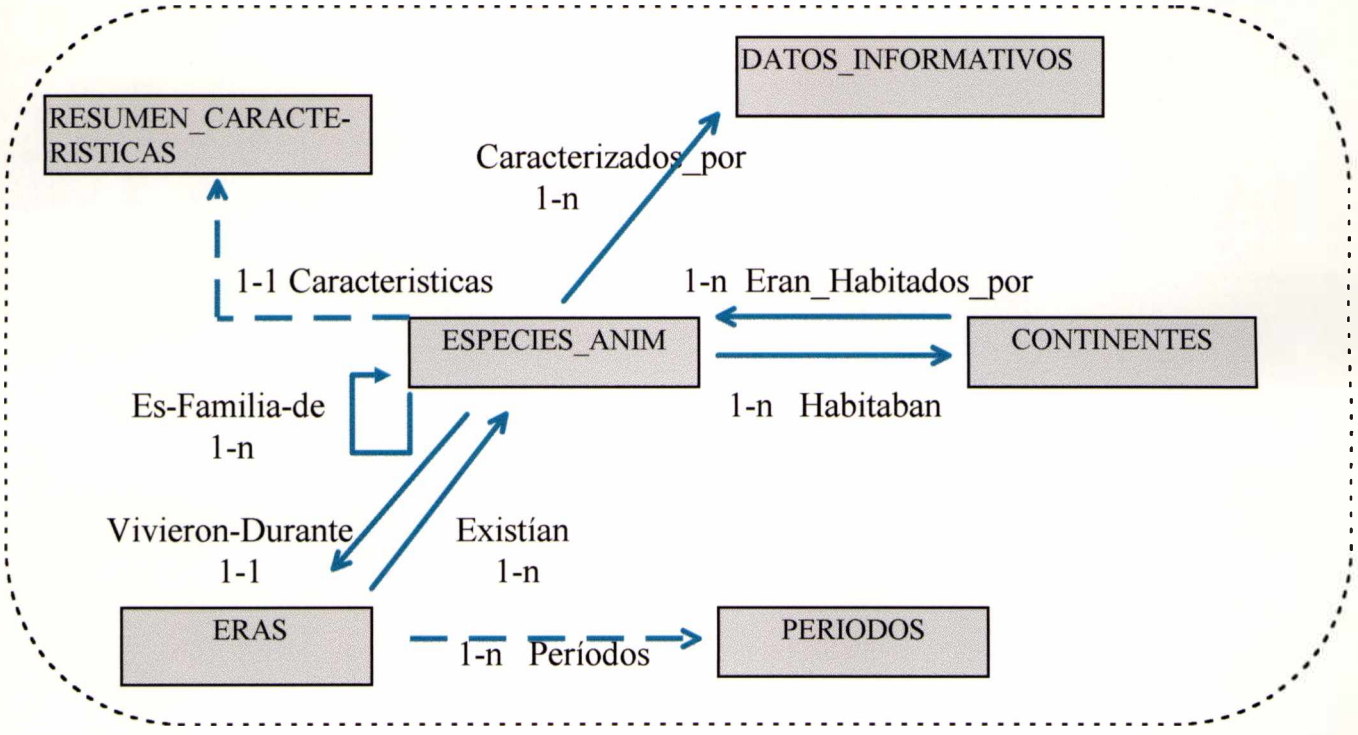


Figura 4.2.: Los rectángulos representan entidades y las aristas representan relaciones entre entidades. En las relaciones debe indicarse la cardinalidad.Las relaciones Parte-de se representan con linea punteada (------).

4.2.1.2.1.- Herencia

En nuestro modelo la jerarquía de clases nodos, como ya hemos visto, captura la relación Es_Un entre una clase y su/s superclase/s, por lo tanto, una clase **hereda** los atributos y métodos de su/s superclase/s.
El modelo implementa el concepto de herencia múltiple, lo que permite realizar un diseño del dominio más intuitivo.

Consideremos las definiciones de las clases nodos Especies_Animales, Dinosaurios, Reptiles e Iguanidos del apéndice B

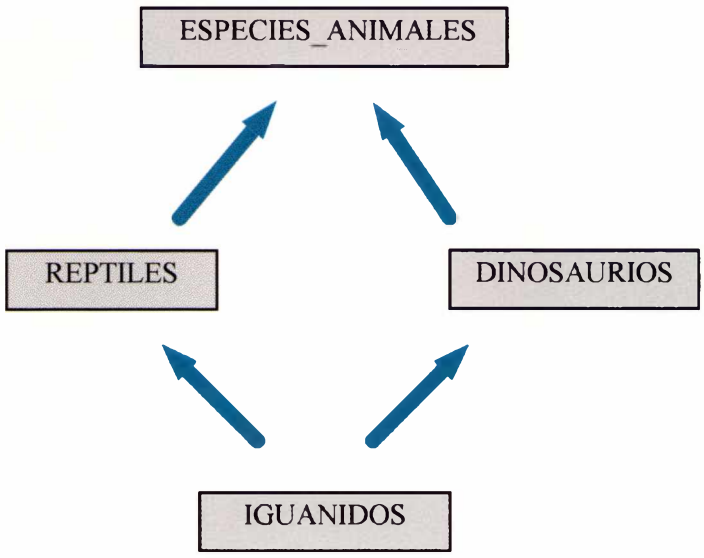


Figura 4.3

La clase IGUANIDOS hereda los atributos y métodos de la clase REPTILES y la clase DINOSAURIOS, pero surgen problemas ¿ Qué sucede con los atributos heredados más de una vez ?.Podríamos distinguir dos situaciones :
a) Cuando proviene de un ancestro común, por ejemplo, los atributos de la clase nodo ESPECIES_ANIMALES. Este caso lo denominamos Herencia Repetitiva y lo resolvemos tomando sólo una vez cada atributo, pues conceptualmente tienen el mismo significado. Por ejemplo, el atributo **nombre** de la clase nodo REPTILES tiene el mismo significado conceptual que el atributo nombre de la clase nodo DINOSAURIOS, y ambos fueron heredados de la clase nodo ESPECIES_ANIMALES.

b) Cuando el atributo en conflicto proviene de ancestros diferentes, nuestra solución es calificar ambas propiedades, poniendo como antecesor el nombre de la clase de la que proviene. Por ejemplo supongamos el atributo atributo_a de la clase DINOSAURIOS y REPTILES ambos con diferente semántica, entonces la clase IGUANIDOS quedaría de la siguiente manera:

clase IGUANIDOS
superclase: {REPTILES, DINOSAURIOS}
reptiles.atributo_a : numérico
dinosaurios.atributo_a : numérico

4.2.1.3. Resumen de sección.

Como resultado de este nivel obtenemos un esquema representando el dominio de una aplicación hipermedia. Los que son definidos por medio de una jerarquía de clases NODOS en la que se representan las entidades del dominio de la aplicación, una jerarquía de clases

LINKS en la que se representan las relaciones explícitas entre nodos, y una jerarquía de composición de clases en la que se representan la relación parte-de.
Cada una de las clases está definida por medio de un conjunto de atributos tipados cuyo dominio puede provenir de medios muy variados (tales como sonido, imágen, video, texto, etc.)

4.2.2.- Diseño de bajo nivel

Durante el diseño de alto nivel vimos conceptos concernientes a la modelización del dominio de la aplicación y durante el diseño de bajo nivel veremos conceptos que definan la navegación, permitinedo de esta manera diferenciar dos etapas que contienen distintos niveles de abstracción de la información.

Consideremos la siguiente situación del CD Microsoft Dinosaurs, donde es posible acceder a información del Tyranosaurus Rex desde una perspectiva general , como vemos en la Figura 4.5, o desde un aspecto óseo, como en la Figura 4.6.

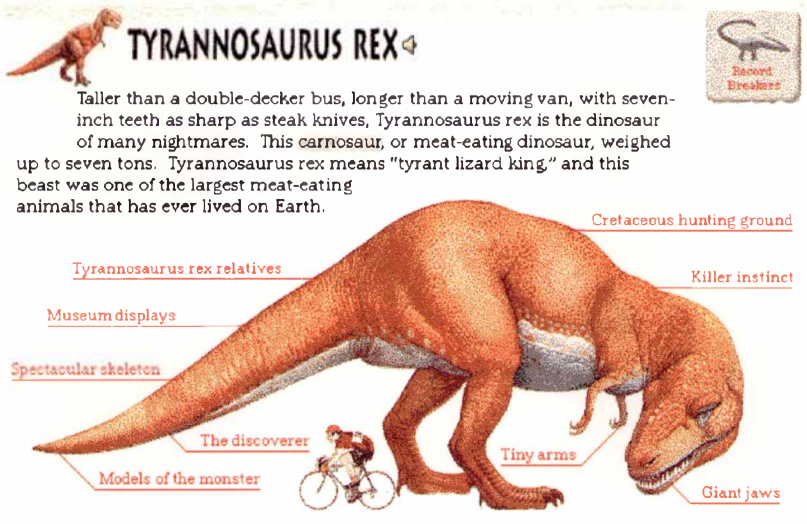


Figura 4.5

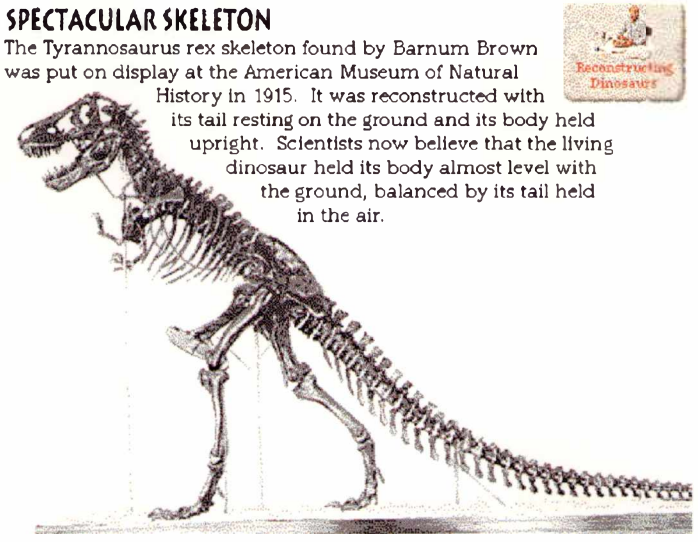
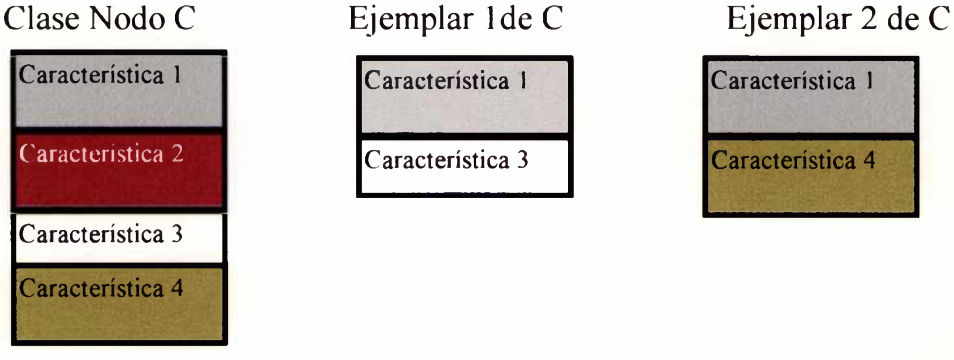


Figura 4.6

La información mostrada en ambas figuras caracterizan al Tyranosaurus Rex, por lo tanto es lógico pensar que en la modelización todas estas características esten contenidas en una sola clase. En nuestro modelo definiremos, en el diseño de alto nivel, una clase nodo con todas las características del Tyranosaurus Rex y en este nivel de diseño cada una de estas perspectivas (general y óseo) serán modeladas mediante ejemplares.

Esta situación podríamos esquematizarla en forma más general de la siguiente manera:



Podemos ver que un ejemplar define sólo aquellos atributos y relaciones de la clase nodo que son relevantes para una perspectiva determinada. La informació será mostrada a través del ejemplar por lo tanto, cada clase nodo tendrá al menos un ejemplar asociado.

4.2.2.1.- Ejemplares

Un ejemplar será definido en nuestro modelo como una clase que denominaremos clase ejemplar y agrupa los siguientes elementos:

- ✓ Atributos: los atributos de la clase nodo asociada que son relevantes para la vista que se esta diseñando. Los dominios de los atributos deberán ser iguales a los de la clase nodo o una especialización de ellos.
- ✓ Métodos: que definen el comportamiento de la clase y es seleccionado de la clase a la que esta asociada el ejemplar.
- ✓ Links: aquellos cuyo origen es la clase nodo que estamos considerando y que el diseñador considera que son necesarios para navegar desde este ejemplar. Serán incorporados a través de la palabra clave **ANCHORS** y definirán la estructura navegacional.

Las clases ejemplares están organizadas en uan jerarquía independiente a la de las clases nodos , denominada jerarquía de ejemplares.

El ejemplo antes presentado queda definido mediante las siguientes clases:

clase ESPECIES_ANIMALES
superclase {NODOS}
nombre : string
título_fon : sonido
descrip : texto
dibujo : imagen
esqueleto : imagen
descr_osea : texto

clase E_ESPECIES_ANIM
superclase{EJEMPLARES}
clase_asociada: ESPECIES_ANIMALES
nombre : string
título_fon : sonido
descrip : texto
dibujo : imagen
anchors (Habitaban, Vivieron_Durante, es_Familia_de)

clase ESPECIES_ANIM_OSEO
superclase{EJEMPLARES}
clase_asociada: ESPECIES_ANIMALES
nombre : string
título_fon : sonido
esqueleto : imagen
descr_osea : texto
anchors (Caractrizados_por)

- Atributos Multidominio

Como vimos durante el diseño de alto nivel, un atributo puede ser definido como multidominio. Estos atributos son utilizados cuando su dominio puede tener diferentes formas de representación y nos interesa mostrarlo desde una de ellas.
Al momento de definición del ejemplar se deberá seleccionar uno de los dominios, esto significa que sólo podrá ser consultado un dominio en cada ejemplar.

Por ejemplo, una descripción de un Reptil puede ser representada como texto o como sonido y en determinados casos mostrarlo como texto y en otros como sonido.
Consideremos la definición de la clase nodo REPTILES del apéndice B .

clase REPTILES
superclase {ESPECIES_ANIMALES}
época_vivió : string
alimentación : string
temperatura_sangre : string
descrip_rep : Multidominio(texto, sonido)

clase E_REPTILES_COMENT
superclase {EJEMPLARES}
clase_asociada:REPTILES
descr_rep : texto

El atributo descrip_rep que representa una descripción más detallada del reptil, es toma valores en los dos dominios definidos, texto y sonido, pero cada ejemplar mostrará la información desde uno de ellos.

4.2.2.2.- Relaciones entre ejemplares

El modelo permite relacionar ejemplares a través del concepto de subclase, representando la relación Es_Un, formando de esta manera una jerarquía de EJEMPLARES.
Cuando el diseñador esta modelando una aplicación pueden surgir situaciones donde es necesario crear relaciones entre ciertas vistas de una entidad, que no han sido reflejadas en las relaciones definidas en el Diseño de Alto Nivel. Este tipo de relaciones son representadas en nuestro modelo por medio de links entre ejemplares, los que amplían la estructura navegacional del ejemplar definido en el origen de la relación.
Es asi que la jerarquía de LINKS se redefine de manera de poder representar tantos los Links entre clases Nodos y entre clases Ejemplares.

La estructura queda modificada de la siguiente manera:

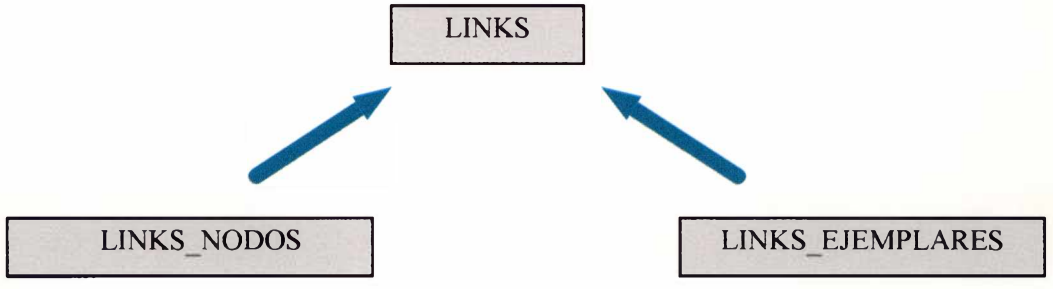


Figura 4.7.

Por ejemplo, supongamos que deseamos representar las variaciones geográficas de los continentes a través del paso de las eras. Esta situación excepcional está representada en nuestro modelo por una relación entre los ejemplares geográficos de Continentes y de Eras. Este tipo de relación sólo estará disponible si estamos consultando la información desde el punto de vista geográfico y se define de la siguiente manera:

clase VARIACION_GEOGRAFICA
superclase: {Links Ejemplares}
from: Continente Geografico
to: Eras Geografico
cardinalidad: 1-1

4.2.2.3.- Templates

Como hemos mencionado, es necesario contar con una interfaz que permita consultar/acceder a la información. Por tal motivo, es necesario asociar al ejemplar la representación visual del mismo, la que definiremos como *Templates*.
A la definición de clases ejemplares incorporamos el atributo fijo templates cuyo dominio será la clase template. Esta clase permite representar los atributos del ejemplar y todas sus relaciones, entre las que se incluyen los links incorporados mediante la palabra clave ANCHORS y los links que tengan como origen al ejemplar.

La figura 4.8 nos muestra un ejemplo de template para el ejemplar

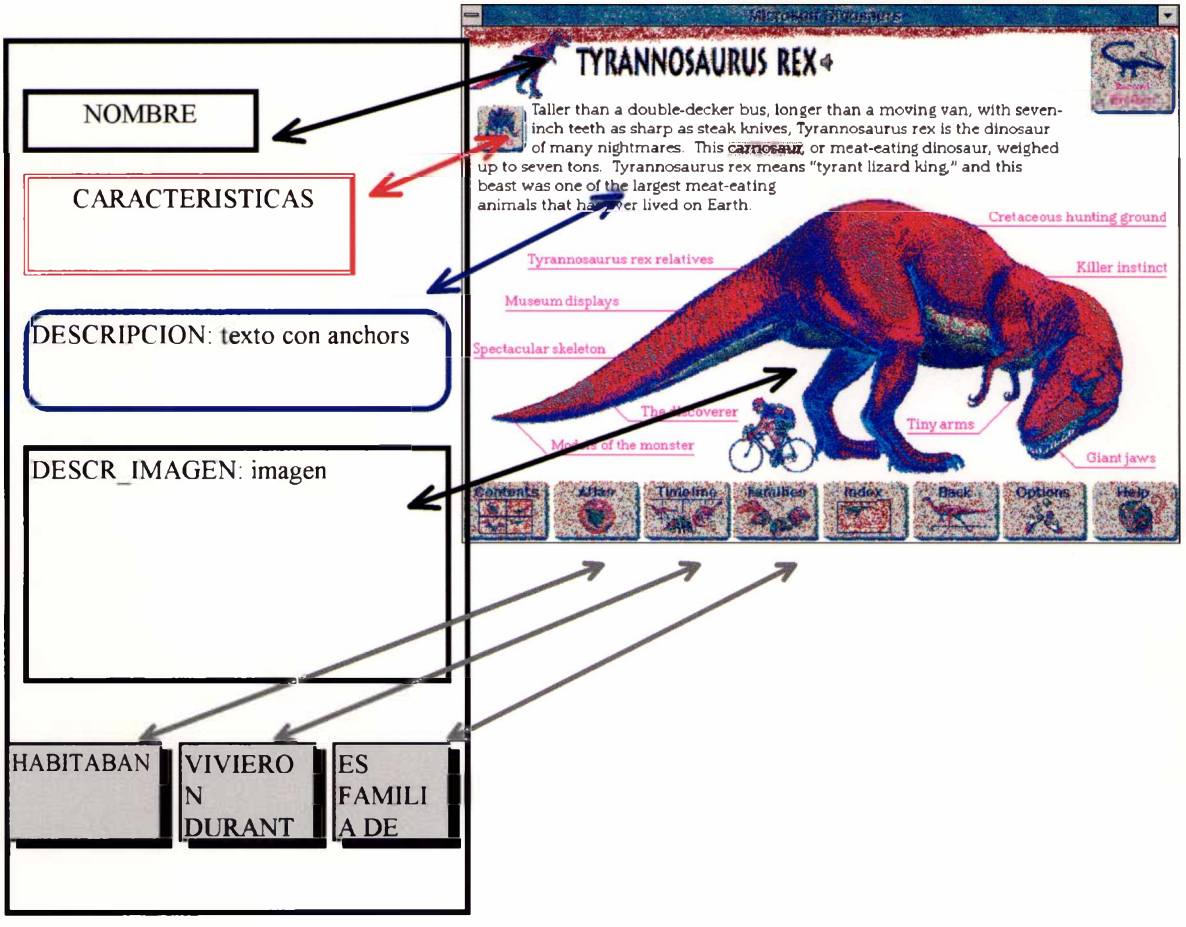


Figura 4.8.

4.2.2..4.- Estructuras de acceso

Las estructuras de acceso son elementos que complementan la navegación, pues permiten seleccionar uno de un conjunto de items (por ejemplo una lista de continentes, flias de animales, etc.) y actúan como índices o diccionarios.
Estas estructuras serán modeladas como clases nodos con la siguiente estructura interna predefinida, permitiéndole al diseñador sólo agregar nuevos atributos .

clase INDICES
nombre:
clase_destino:
selectores:
predicado:

El atributo **clase_destino** indica la clase de los elementos apuntados por el índice. El atributo **predicado** expresa que objetos serán accesibles y es expresado en términos de sus propiedades. El atributo **selectores** es por lo general un atributo del objeto destino y son organizados acorde a una estructura de datos predefinida (una lista ordenada, un conjunto de iconos, etc.).
Las estructuras de acceso tienen un comportamiento por defecto, que permite navegar desde el índice a cualquier elemento y viceversa.
Como ya hemos visto, por tratarse de una clase nodo debemos definir al menos un ejemplar que defina su apariencia y por ende el template que lo representará.
En Dinosaurs de Microsoft encontramos casos como este:

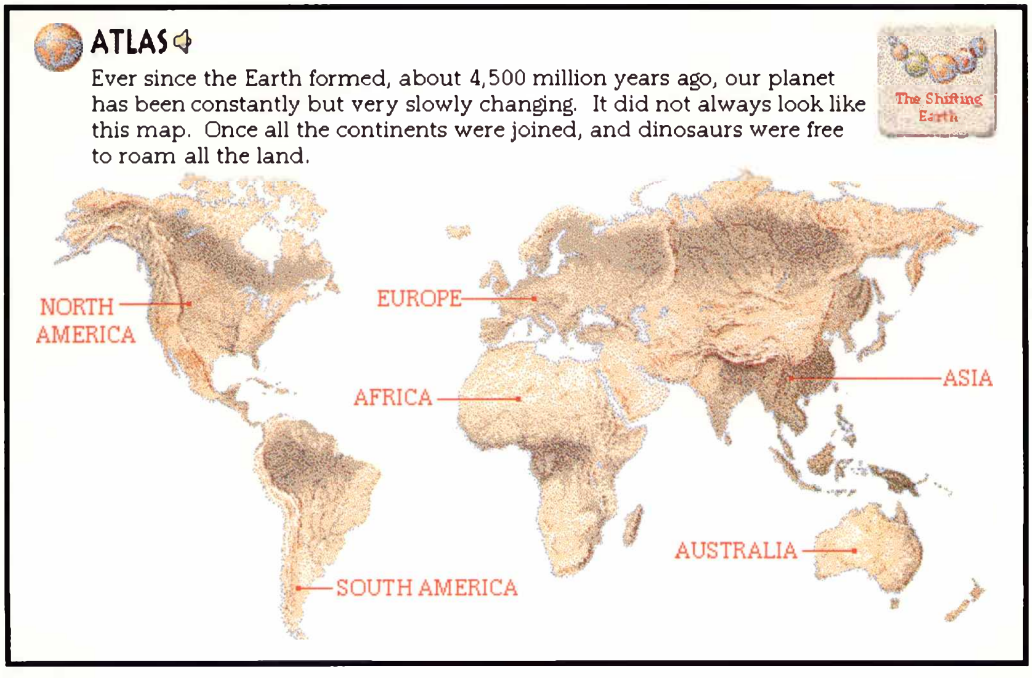


Figura 4.9.

que pueden ser representadas mediante una estructura de acceso que permita acceder al continente seleccionado, presionando en cualquiera de los nombres de los continentes. La estructura queda definida de la siguiente manera:

clase INDICE
nombre: INDICE_CONTINENTES
clase_destino: CONTINENTES
selectores: list of (nombre)
predicado: NULO

Como el predicado no es especificado el índice permitirá acceder a cualquier contienente instanciado de la clase CONTINENTES.

4.2.2.5. Resúmen de la sección

Como resultado de este nivel obtenemos una estructura navegacional dada por los diferentes ejemplares de una entidad y las relaciones definidas entre ellas, además de la representación visual del mismo.
Al esquema hasta ahora definido se incorpora una jerarquía de ejemplares y se especializa la jerarquia de links con las subclases Links Clases y links Ejemplares.

4.2.3.- Población del hipertexto

La construción de aplicaciones hipermedia en nuestro modelo consta de dos niveles, un diseño de alto nivel en el que se modelará los conceptos del dominio de la aplicación, definiendo una clase NODOS por cada entidad y una clase LINKS por cada relación entre dichas entidades y un diseño de bajo nivel en el que se definirán los diferentes puntos de vista, desde los que se podrá consultar cada clase nodo. Cada punto de vista será representado por una clase en la jerarquía de EJEMPLARES, y cada ejemplar tendrá asociado una interfaz con el usuario definida mediante un "template".

Hasta el momento contamos con el siguiente esquema de diseño:

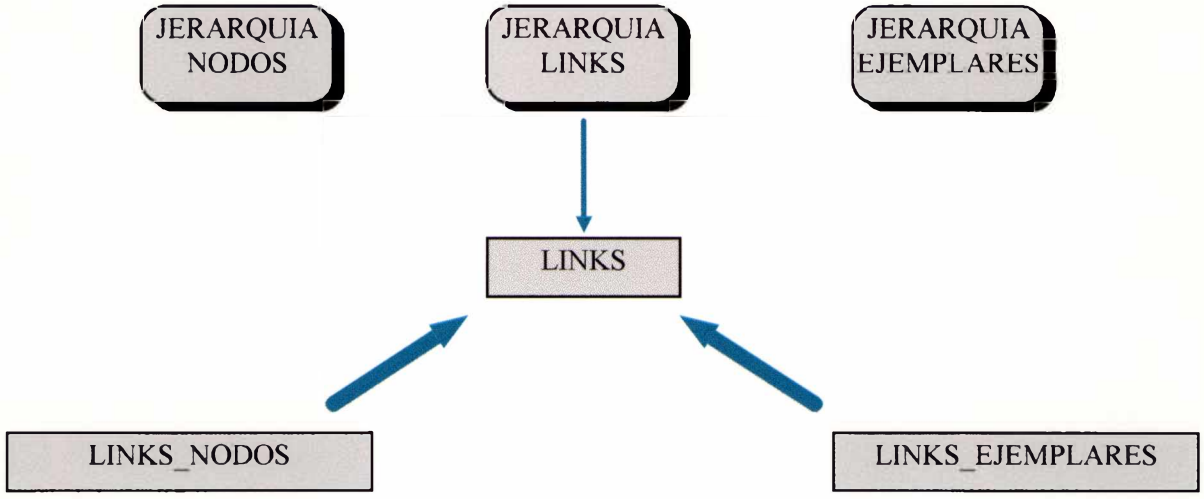


Figura 4.10.

La población del hipertexto consiste en la instanciación del esquema de la aplicación. Los objetos serán instanciados desde una clase nodo determinada, permitiendo de esta manera que tomen valor todos los atributos de la clase nodo y así poder consultarlo desde cualquier punto de vista (ejemplar).
Además recordemos que, si una clase está integrada por un atributo multidominio, este será instanciado con un valor en cada dominio definido.

- Relaciones entre instancias

En el momento en que el hipertexto es poblado podemos encontrarnos con situaciones excepcionales, donde resulta necesario crear relaciones que sólo serán válidas para dos instancias particulares. A tal fin, es posible definir este tipo de relaciones asociadas a la clase LINKS_INSTANCIA
Supongamos situaciones como la representada en la Figura 4.11,. donde el diseñador le provee al lector un camino que lo conduce al documento que él considera conveniente seguir navegando.

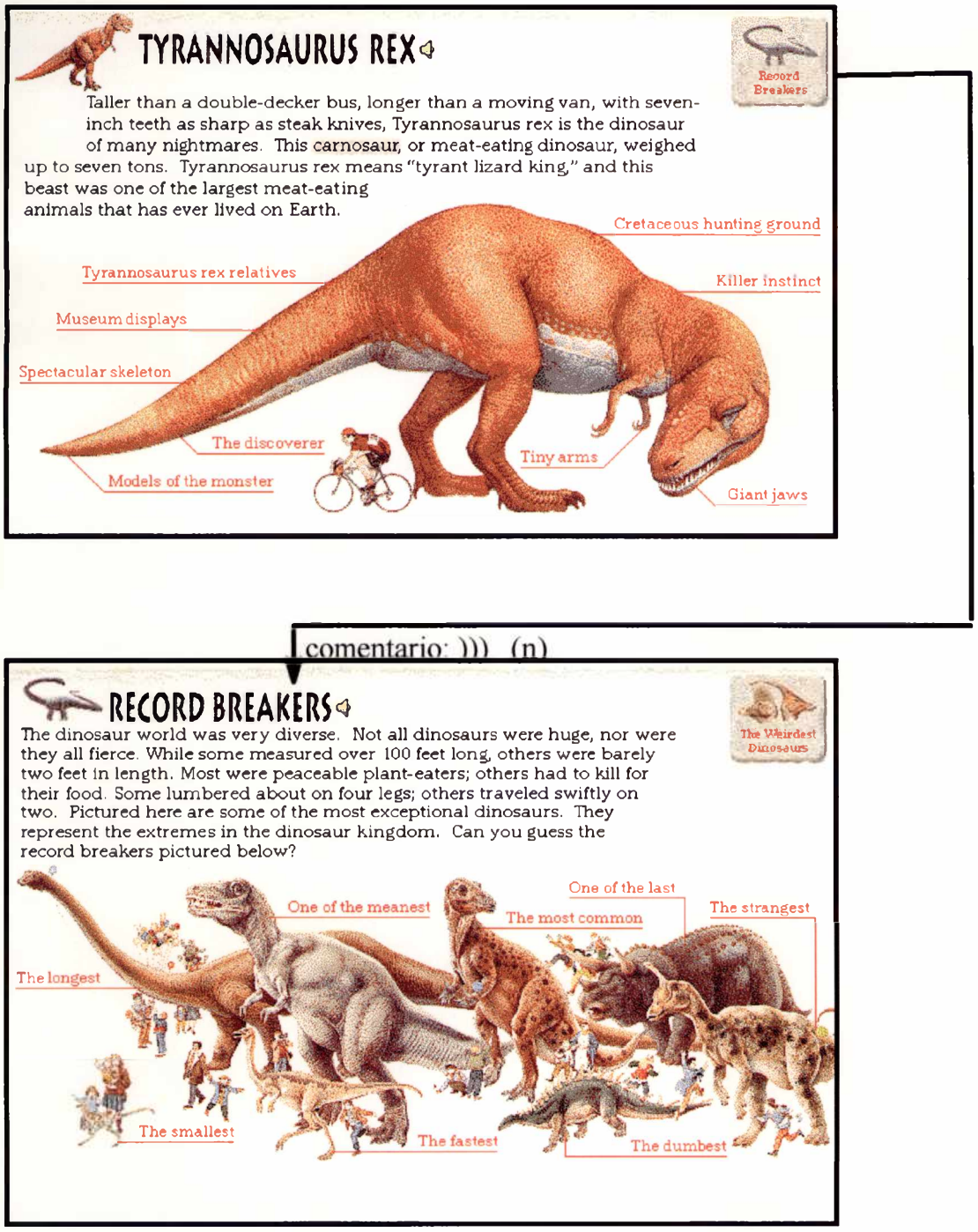


Figura 4.11

En este ejemplo cuando el lector "presiona" el botón que representa la relación, una breve descripción acerca del documento destino es comentada. En nuestro modelo estas relaciones son representadas por medio de links entre instancias, que continuando con el ejemplo anterior, se definirían de la siguiente manera:

clase LINKS_INSTANCIA
from : Instancia(Tyrannosaurus Rex)
to : Instancia(de la instancia Reacord Breakers de la clase Datos Informativos)
comentario : Sonido
cardinalidad : 1-1

4.3 - Consultas a hipertextos

4.3.1.- Introducción

El método primario para acceder a la información en hipertextos se resuelve mediante la navegación, esto no causará grandes dificultades en redes pequeñas y bien estructuradas.

Supongamos la siguiente situación, un lector desea encontrar en la red cierta información de su interés pero desconoce como llegar a ella. En este caso sería conveniente contar con una herramienta que nos facilite el acceso, por ejemplo mediante un **Lenguaje de Consultas** que permita localizar datos que cumplen con determinadas condiciones.
Además, durante el diseño de la aplicación hipermedia pueden surgir situaciones, donde resulte necesario consultar la estructura de la información relacionada con la definición de clases, atributos, etc., por lo que es deseable ampliar las capacidades del lenguaje de consultas, de manera de satisfacer estos requerimientos.

En este capítulo definiremos un **Lenguaje de Consultas** basado en el modelo anteriormente presentado. El mismo intenta combinar las características más destacables de las consultas de B.D.O.O. junto con la navegación provista por los hipertextos.

4.3.2.- Consultas a la Hiperbase

A través de las consultas a la Hiperbase el lector puede especificar la información que desea recuperar, permitiendo de esta manera acceder a los datos sin necesidad de navegar todo el hipertexto.
El objetivo de una consulta puede ser la proyección de valores de atributos, o la selección de algunos objetos de la hiperbase , obteniendo un espacio de trabajo más reducido, a partir del cual podremos navegar minimizando la desorientación.

Una consulta a la hiperbase tiene el siguiente formato:

SELECT <cláusula respuesta> **From** <cláusula de rango> [**In** <fuente>] [**Where** <predicado>]

cláusula respuesta: especifica los datos que serán recuperados por la consulta. Esta formada por una lista de nombres de clases C1,..,Cn o por una lista de nombres de atributos, precedidos por la clase a la que pertenecen, C1.a1,..,C1.am,..,Cn.a1,..Cn.am.

cláusula de rango: indica las clases involucradas en la consulta y esta formada por pares vi:Ci (variable objeto: nombre de clase), donde la variable objeto vi se utiliza para identificar a un conjunto de objetos de la clase Ci. Estas clases pertenecen a la **Jerarquía de Clases Nodos y Clases Links**.

fuente: indica el hipertexto fuente sobre el cual se formula la consulta. Es omitida de tratarse con la Red original.

predicado: permite restringir los datos a seleccionar. Si no fuera especificada ninguna condición, todas las instancias de las clases o atributos especificados en la cláusula respuesta serán seleccionados.

4.3.2.1- Predicados

Un predicado es una expresión que permite restringir los datos a seleccionar y se define de la siguiente manera:

predicado ::= <predicado> <conectivo lógico> <expresión> / <expresión> / Nulo.

expresión ::= <elemento-simple> <operador-simple> <elemento-simple> /
<elemento-multivaluado> <operador-multivaluado> <elemento-multivaluado/
<elemento-simple> <operador-simple-multiv> <elemento multivaluado> /
<elemento-multivaluado> <operador multiv-simple> <elemento-simple>.

conectivo-lógico ::= .Y. / .O. / .NO.

elemento-simple ::= nombre-atributo-simple / <cuantificador> (nombre-atributo-multivaluado) / constante simple.

elemento-multivaluado ::= nombre-atributo-multivaluado / constante multivaluada.

operador-simple ::= > / >= / < / <= / = / = = / ≠ .

operador-multivaluado ::= ⊂ / ⊄ / =

operador-multiv-simple ::= tiene-elemento / ¬ tiene-elemento.

operador-simple-multiv ::= ∈ / ∉ .

cuantificador ::= EXISTE / CADA.

La constante simple es una instancia de alguna de las clases existentes (un número, un string, una persona, etc,), y una constante multivaluada es una lista o un conjunto de instancias de clases definidas (lista de números, lista de continentes, etc).

Tratamiento de Cuantificadores

El modelo permite definir atributos multivaluados, definidos mediante los constructores SET_OF y LIST_OF. A este tipo de atributos podemos asociarle un **cuantificador universal** o **existencial** como los utilizados en el cálculo de predicados. Identificamos con la palabra "**CADA**" al cuantificador universal, y " **EXISTE**" al existencial. Estos cuantificadores deben preceder el nombre de un atributo multivaluado e indican que la comparación debe cumplirse para todos los elementos del conjunto (CADA), o al menos para alguno (EXISTE).

Ejemplo: supongamos que en la clase ESPECIES_ANIMALES del Apéndice B, agregamos un atributo llamado otros_nombres que representa otros nombres con los que se conoce cada especie animal.

clase ESPECIES_ANIMALES
superclase {NODOS}
nombre : string
título_fon : sonido
descrip : texto
dibujo : imagen
esqueleto : imagen
descr_osea : texto
caracteristicas : Resumen-Caracteristicas
otros_nombres: set of (String)

Si deseáramos obtener la especie animal que también es conocida con el nombre 'Saurios' podríamos expresarlo de las siguientes formas:

```
Qej1 = SELECT Especies_Animales
       FROM e:Especies_Animales
       WHERE {'Saurios'} ⊂ e:otros_nombres
```

o bien,

```
Qej2 = SELECT Especies_Animales
       FROM e:Especies_Animales
       WHERE EXISTE (e:otros_nombres) = 'Saurios'
```

Tratamiento de Operadores

El modelo permite utilizar los operadores tradicionales tales como > , < , >=, <=, <>, igual por valor (=) e igualdad idéntica (==). El tratamiento de las igualdades es similar al definido por [BER92].

igualdad idéntica : dos instancias de clases i e i' son idénticamente iguales si y solo si son el mismo objeto.

igualdad por valor :

i) dos instancias de clases primitivas son iguales por valor si y solo si tienen los mismos valores.

ii) dos instancias de clases compuestas c y c' son iguales por valor si y solo si se verifica que:

- ✓ c y c' tienen el mismo número de atributos
- ✓ Para todo Pi atributo de c , existe una propiedad Pj de c' tal que el nombre(Pi) = nombre(Pj) y se verifica que Pi es igual por valor a Pj.
  Si la estructura de Pi está formada por el constructor set-of, entonces c.Pi y c'.Pj tienen el mismo cardinal y para cada elemento del conjunto de c.Pi existe un elemento de c.Pj que son iguales por valor.
  Si la estructura de Pi está formada por el constructor list-of, entonces c.Pi y c'.Pj tienen la misma cantidad de elementos y para cada elemento k-ésimo, de c.Pi existe un elemento k-ésimo de c'.Pj que son iguales por valor .

Veamos un ejemplo de una consulta que involucra la igualdad por valor (=). Supongamos que deseamos consultar los nombre de los continentes cuya superficie sea mayor a 500.000 km2. Formulemos la siguiente consulta:

```
SELECT nombre
FROM c:Continentes
WHERE c:superficie-enKm2 = 500.000
```

A los operadores ya definidos agregamos el operador-simple **LIKE** para comparaciones entre strings:
El operador LIKE se utiliza siempre con el carácter especial '%', los que combinados permite comparar una cadena de caracteres con una subcadena de caracteres.

El ' % ' se utiliza como comodín para reemplazar la parte de la cadena de caracteres que desconocemos. Por ejemplo:
'%String', las palabras que finalicen con String.
'String%', las palabras que comiencen con String.
'%Strintg%' las palabras que contengan al String sin importar en que posición.

Supongamos que deseamos recuperar el nombre de los dinosaurios que finalicen en podos (que tienes patas)

```
SELECT Dinosaurios.nombre
 FROM  d:Dinosaurios
WHERE d:nombre LIKE  '%podos'
```

Las respuestas podrían ser : Sauropodos, Ornitopodos, etc. .

Cuando el nombre del atributo proyectado no presenta ambigüedades con respecto a su origen, es posible obviar el nombre de la clase a la que pertenece.

4.3.2.2 -Extensión del Predicado: Camino de Atributos:

Este modelo permite definir una clase como composición de otras clases y estas agregaciones pueden ser de varios niveles. Por lo tanto el lenguaje de consultas permite la especificación de predicados sobre una secuencia anidada de atributos de una clase, llamados caminos de atributos. Esto permite que no sea necesario introducir variables objetos adicionales en la cláusula de rango y en la especificación del predicado.

Por ejemplo: Si queremos consultar las especies animales que contengan características referidas a la alimentación, podríamos realizar la siguiente consulta

```
Q ej4 = SELECT Especies_Animales
        FROM e:Especies_Animales
        WHERE  e:características.resumen LIKE '%alimentación%'
```

Si queremos recuperar las eras que contengan a los períodos: Triásico y Jurásico, haremos la siguiente consulta:

```
Q ej5 =  SELECT Eras
         FROM r:Eras
         WHERE r:períodos.nombre ⊂ {'Triásico', 'Jurásico'}
```

Podemos distinguir dos tipos de caminos de atributos: simples y multivaluados. Un camino de atributos simple es aquel cuya secuencia de atributos son todos simples. Se utilizarán operadores simples para comparar caminos simples. Un camino de atributos multivaluado tiene al menos un atributo multivaluado, por lo tanto se podrán utilizar los operadores y cuantificadores que definimos para dichos atributos.

Por ejemplo: Si deseamos recuperar las Eras que tengan al menos un período que comience después de 10000 millones de años a.C.

Q ej6 = **SELECT** Eras
FROM r:Eras
WHERE EXISTE (r:periodos.AñoDesde) > '10000 aC'

Podemos entonces extender la definición inicial del predicado incorporando los caminos de atributos de esta forma:

predicado ::= <predicado> <conectivo lógico> <expresión> / <expresión> / Nulo.
expresión ::= <variable-simple> <operador-simple> <variable-simple> /
<variable-multivaluado> <operador-multivaluado> <variable-multivaluado/
<variable-simple> <operador-simple-multiv> <variable-multivaluado> /
<variable-multivaluado> <operador multiv-simple> <variable-simple>.
conectivo-lógico ::= .Y. / .O. / .NO.
variable-simple ::= variable-objeto.camino-simple / camino-simple.
camino-simple ::= elemento-simple / elemento-simple.camino-simple
elemento-simple ::= nombre-atributo-simple / <cuantificador> (nombre-atributo-multivaluado) / constante simple.
variable-multivaluada ::= variable-objeto.camino-multivaluado / camino-multivaluado.
camino-multivaluado ::= elemento-multivaluado / elemento-multivaluado.camino-multivaluado.
elemento-multivaluado ::= nombre-atributo-multivaluado* / elemento-simple / constante multivaluada.

(*) En un camino de atributos multivaluado, por lo menos uno de los elementos del camino debe ser un atributo multivaluado.

4.3.2.3 Funciones

Definimos la siguiente función con el objetivo de facilitar la escritura de los predicados.

✓ Es-Relación:

La función Es-Relación permite conocer la existencia de una relación R entre dos objetos. Esta puede sustituir cualquier parte del predicado donde se espere un valor lógico. Su formato es el siguiente: Es-Relación (Nombre-Link, objeto-fuente, objeto-destino). Retorna el valor verdadero si existe una instancia **r** de la relación Nombre-Link donde r:from es el objeto-fuente y r:to es el objeto-destino.

Este tipo de información es posible obtenerla consultando la clase LINKS, pero resulta más natural utilizando esta función.

Supongamos que deseamos seleccionar los datos informativos de las especies animales que vivieron durante la era Cenozoica. Sin contar con esta función podríamos realizarla de la siguiente manera:

```
Q ej3 = SELECT Datos_Informativos
                FROM d:Datos_Informativos, c:Caracterizados_Por,
                     v:Vivieron_Durante
                WHERE c:from       == v:from
                   .Y.   v:to.nombre = 'Cenozoica'
```

Con la función será:

```
Q ej3 = SELECT Datos_Informativos
                FROM d:Datos_Informativos, r:Eras, e:Especies_Animales
                WHERE  Es-Relacion('Caracterizados_Por',:e,:d)
                    .Y.   Es-Relacion('Vivieron_Durante',:e,:r)
                    .Y.   r:nombre  = 'Cenozoica'
```

4.3.2.4 Consultas Recursivas

Incorporamos consultas recursivas porque, como hemos visto en la definición del esquema de Base de Datos, podemos definir jerarquías de clases conteniendo ciclos. Dentro de la estructura de clases podemos encontrar distintos tipos de ciclos:

✓ Ciclos Directos :

Encontramos un ciclo directo cuando un atributo p de la clase C tiene el dominio definido en C o en cualquiera de sus superclases o subclases. Como vemos en las Figuras 4.12 y 4.13.

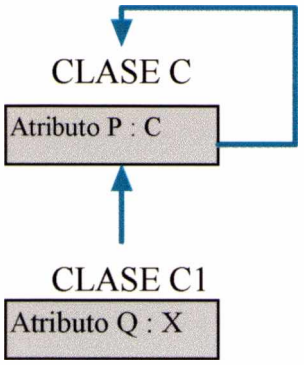


Figura 4.12

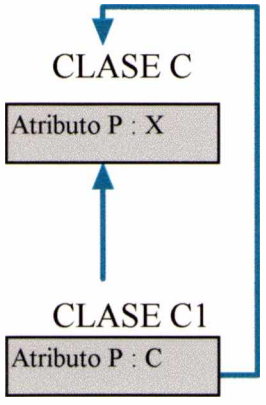


Figura 4.13

Ciclos Indirectos

Encontramos un ciclo indirecto cuando un atributo p de la clase C tiene dominio en una clase Ci, y ésta tiene definido un atributo pi, cuyo dominio es la clase C, o alguna superclase o subclase de ella. Donde i es mayor o igual 1. Las figuras 4.14 y 4.15 representan este tipo de ciclos.

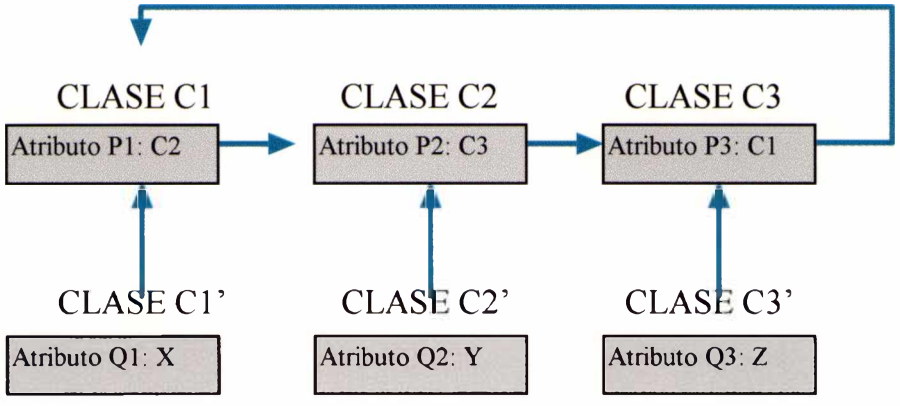


Figura 4.14

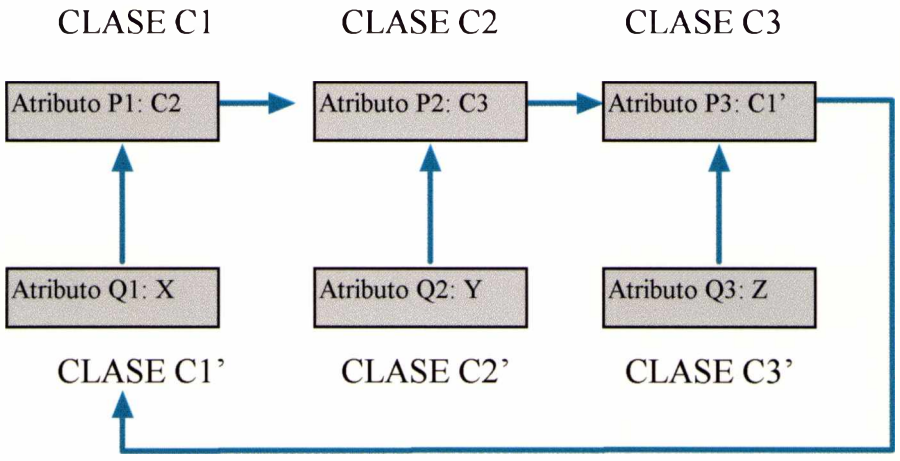


Figura 4.12

Introducimos a R como el operador de recursión, para especificar condiciones en consultas que involucren ciclos directos e indirectos.
Dado una instancia i, de una clase C y sea p un atributo de C, definimos R(i.p) a un conjunto de instancias que son valores del atributo p de C, o son valores del atributo p de alguna instancia i' , la cual es un valor de la propiedad p de O, etc.
Si la recursión se realiza sobre un ciclo directo p será el nombre de un atributo, si la recursión se realiza sobre un ciclo indirecto, p será un camino de atributos.

A fines de ejemplificar ampliamos la definición de las clases Especies-Animales, Dinosaurios y Continentes.

clase ESPECIES_ANIMALES
nombre : String
...
descendiente : ESPECIES_ANIMALES.

clase DINOSAURIOS
...
hallados-en : CONTINENTE.

clase CONTINENTE
...
habitan-en-actualidad: ESPECIES_ANIMALES

Por ejemplo, si queremos recuperar todos las especies animales que tienen como descendiente al Tyranosaurio Rex.

```
SELECT Especies_Animales
FROM e: Especies_Animales , e': Especies_Animales
WHERE  e':nombre = 'Tyranosaurio Rex' .Y. e': ∈ R(e:descendientes)
```

En este ejemplo estamos en presencia de un ciclo simple, porque el atributo descendientes de la clase ESPECIES_ANIMALES, es recursiva en sí misma.
Supongamos que deseamos hallar las especies animales que habitan en los continentes donde fue hallado los restos del Tyranosaurio Rex.

SELECT Especies_Animales
FROM e:Especies_Animales, d:Dinosaurios
WHERE d:nombre = 'Tyranosaurio Rex' .Y. e: ⊂ R(d:hallados-en.habitan-en-actualidad)

en este ejemplo la recursión se realiza sobre un ciclo indirecto.

4.3.2.5.- Resultado de Consultas:

Para explicar el resultado de las consultas recordemos el formato que hemos definido:

[**Resp**] = **SELECT** <cláusula respuesta> **From** <cláusula de rango> [**In** <fuente>] [**Where** <predicado>]

Si en la cláusula respuesta se especifican los nombres de clases C1,..,Cn, la consulta es una operación de Selección sobre esas clases del hipertexto fuente. Esta consulta recupera las instancias de las clases C1,..,Cn, que satisfacen una combinación booleana de predicados.

Si en la cláusula respuesta se especifican los nombres de atributos C1.a1,..,Cn.am, la consulta es una proyección de los atributos especificados de las clases C1..Cn del hipertexto fuente, en las instancias que satisfacen el predicado.

Podemos clasificar las respuestas de las consultas según la **cláusula respuesta**. Si la cláusula respuesta está formada por los atributos de una o más clases (similar a la operación de Proyección en el modelo Relacional), se generará una lista, donde cada elemento de esta estructura es una lista conformada por las instancias de los atributos proyectados.
Si alguno de los atributos proyectado es multivaluado, se generarán listas anidadas.

LISTA
RESPUESTA

(C1.a1,..,C1.am,.C2.a1,(C2.a21,.C2.a22,.,C2.a2k))
(C1.a'1,..,C1.a'm,.C2.a'1,(C2.a'21,.C2.a'22,.,C2.a'2k))
....
....

En este caso el atributo proyectado a2 de la clase C2 es multivaluado.

A fines de ejemplificar ampliamos la definición de la clase CONTINENTE, agregándole el atributo Países, el cual contendrá un conjunto con los nombres de los países que lo componen.
Veamos el siguiente ejemplo: supongamos que deseamos seleccionar los nombres de las especies-animales y los países que en la actualidad componen los continentes donde ellos habitaron

clase ESPECIES_ANIMALES

superclase {NODOS}
nombre : string
título_fon : sonido
descrip : texto
dibujo : imagen
esqueleto : imagen
descr_osea : texto
características : Resumen-Caracteristicas
otros_nombres: set of (String)

clase CONTINENTES

nombre : string
titulo_fon : sonido
descripción : texto
descr_imag : imagen
ubicación : texto
superficie-enKm²: numérico

Q ej7 = SELECT ESPECIES_ANIMALES.nombre, países
FROM a:Especies-Animales , c:Continentes
WHERE Es-Relación('Habitaban', a:, c:)

ESTRUCTURA DE ACCESO Q ej 7

('Tyranosaurus Rex', ('Argentina',...,...))
....
('Albertosaurus', ('Italia',...,...))

En cambio, si la cláusula respuesta está formada por Nombres de Clases (similar a la operación de Selección en el Modelo Relacional), la respuesta será un hipertexto. Este podrá ser nuevamente consultado (como fuente de la nueva consulta) , reduciendo de ésta manera el espacio de búsqueda. Ese hipertexto respuesta está formado por las instancias de las clases especificadas (en la cláusula respuesta) que satisfagan el predicado y por las relaciones implícitas existentes entre dichas instancias. Si alguna de las clases involucradas en la respuesta es compuesta, incluiremos en el hipertexto resultado toda su composición (su grafo de agregación).

El modo de acceder a las respuestas es mediante una estructura de acceso formada por las instancias de las clases involucradas, respetando el orden de aparición en la cláusula respuesta.

4.3.3.- Consultas a la estructura

Como vimos en los puntos 4.1 y 4.2 el modelo presenta tres jerarquías llamadas, NODOS, EJEMPLARES y LINKS y es el diseñador / autor la persona encargada de actualizarlas. Por tal motivo consideramos que además de las herramientas que permiten modificar el esquema, es necesario disponer de consultas a la estructura interna de una clase, una determinada jerarquía, etc.
Cuando el lector desea realizar una consulta, y no conoce los nombre de los atributos involucrados, podría necesitar de este tipo de operaciones.

Las consultas que a continuación detallaremos responderán al siguiente formato:

i) nombre_consulta **FROM** clase_C1 [,clase_C2,..,clase_Cn] [**OF** id_jerarquía]

donde:
nombre_consulta: es un nombre definido por el sistema que determina el tipo de consulta a realizar. Las mismas serán identificadas en el texto con letra mayúscula-imprenta. Los posibles nombres son JERARQUIA, DET-JERARQUIA, ESTRUCTURA, EJEMPLARES.

clase_Ci: es el nombre de la clase que se desea consultar

id_jerarquía: es utilizada para identificar la jerarquía a la que están asociadas la/s clase/s clase_C1, ...,clase_Cn. Los valores posibles son:

- ND: Jerarquía de clases Nodos
- EJ: Jerarquía de clases Ejemplares
- LK: Jerarquía de clases Links.

Todas las consultas serán realizadas sobre el hipertexto original.

A continuación detallaremos las diferentes consultas y sus funciones. Los ejemplos utilizados están basados en las definiciones del Apéndice B.

✓ Consultas a las jerarquías

Esta consulta está orientada al diseñador ya que le permite conocer como está formada la jerarquía encabezada por una clase particular. Esto puede resultarle de mucha utilidad, por ejemplo, al momento de crear una clase le permitiría conocer las clases existentes y sus relaciones de herencia.
La misma puede estar dirigida a cualquier clase (sólo una) de cualquier jerarquía y muestra como respuesta el primer nivel de subclases, si es elegida la opción con detalle (DET-JERARQUIA) serán mostrados todos los niveles de subclases.
Su formato:

JERARQUÍA FROM clase_A **OF** id_jerarquía: muestra el primer nivel de subclases de la clase_A

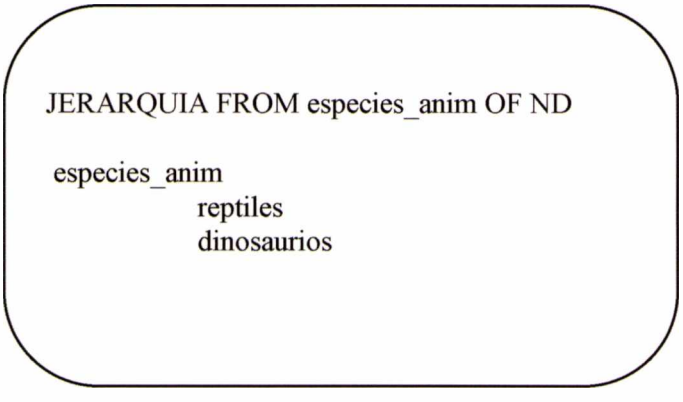
DET_JERARQUIA FROM clase_A **OF** id_jerarquía: muestra todos los niveles de subclases de la clase_A

clase_A: puede ser cualquier clase existente en la jerarquía especificada por id_jerarq.
id_jerarquía: puede ser cualquiera de los identificadores de jerarquías especificados anteriormente (ND, EJ, LK).

Ejemplos:
- Basándonos en la jerarquía especificada en la figura C.1, supongamos que el diseñador desea conocer las clases creadas como subclases de especies animales.

JERARQUIA FROM especies_anim **OF** ND

y la respuesta obtenida sería la siguiente



- Basándonos en la jerarquía de la figura C.2, supongamos que el diseñador desea conocer los ejemplares existentes que son subclase de e_especies_anim. La consulta sería la siguiente:

DET_JERARQUIA FROM e_especies_anim OF EJ

y obtendríamos la siguiente respuesta:

DET_JERARQUIA FROM e_especies_anim OF EJ

e_especies_anim
- e_reptiles
  - e_reptiles_mar
  - e_reptiles_aire
- e_dinosaurios
  - e_ornithischians
    - e_ceraptosians
    - e_ankylosaurus
    - e_ornithopods
    - e_stegosaurus
    - e_pachycephalosaurus
  - e_saurichians
    - e_small_theoropods
    - e_large_theoropods
    - e_pro_sauropods

✓ Consultas a la Estructura de las clases.

Las consultas a la estructura de una clase nos permiten ver los nombres de los atributos que la definen , sus dominios, y los métodos asociados. Si la clase en cuestión está asociada a la jerarquía de ejemplares, entonces también serán mostrados los anchors definidos en ellas.
Esta consulta es de utilidad tanto para el lector como para el diseñador. Por ejemplo, supongamos que el lector desea consultar ciertos atributos de una clase y no recuerda con exactitud sus nombres o el caso del diseñador al momento de crear una clase y desea asegurarse que la clase elegida como superclase realmente lo sea.
Su formato:

ESTRUCTURA FROM clase_A OF id_jerarquía

clase_A: podría ser cualquier clase existente en la jerarquía especificada en id_jerarquía.

id_jerarquía: puede ser cualquiera de los identificadores de jerarquías especificados anteriormente : ND - Jerarquía de Nodos, EJ - Jerarquía de ejemplares o LK - Jerarquía de Links.

Ejemplos:
- Supongamos que el lector desea una consulta sobre ciertos atributos de ERAS y no recuerda su estructura , entonces la consulta a realizar sería la siguiente:

ESTRUCTURA FROM eras OF ND

y obtendríamos la siguiente respuesta

ESTRUCTURA FROM eras OF ND

clase ERAS
nombre : String
título_fon : Sonido
descripción : Texto
descrip_imag: Imagen
año_desde : String
año_hasta : String
periodos : set of (periodos)

métodos: .....

- Supongamos que el diseñador desea crear un subejemplar de ESPECIES_ANIM_OSEO y desea conocer exactamente qué atributos se heredarán, entonces debería realizar la siguiente consulta:

ESTRUCTURA FROM especies_anim_oseo OF EJ

y obtendría la siguiente respuesta:

ESTRUCTURA FROM especies_anim_oseo OF EJ

clase ESPECIES_ANIM_OSEO
clase_asoc : ESPECIES_ANIM
nombre : Texto
título_fon : Sonido
esqueleto : Imagen
descr_osea : Texto
características : RESUMEN_CARACTERISTICAS

métodos:

anchors: (Habitaban, Vivieron-Durante, Es-Familia-De)

✓ Ejemplares

Esta consulta está orientada sólo al diseñador, pues la misma permite conocer el nombre de todos los ejemplares asociados a la/s clase/s nodos a la / s que está dirigida la consulta. El formato es el siguiente:

EJEMPLARES FROM Clase1 [,Clase2,..,ClaseN]

clase_I: podría ser cualquier clase existente en la jerarquía de nodos (ND).
id_jerarq: la identificación de jerarquía está restringida sólo a ND, pues los ejemplares están asociados a clases nodos.

Ejemplo:
- Supongamos que el diseñador desea conocer qué ejemplares han sido creados para las clases ESPECIES_ANIM y DATOS_INFORMATIVOS, entonces debería realizar la siguiente consulta:

EJEMPLARES FROM especies_anim, datos_informativos OF ND

y obtendríamos la siguiente respuesta:

EJEMPLARES OF especies_anim,
datos_informativos OF ND

Ejemplar	Clase Asociada
e_especies_anim	: ESPECIES_ANIM
especies_anim_oseo	: ESPECIES_ANIM
e_datos_informativos	: DATOS_INFORMATIVOS
artículos_relacionados	: DATOS_INFORMATIVOS

4.3.3.1. Funciones

A continuación presentamos una función que resulta de utilidad al momento de realizar consultas a la estructura de la información y puede ser utilizada tanto en la cláusula respuesta, como en el predicado.

✓ Dominio

Posee el siguiente formato: Dominio (Nombre de clase-.nombre-atributo). La función Dominio retorna un string cuyo valor es el nombre de la clase primitiva o compleja, que es dominio del atributo nombre-atributo perteneciente a la clase Nombre-Clase.
Supongamos que deseamos conocer las relaciones existentes entre Especies-Animales y Eras.

```
Q ej3 = SELECT Links
            WHERE Dominio(Link.from) = 'Especies-Animales' .Y.
            Dominio(Link.to) = 'Eras'.
```

4.4.- Arquitectura del sistema

El modelo definido es un sistema hipermedia orientado a objetos que le permite al autor/diseñador crear su propia red hipermedia.

Cuando un diseñador comienza a crear su propio hipertexto, el mismo comienza definiendo sus tipos, los que serán representados como clases NODOS (4.2.1.1). El modelo exige crear al menos un ejemplar que defina la interfaz de cada nodo (4.2.2), estos ejemplares serán representados como clases EJEMPLARES. Y por último, el diseñador estará interesado en relacionar sus nodos, definidas por medio clases LINKS (4.2.1.2, 4.2.2.2, 4.2.3).
Las clases mencionadas están organizadas en jerarquías de nodos ejemplares y links y la semántica asociada a cada una es la relación Es-Un entre una clase y sus superclases, lo que lleva implícito el concepto de herencia.

Todas las clases anteriormente mencionadas serán definidas por un conjunto de atributos tipados, cuyo dominio podrá provenir de diferentes medios, por ejemplo, texto, video, sonido, etc. Además como vimos es posible definir atributos multidominio, los que serán representados por medio de clases cuyo comportamiento es similar al definido en 4.2.1.
Diferentes tipos de atributos tales como los multivaluados, simples, compuestos y complejos tendrán comportamiento similar al definido en B.D.O.O. tradicionales, lo mismo sucede con los Valores de Clases del punto 4.2.1.2.2.

Como vimos en 4.2.2.3 cada ejemplar tiene asociado un template (su interfaz) el que será definido como un atributo más dentro de la clase ejemplar definida y cuyo dominio es la clase Template.

Por lo mencionado anteriormente estamos en condiciones de asegurar que el modelo puede ser bien soportado por una base de datos orientada a objetos tradicional que permita herencia múltiple, con los cambios necesarios para adecuarse a nuestro modelo de datos.

4.5.- Modificación del esquema

Como vimos en el punto 4.4 este sistema hipermedia orientado a objetos está soportado por una Base de Datos orientada a objetos, es decir, todos los cambios (creación, modificación, borrado) son similares a los realizados en una base de datos orientada a objetos convencional considerando los cambios necesarios para que se ajuste al modelo de datos definido.

Como en Manejadores de Bases de Datos Orientadas a Objetos tradicionales nuestro modelo debe incluir un sublenguaje de definición de datos (DDL), un sublenguaje de consulta y manipulación de datos (DML) y un sublenguaje de control de datos (DCL), los que deben preservar las restricciones y flexibilidades del modelo de datos presentado anteriormente.

4.5.1.- Lenguaje de diseño de datos (DDL)

Como el diseñador es el encargado de crear los tipos (clases) necesarios para modelar su sistema hipermedia, el mismo debe contar con un lenguaje de diseño de datos (DDL) que le permita realizar operaciones de modificación del esquema. Estas operaciones son:

1) Creación de una clase
2) Cambios al contenido de una clase
3) Cambios en la organización de la jerarquías de NODOS, EJEMPLARES y LINKS

Como el modelo es soportado por B.D.O.O. la comunicación con los objetos será por medio de mensajes implementados a través de métodos.
Un mensaje tendrá el siguiente formato:

Id-Jerarquía Operación Receptor [Argumentos]

donde, Id-Jerarquía identifica la jerarquía a la que estará asociada la clase receptora, los posibles valores son ND, EJ y LK.
Operación : es el nombre del mensaje.
Receptor :es el nombre de la clase a la que está dirigido el mensaje .
Argumentos : es opcional.

La taxonomía de cambios del esquema y especificación de su semántica son versiones adaptadas de las enunciadas por [KIM95]. Por esta razón, solamente especificaremos con más detalle los cambios introducidos para adaptarlo a nuestro modelo.

1) Creación de una clase.

El lenguaje definido en nuestro modelo provee de las siguientes instrucciones para permitir crear un a clase nodo, una clase ejemplar o una clase link.
La sintaxis es la siguiente:

Id_Jerarquía **CREAR_CLASE** nombre_clase **[:superclases ListaSuperClases]**
:atributos ListaAtributos
[:métodos ListaMétodos]

Id_Jerarquía: Identifica la jerarquía en la que será agregada la nueva clase. Los valores posibles son:

ND: si es una clase nodo,
EJ: si es un ejemplar,
LK: si es una relación

nombre_clase: es el nombre de la nueva clase

ListaSuperclases: Este argumento es optativo y permite especificar la lista de superclases de la nueva clase. En caso de no especificarse se asumirá la clase definida por el sistema como tope de la jerarquía.

ListaAtributos: es una lista de especificaciones de atributos. Una especificación de atributo tiene el siguiente formato:

```
Nombre-Atributo   : dominio          ClaseNodo
                  [: heredado-de     Superclase]
```

Nombre-Atributo es el nombre del atributo que se esta definiendo.

Supongamos que deseamos crear la clase NODO que define las eras (Apéndice B), el mensaje sería el siguiente:

```
ND CREAR_CLASE eras
        :atributos (  nombre        :dominio String
                      título_fon    :dominio Sonido
                      descr_imagen  :dominio Imagen
                      descripción   :dominio Texto
                      año_desde     :dominio String
                      año_hasta     :dominio String
                      periodos      :dominio set of (Periodos))
```

Como este mensaje no especifica el argumento :superclases el sistema asumirá la clase creada como subclase de la clase NODOS (tope de la jerarquía).

2) Cambios al contenido de una clase.

Los cambios a la definición de una clase incluye cambios a los atributos y métodos definidos, los que impactan en todas las instancias de la clase modificada y en sus subclases. Los cambios posibles al contenido de una clase son los siguientes:

- Cambios a un atributo
  - ✓ Agregar un nuevo atributo a una clase
  - ✓ Borrar un atributo existente de una clase
  - ✓ Cambiar el nombre de un atributo de una clase
  - ✓ Cambiar el dominio de un atributo

- ✓ Cambiar el valor por defecto (valor de clase) de un atributo

- Cambios a un método
  - ✓ Agregar un método a una clase
  - ✓ Borrar un método existente de una clase
  - ✓ Cambiar el nombre de un método de una clase
  - ✓ Cambiar el código de un método de una clase

- Cambiar el nombre de una clase

3) Cambios en la organización de la jerarquía.

Los cambios de las relaciones de subclase / superclase incluyen operaciones tales como agregado y borrado de una clase. Debe considerarse que la herencia múltiple sólo está permitida en la jerarquía de nodos. Las posibles operaciones a realizar son las siguientes:

- Cambios a la relación de superclase / subclase
  - ✓ agregar una clase S a las superclases de la clase C (sólo para jerarquía de nodos)
  - ✓ remover una clase S de la lista de superclases de C (sólo para jerarq. de nodos)
  - ✓ agregar una nueva clase
  - ✓ borrar una clase existente
  - ✓ crear una clase C como una generalización de n clases existentes (sólo para jerarquía de nodos)
  - ✓ particionar una clase C en nuevas clases

Para preservar la consistencia de la información el modelo presenta un conjunto de invariantes y un conjunto de reglas. El conjunto de invariantes provee las bases para las especificaciones semánticas de cambios del esquema. Las reglas son utilizadas para elegir una forma de preservar las invariantes (cuando se presenta más de una opción).

Las invariantes para modificación del esquema son las siguientes:
- **Invariante de jerarquía de clases** : Cada jerarquía es un grafo dirigido acíclico (DAG). La raíz del DAG para cada una de las jerarquías son clases definidas por el sistema llamadas Nodos, Ejemplares y Links para la jerarquía de nodos, ejemplares y links respectivamente.
- **Invariante de nombre**: Una clase para una jerarquía determinada tiene nombre único y todos los atributos y métodos definidos en una clase tienen nombre único.
- **Invariante de origen**: Si una clase tiene más de una superclase con atributos o métodos con el mismo nombre la clase sólo puede heredar un atributo y/o método.
- **Invariante de herencia completa**: Una clase hereda todos los atributos y métodos de cada una de sus superclases, salvo que viole la invariante de nombre y de origen.

Cuando existe más de una forma de preservar las invariantes, las reglas ayudarán a decidir cómo se preservarán. El modelo clasifica las reglas en cuatro categorías: reglas de resolución de conflictos, reglas de herencia, reglas de dominio y regla de jerarquía de clases.

Dentro de las reglas utilizadas en caso de producirse un conflicto de nombres o de orígenes, la Nº 2 es distinta porque resolvemos de otra forma los problemas producidos por la herencia múltiple (sólo en la jerarquía de clases nodos).

Regla 1: Si el nombre de un atributo o método localmente definido para una clase provoca conflicto de nombre con un atributo o método heredado, entonces el atributo o método localmente definido es seleccionado. En cambio, si un atributo o método localmente definido provoca conflicto con un atributo o método ya definido para la clase, entonces el nuevo atributo o método es desechado.

Regla 2: Si dos o más superclases de una clase tienen un atributo o método con el mismo nombre, pero diferentes ancestros, entonces ambos atributos o métodos son heredados con la salvedad que ambos serán antecedidos con el nombre de la clase de la que son heredados.

Regla 3: Si dos o más superclases de una clase tienen atributo con el mismo nombre y origen, entonces la clase hereda atributos con el dominio más especializado.

Utilizaremos las mismas reglas que las definidas por [KIM95] para preservar la invariante de herencia completa y la que permite que el dominio de un atributo sea generalizado preservando la invariante de dominio.
Estas dos reglas ayudan a preservar la invariante de jerarquía de clases, la que al ser violada causa la desconexión de algunas clases.

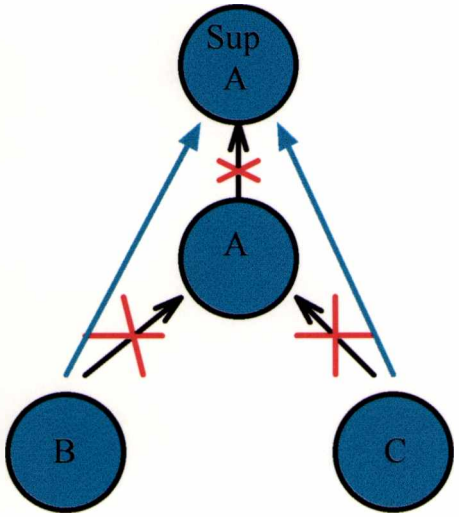
Regla 7: Si cuando una clase es creada no se especifica la lista de superclases, el sistema asumirá como superclase la clase Nodos si la jerarquía asociada es ND, la clase ejemplares si la jerarquía asociada es EJ y clase LINKS si la jerarquía asociada es LK.

Regla 8: Si la clase A es la única superclase de B y A es removida de la lista de superclases de B, entonces B es subclase directa de cada una de las superclases de A.

La denominada **Regla Nº 9** por [KIM95] la hemos eliminado porque no tiene incidencia sobre nuestro modelo, ya que determina el orden para agregar una nueva superclase para evitar conflictos en la herencia. Como ya hemos explicado, esto no afectaría a la integridad de nuestra jerarquía, porque en el momento de la herencia no tenemos en cuenta dicho orden.

Esta regla provee de un orden específico para el borrado de un nodo evitando de este modo cambios muy costosos.

Regla 9: El borrado de un nodo A se realiza en tres pasos:
1) todas las aristas desde A a sus subclases son removidas
2) todas las aristas dirigidas a A desde sus superclases son removidas
3) el nodo A es borrado



Sin embargo debemos agregar la siguiente regla, que es muy importante para mantener la integridad de las relaciones entre las distintas jerarquías.

Regla 10: Cuando una clase nodo es borrada, también deben borrarse los ejemplares asociados y las relaciones que la involucren. Es aplicado el mismo orden de borrado que la regla 9.

4.5.2.- Lenguaje de manejo de datos (DML)

El lenguaje de manejo de datos permite la creación, actualización, borrado y consultas de datos. La sintaxis para cada una de las operaciones es detallada a continuación.

- Creación de instancias
Esta operación permite crear instancias de cualquier clase nodo o clase link. Su sintaxis es la siguiente:

```
Id-Jerarquía CREAR_INSTANCIA Nombre_Clase :Atributo1  Valor1
                                          :Atributo2  Valor2
                                              .           .
                                          :Atributo n  Valor n
```

Id-Jerarquía : los valores pueden ser ND o LK.
Nombre_Clase: es el nombre de la clase a la que estará asociada la nueva instancia. Sólo pueden estar asociadas a la jerarquía de nodos y a la jerarquía de links.

Este mensaje retorna un nuevo objeto, por lo tanto podrá utilizarse en forma anidada en cualquier lugar donde un objeto sea requerido.

Supongamos que deseamos crear una instancia de la clase nodo ERAS.
Si los periodos estuvieran instanciados utilizaríamos el mensaje Crear-Instancias anidado con elige-objeto, pero si no existe debemos hacer la creación en forma anidada.

```
ND CREAR_INSTANCIA  eras
                              :nombre        'CENOZOICA'
                              :título_fon    nombre_archivo
                              :descr_imag    nombre_archivo
                              :descripción   '.............'
                              :año_desde     '1000000 años atrás'
                              :año_hasta     '800000   años atrás'
                              :eras   (ND CREAR_INSTANCIA  periodos
                                       :nombre_per     '.....'
                                  :título_fon     ))))))))))))
                                  :descr_imag
                                  :descripción   '.............'
                                  :año_desde     '......años atrás'
                                  :año_hasta     '.......años atrás'),
                                       (ND CREAR_INSTANCIA  periodos
                                   :nombre_per     '.....'
                                  :título_fon      ))))))))))))
                                  :descr_imag
                                  :descripción   '.............'
                                  :año_desde     '......años atrás'
                                  :año_hasta     '.......años atrás')).
```

Ahora, supongamos que deseamos relacionar la era Cenozoica con uno de los animales que la habitaron, en particular con el Tyranosaurus Rex , para esto deberíamos instanciar la relación vivieron_durante definida en el apéndice B de la siguiente manera.

```
LK CREAR_INSTANCIA  vivieron_durante
        :from (SELECT eras FROM e:eras
                WHERE nombre = 'CENOZOICA')
        :to    (SELECT especies_anim FROM especies_anim
                WHERE nombre = 'Tyranosaurus Rex')
```

- Borrado de Instancias

Para borrar las instancias de una clase nodo o link determinada deberá utilizarse el siguiente mensaje :

```
Id-Jerarquía  BORRAR_INSTANCIA  Nombre_Clase
                        [WHERE  predicado]
```

Id-Jerarquía : los valores pueden ser ND o LK.
Nombre_Clase es el nombre de la clase a la que están asociadas las instancias a ser borradas.
predicado: permite restringir las instancias a borrar. Similar al definido en 4.3.1. Si no se especifican un predicado todas las instancias de la clase serán borradas.

Cuando una instancia A de una clase nodo es borrada también serán borradas las instancias de las relaciones (links) que involucren a A .

El mensaje es propagado a todas las subclases de la clase receptora.
Veamos un ejemplo: Supongamos que queremos borrar el periodo Triásico

ND BORRAR_INSTANCIA periodos WHERE nombre_per=' Triasico'

-Modificación de instancias

Para modificar los valores de ciertos atributos de todas las instancias de una clase nodo o link que satisfaga un predicado se utilizará el siguiente mensaje:

Id-Jerearquía MODIFICAR_INSTANCIA Atributo_Nombre1 Nuevo_Valor1,
Atributo_Nombre2 Nuevo_Valor2..
[WHERE predicado]

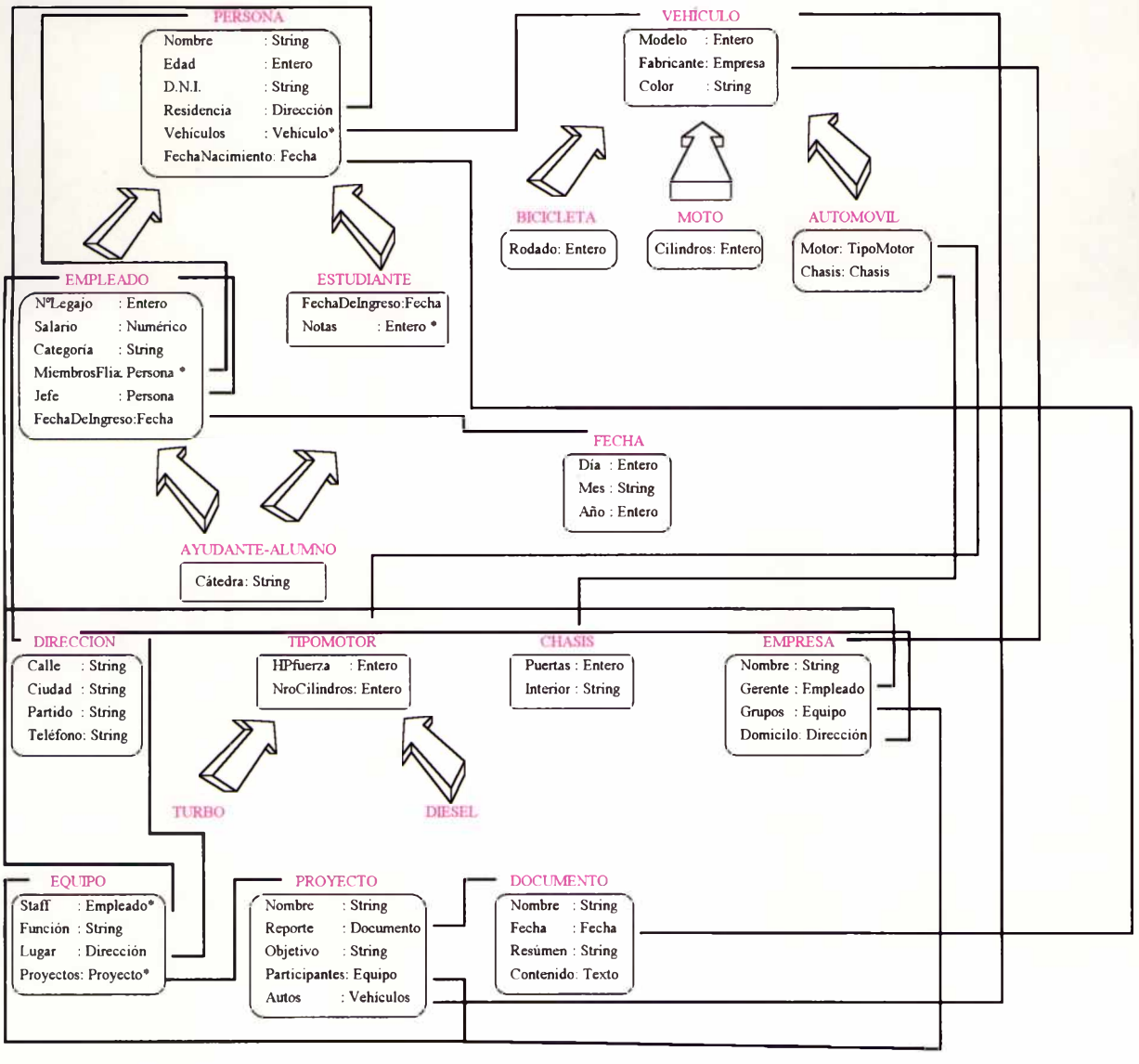
Id-Jerarquía : los valores pueden ser ND o LK.
Nombre_Clase es el nombre de la clase a la que están asociadas las instancias a ser borradas.
predicado: permite restringir las instancias a borrar. Similar al definido en 4.3.1. Si no se especifica un predicado todas las instancias de la clase serán borradas.
El mensaje es propagado a todas las subclases de la clase receptora.

4.5.3.- Lenguaje de control de datos (DCL)

Debe permitir la especificación de transacciones, control de integridad semántica, autorización y manejo de métodos de acceso. Las facilidades para especificar transacciones son las mismas que en sistemas de Bases de Datos convencionales y las restricciones de integridad semántica pueden ser especificadas en métodos asociados a las clases.

APÉNDICE A

Esquema 1 : Usado para presentar los ejemplos en el Capítulo 1



Grafo de Agregación

Jerarquía Es Un

APENDICE B

EJEMPLO: Dinosaurios (Microsoft)

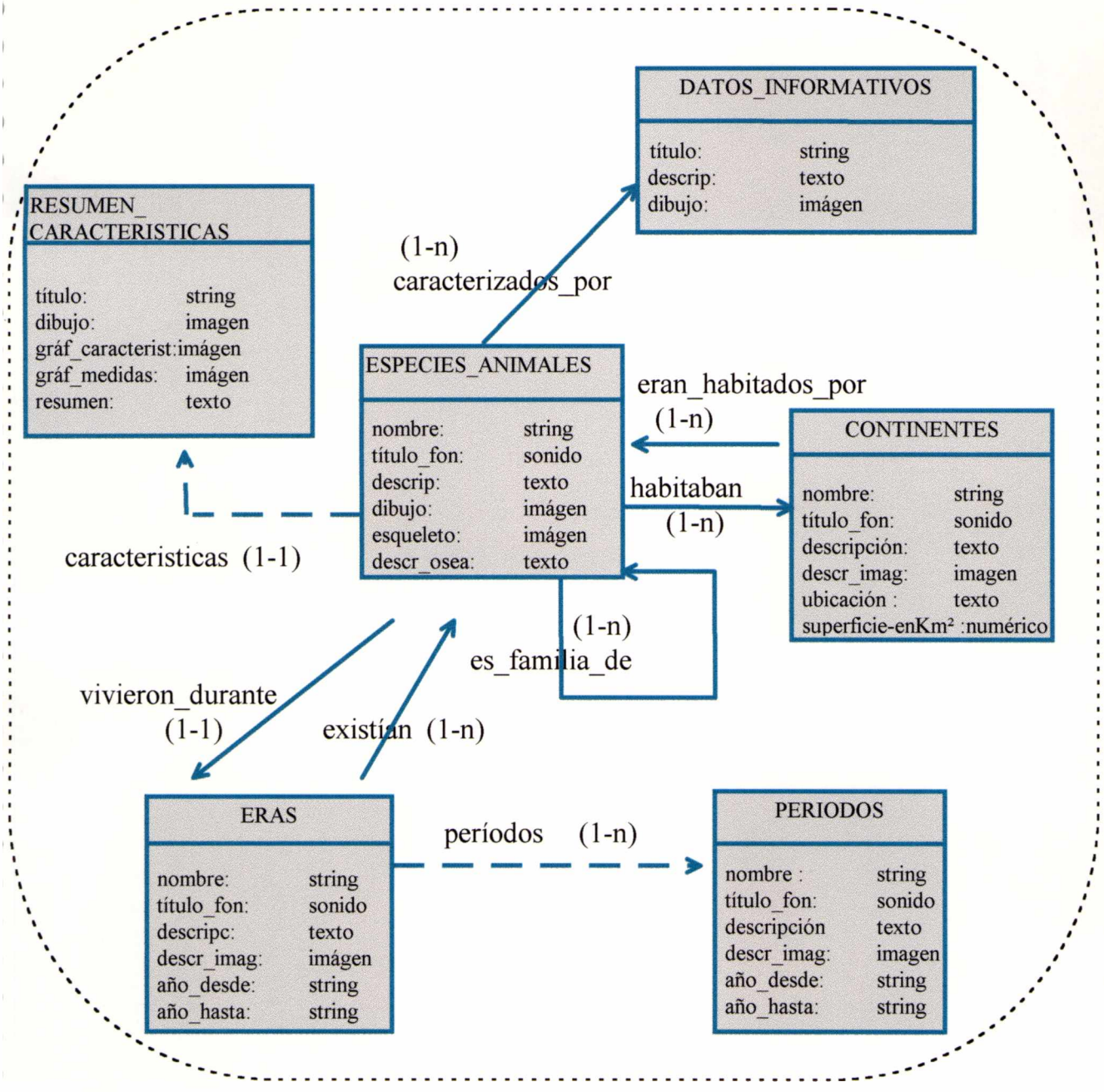
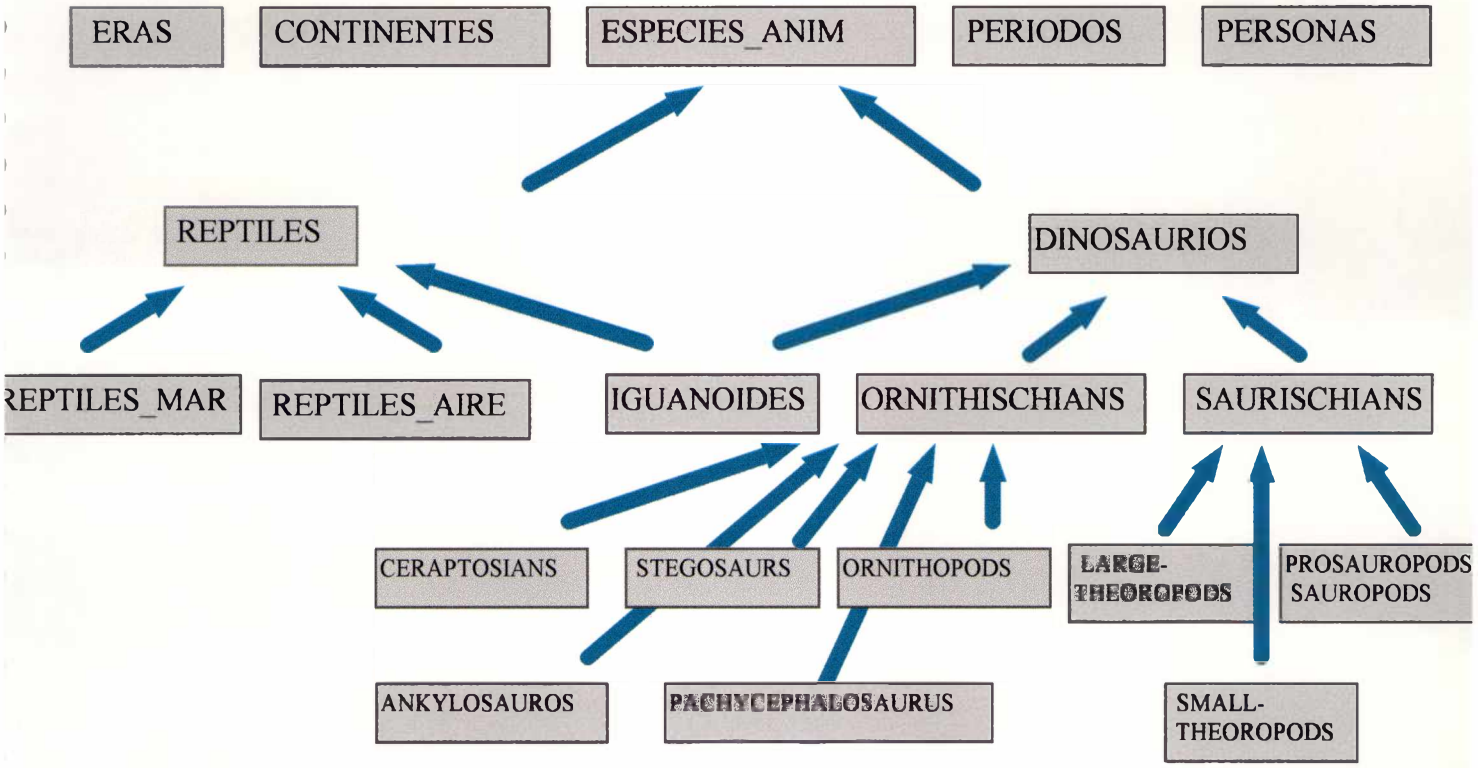


DIAGRAMA DE CLASES NODOS



DATOS INFORMATIVOS

RESUMEN CARACTERISTICAS

DEFINICION

DEFINICION DE CLASES NODOS

clase RESUMEN-CARACTERISTICAS
superclase {NODOS}
título :string
graf_característica : imágen
graf_medidas : imágen
resúmen : texto

clase DATOS-INFORMATIVOS
superclase {NODOS}
título : string
titulo_fon : sonido
descripción : texto (con anchors)
dibujo : imágen

clase DEFINICION
superclase {NODOS}
título : string
descripción : texto (sin anchors)

clase ERAS
superclase {NODOS}
nombre : string
titulo_fon : sonido
descripc : texto
descrip_imag : imágen
año_desde : string
año_hasta : string
períodos : set of (PERIODOS)

clase CONTINENTES
superclase {NODOS}
nombre : string
título_fon : sonido
descripción : texto
descr_imag : imágen
ubicación : texto
superficie-enK² : numérico

clase PERIODOS
superclase {NODOS}

nombre : string
título_fon : sonido
descripcion : texto
descr_imag : imágen
año_desde : numérico
año_hasta : numérico

clase PERSONAS
superclase {NODOS}
nombre : string
foto : imagen
fecha de nacimiento : fecha

clase ESPECIES_ANIMALES
superclase {NODOS}
nombre : string
título_fon : sonido
descrip : texto
dibujo : imagen
esqueleto : imagen
descr_osea : texto
características : RESUMEN-CARACTERISTICAS

clase REPTILES
superclase {ESPECIES_ANIMALES}
época_vivió : string
alimentación : string
temperatura_sangre : string
descrip_rep : multidominio(texto, sonido)

clase REPTILES_MAR
superclase {REPTILES}
sistema_respiratorio: string (branquial o pulmonar)

clase REPTILES_AIRE (también llamados pterosaurus)
superclase {REPTILES}
cola : string (con o sin)

clase DINOSAURIOS
superclase {ESPECIES_ANIM}
hips : string

clase SAURICHIANS
superclase {DINOSAURIOS}
alimentación : string

dentadura : string

clase SAUROPODS
superclase {SAURICHIANS}
cuellos : string

clase LARGE_THEROPODS
superclase{SAURICHIANS}
movimiento : string

clase SMALL_THEOROPODS
superclase{SAURICHIANS}
movimiento :string

clase ORNITICHIANS
superclase {DINOSAURIOS}

clase CERAPTOSIANS
superclase {ORNITICHIANS}
características: string
alimentación : string

clase ANKYLOSAURUS
superclase{ORNITICHIANS}
características: string

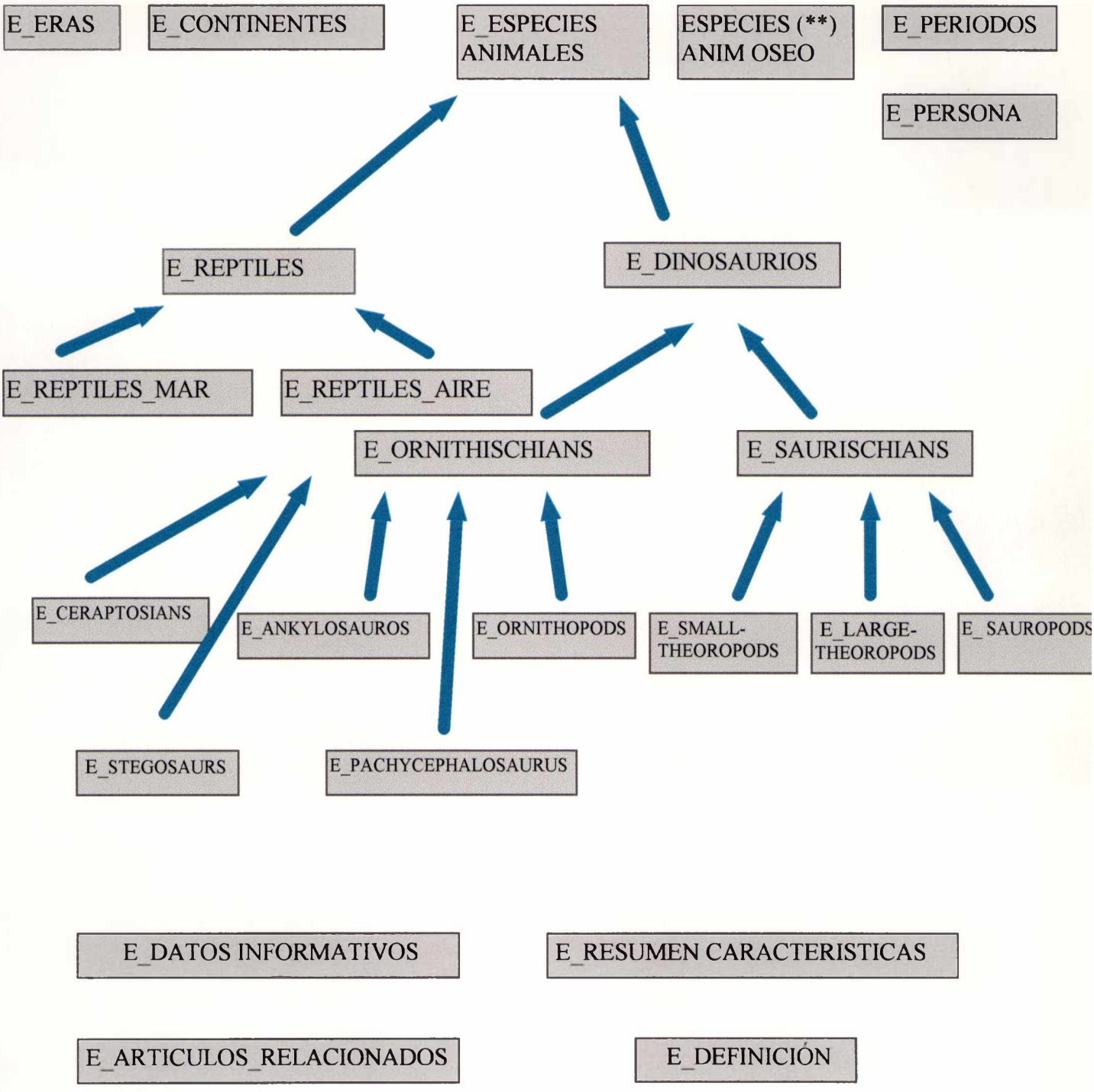
clase PACHYCEPHALOSAURUS
superclase {ORNITICHIANS}
características : string

clase STEGOSAURUS
superclase {ORNITICHIANS}
alimentación : string
movimientos : string
características:string

clase ORNITHOPODS
superclase {ORNITICHIANS}
alimentación : string
características:

clase IGUANIDOS
superclase {REPTILES, DINOSAURIOS}
locomoción : string

DIAGRAMA DE EJEMPLARES



DESCRIPCION DE EJEMPLARES

clase E_DATOS_INFORMATIVOS
superclase {EJEMPLARES}
clase {DATOS_INFORMATIVOS}
título : string
descripcion : texto
dibujo : imagen

clase E_ARTICULOS_RELACIONADOS
superclase {EJEMPLARES}
clase {DATOS_INFORMATIVOS}
título : string
título_fon : sonido
descripcion : texto
dibujo : imagen

clase E_DEFINICION
superclase {EJEMPLARES}
clase {DEFINICION}
título : string
descripcion : texto

clase E_CONTINENTES
superclase {EJEMPLARES}
clase {CONTINENTES}
nombre : string
descripción : texto
descr_imag : imagen

clase E_ERAS
superclase {EJEMPLARES}
clase {ERAS}
nombre :string
descripción : texto
descr_imag : imágen
año_desde : numérico
año_hasta : numérico

clase E_PERIODOS
superclase {EJEMPLARES}

clase {PERIODOS}
nombre : string
descripción : texto
descr_imag : imágen
año_desde : numérico
año_hasta : numérico

clase E_ESPECIES_ANIMALES
superclase {EJEMPLARES}
clase {ESPECIES_ANIMALES}
nombre : string
descripción : texto
descr_imag : imágen
características : RESUMEN-CARACTERISTICAS

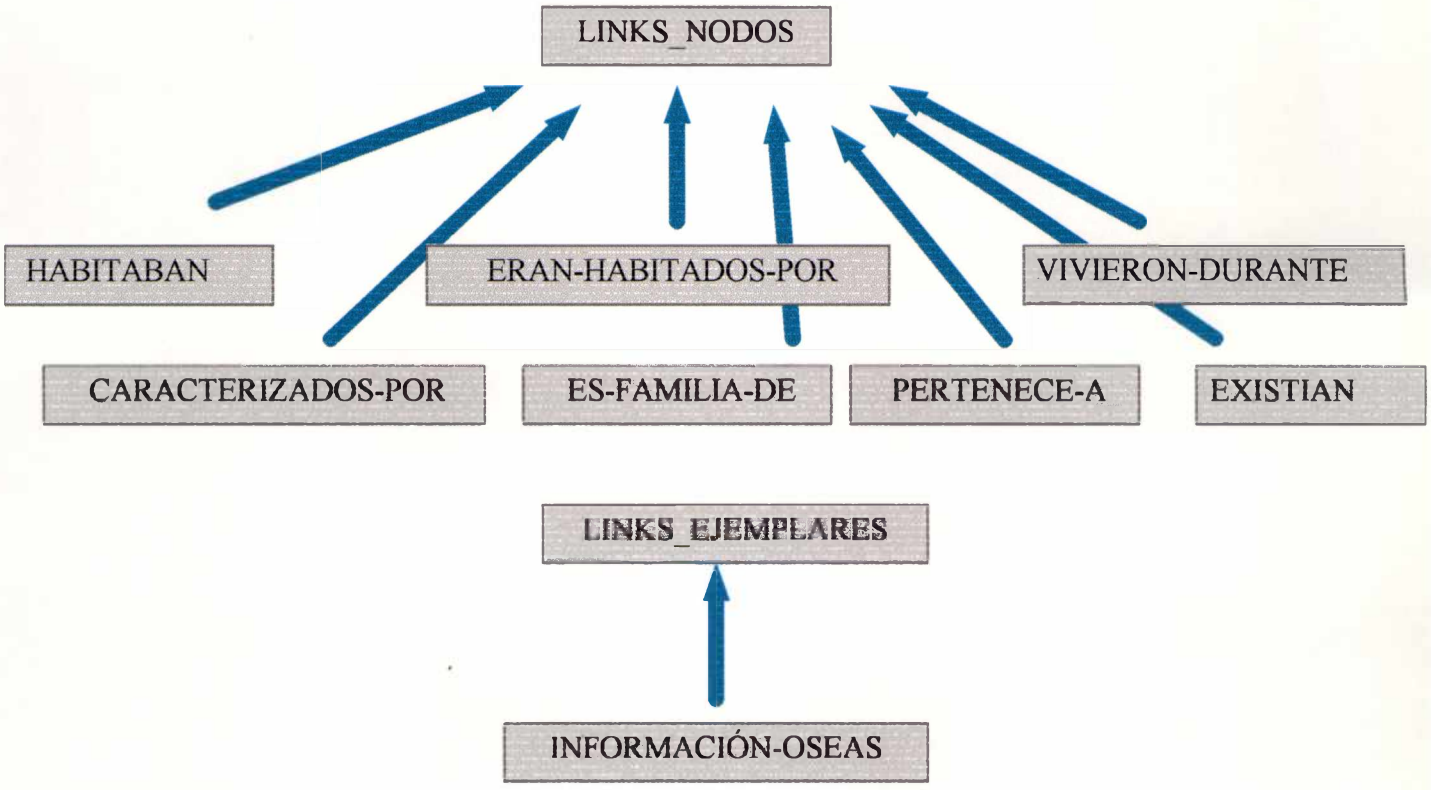
clase ESPECIES_ANIM_OSEO
superclase {EJEMPLARES}
clase {ESPECIES_ANIMALES}
nombre : string
esqueleto : texto
descr_oseo : imágen
características : RESUMEN-CARACTERISTICAS

clase E_REPTILES
superclase {E_ESPECIES_ANIMALES}
clase {REPTILES}
/* los atributos que se muestran son todos heredados */

clase E_SAUROPODS
superclase {E_ESPECIES_ANIMALES}
clase {SAUROPODS}

/* los atributos que se muestran son todos heredados */

DIAGRAMA DE RELACIONES



DESCRIPCION DE RELACIONES

clase HABITABAN
superclase {LINKS_CLASES}
from : ESPECIES_ANIMALES
to : CONTINENTE
ícono : IMAGEN
cardinalidad : 1-n

clase EXISTIAN
superclase {LINKS_CLASES}
from : ERAS
to : ESPECIES_ANIMALES
ícono : IMAGEN
cardinalidad : 1-n

clase ERAN_HABITADOS_POR
superclase {LINKS_CLASES}
from : CONTINENTE
to : ESPECIES_ANIMALES
ícono : IMAGEN
cardinalidad : 1-n

clase VIVIERON_DURANTE
superclase {LINKS_CLASES}
from : ESPECIES_ANIMALES
to : ERAS
ícono : IMAGEN
cardinalidad : 1-1

clase CARACTERIZADOS_POR
superclase {LINKS_CLASES}
from : ESPECIES_ANIMALES
to : DATOS_INFORMATIVOS
ícono : IMAGEN
cardinalidad : 1-n

clase ES_FAMILIA_DE
superclase {LINKS_CLASES}
from : ESPECIES_ANIMALES
to : ESPECIES_ANIMALES
ícono : IMAGEN
cardinalidad : 1-n (por iguanoide)

clase INFORMACION_OSEA

superclase {LINKS_EJEMPLARES}
from : ESPECIES_ANIM_OSEO
to : E_DATOS_INFORMATIVOS
ícono : IMAGEN
cardinalidad : 1-n

BIBLIOGRAFIA

[AMA92] - B. Amann, M. School, "Gram: Graph Data Model and Query Language ",
ACM Echt Conference, Diciembre 1992.

[BER92] - E. Bertino, M. Negri, G. Pelagatte and L. Sbattella,"Object Oriented Query Language: The Notion and the Issues",
IEEE Transactions on Knowledge and Data Engineering, Junio 1992.

[BRA92] - P. De Bra, G. Houben, Y. Kornatzky, " An Extensible Data Model for Hyperdocuments",
ACM Echt Conference, Diciembre 1992.

[CAR95] - H. Caro, " Introducción al Mundo de Internet ", Compumagazine, Año VII, Número 84, pp. 66-102, 1995.

[CON87] - Jeff Conklin , "Hypertext : An Introduction and Survey",
IEEE Computer, pp. 17-40, Septiembre 1987.

[FUL91] - M. Fuller, A. Kent, R. Sacks David, J.Thom, R. Wilkinson, J. Zobel, "Querying in a Large Hyperbase",
DEXA 91.

[GAR91] - F. Garzotto, P. Paolini, D. Schwabe , "HDM - A model for the Design of Hipertext Aplications ",
Hypertext '91 Proceedings, Diciembre 1991.

[HAL88] - F.G. Halasz, " Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems ",
Communication of ACM , Julio 1988.

[ISA94] - T. Isakowitz, E. A. Stohr, P. Balasuvramanian, " RM: A Methodology for the Design of Structured Hypermedia Applications ",Noviembre 1994.

[KIM95] - W. Kim, "A Model Database System - The Object Model, Interoperability and Beyond", Addison Wesley, 1995.

[KIF92] - M. Kifer, W. Kim, Y. Sagiv, "Querying Object-Oriented Databases",
ACM Echt Conference, Diciembre 1992.

[LAL89] - W. R. Lalonde, " Designing Families of Data Types Using Exemplars ",
ACM Transactions on Programming Languages and Systems, Abril 1989.

[NAN91] - J. Nanard, M. Nanard, "Using Structured Types to incorporate Knowledge in Hypertext",
Hypertext '91 Proceedings, Diciembre 1991.

[MAR92] - M. Marmann, G. Schlageter, " Towards a Better Support for Hypermedia Structuring: the HYDESIGN Model ",
ACM Echt Conference, Diciembre 1992.

[NIE95] - J. Nielsen , "Multimedia and Hypertext",
Academy Press Inc., 1995

[SCH94] - D. Schwabe, G. Rossi, " OOHDM. An Object Oriented Hypermedia Design Model ",
Technical Report , PUC-Rio, 1994.

[WAN91] - B. Wang, P. Hichcock , "InterSect: General Purpose Hypertext System based on an Object Oriented Database",.
DEXA91.

[ZHE92] - Yi Zheng, M. Pong, " Using Statecharts to Model Hypertext ",
ACM Echt Conference, Diciembre 1992.