Esquemas e Instancias en Bases de Datos Orientadas a Objetos con Tipos y Clases

Gustavo G. Larriera
(Grupo de Proyecto OO) *
Reporte 900309

Marzo de 1990

Resumen

Se presentan formalmente en este informe los conceptos de *esquema* e *instancia de un esquema* en sistemas de bases de datos orientados a objetos, en los que coexisten pacificamente las nociones de *tipo* y *clase*. Adicionalmente se definen los conceptos de *subtipo* y *subclase*, con mecanismo de herencia simple.

1 Introduccion

El modelo de bases de datos orientadas a objetos que vamos a presentar se basa en las nociones de *tipo* y *clase*. Las instancias de las clases son *objetos con identidad*, y las instancias de los tipos son *valores complejos* construidos en base a valores atomicos, tuplas y conjuntos. Creemos que un sistema de base de datos orientado a objetos debe trabajar con ambas nociones. La nocion de tipo aporta *propiedades intensionales (estructurales)* al esquema de la base de datos. La nocion de clase aporta *propiedades extensionales (valores)*. Los tipos nos permiten definir la estructura de la base de datos mientras que las clases nos permiten almacenar objetos en ella.

En nuestra concepcion, un usuario de un sistema de base de datos orientado a objetos deberia disponer de:

1. *Clases:* cuyas instancias son objetos que encapsulan datos y comportamiento (estado interno y mensajes).

*en desarrollo en: Escuela de Informatica, Cuareim 1522, Montevideo-Uruguay.

2. *Tipos:* cuyas instancias son valores que son componentes privadas de los objetos.

En este marco, a cada clase se le asocia un unico tipo que describe la estructura de las instancias de la clase. Las operaciones (mensajes) aplicables a las instancias de la clase son las operaciones asociadas al tipo de la clase.

2 Un ejemplo

A continuacion presentaremos un ejemplo sencillo que ilustre las principales caracteristicas que vamos a tratar. Supongamos la siguiente definicion del esquema de una base de datos en una escuela de Informatica:

Ejemplo 2.1 `Tipos y clases`

```
;;; Declaracion de tipos y su estructura
type TipoPersona =
   <nombre:String;
    edad:Integer;
    hijos:{TipoPersona} >.

type TipoCurso = <nomCurso:String; docente:TipoPersona>.

type TipoEstudiante isA TipoPersona =
   <aprobaciones:{ <c:TipoCurso; pts:Integer> } >.

;;; Declaracion de clases y sus tipos asociados
class Persona:TipoPersona.
class Estudiante:TipoEstudiante.
class Curso:TipoCurso.
```

En este ejemplo podemos observar varios puntos de interes que forman parte de la declaracion del esquema de la base de datos, realizado en algun lenguaje de definicion de datos. En particular tenemos

1. *Identificadores de tipos:* tales como TipoPersona, TipoCurso, TipoEstudiante, String o Integer.

2. *Representaciones de tipos:* que aparecen asociadas a los identificadores de tipos. Entre las construcciones podemos notar que hay representaciones de tuplas y de conjuntos (por ejemplo, la representacion del atributo aprobaciones en TipoEstudiante es un conjunto de tuplas.

3. *Atributos:* como son nombre, edad o nomCurso.

4. *Declaraciones de tipos:* que nos asocian a cada identificador de tipo con una representacion.

5. *Jerarquia entre tipos:* por ejemplo, TipoEstudiante es subtipo de TipoPersona, esto implicara que la representacion de TipoEstudiante tiene todos los atributos de la representacion de TipoPersona mas el atributo aprobaciones.

6. *Identificadores de clases:* tales como Persona, Estudiante o Curso.

7. *Declaraciones de clases:* que nos asocian a cada identificador de clase con un tipo (por ejemplo, la clase Persona es de tipo TipoPersona).

Veremos en el proximo ejemplo como crear instancias de cada clase y asignarles valores:

Ejemplo 2.2 `Instancias y valores`

```
;;; Creacion de instancias de las clases
object ingles:Curso new.
object poesia:Curso new.
object juan:Persona new.
object todd:Estudiante new.

;;; Asignacion de valores a las instancias
juan := <nombre:'John Keating',edad:36,hijos:nil>.
ingles := <nomCurso:'Idioma Ingles',docente:juan>.
poesia := <nomCurso:'Poesia',docente:juan>.
todd := <nombre:'Todd Anderson',edad:15,hijos:nil,
         aprobaciones:{ <c:poesia,pts:10>,<c:ingles,pts:5> } >.
```

En este ejemplo hemos creado cuatro objetos o instancias de las clases mencionadas y le hemos asignado valores a cada uno de ellos. Observemos que los valores tienen una representacion que debe ser compatible con la representacion del tipo asociado a la clase de la instancia.

3 Tipos

3.1 Representacion de tipos

Sean los siguientes conjuntos disjuntos:

1. El conjunto *dominio* $\mathcal{D} = \mathcal{Z} \cup \mathcal{R} \cup \mathcal{S} \cup \mathcal{L}$ donde $\mathcal{Z}$ es el conjunto de los numeros enteros, $\mathcal{R}$ es el conjunto de los numeros reales, $\mathcal{S}$ es el conjunto de los strings de cualquier largo y $\mathcal{L}$ es el conjunto de valores logicos {true,false}. Consideraremos al valor nulo *nil* perteneciente a $\mathcal{D}$.

2. El conjunto $\mathcal{A}$ que es un conjunto infinito numerable de simbolos que representan *atributos* definidos por el usuario (e.g. nombre, edad, aprobaciones).

3. El conjunto $\mathcal{I}_T$ que es un conjunto infinito numerable de simbolos que representan *identificadores de tipos* definidos por el usuario (e.g. TipoPersona, TipoEstudiante). Asumiremos que Integer, Real, String y Boolean pertenecen a $\mathcal{I}_T$.

Definicion 3.1 *El conjunto* $\mathcal{R}_T$ *de* representaciones de tipos *se define como*

1. Representacion de tipos basicos: *los elementos de* $\mathcal{D}$ *pertenecen a* $\mathcal{R}_T$.
2. Representacion de tipos tupla: *si* $a_1, \ldots, a_n$ *(distintos entre si) pertenecen a* $\mathcal{A}$ *y* $t_1, \ldots, t_n$ *pertenecen a* $\mathcal{R}_T$ *o a* $\mathcal{I}_T$*, entonces* $< a_1 : t_1, \ldots, a_n : t_n >$ *pertenece a* $\mathcal{R}_T$.
3. Representacion de tipos conjunto: *si* t *pertenece a* $\mathcal{R}_T$ *o a* $\mathcal{I}_T$ *entonces* $\{$t$\}$ *pertenece a* $\mathcal{R}_T$.

Ejemplo 3.1 `Representaciones de tipos`

```
<nombre:String;
 edad:Integer;
 hijos:{TipoPersona};
 aprobaciones:{ <c:TipoCurso; pts:Integer> } >

<nomCurso:String; docente:TipoPersona>.
```

3.2 Declaracion de tipos

Podemos darle un nombre a cada tipo si asociamos a cada identificador de tipo una representacion, en forma similar a la declaracion *type* del lenguaje Pascal. Para ello utilizaremos una funcion $\mu(\mathcal{I}_T) : \mathcal{R}_T$, y asumiremos que estan predefinidos los siguientes valores:

$\mu(Integer) = \mathcal{Z}$
$\mu(Real) = \mathcal{R}$
$\mu(String) = \mathcal{S}$
$\mu(Boolean) = \mathcal{L}$

Los demas valores de μ son definidos por el usuario mediante la expresion *type* $\mathcal{I}_T = \mathcal{R}_T$.

Ejemplo 3.2 `Tipo y representacion asociada`

```
type TipoPersona =
   <nombre:String;
    edad:Integer;
    hijos:{TipoPersona} >.
```

Definicion 3.2 *Definiremos* conjunto de tipos *como* $\mathcal{T} \subseteq \mathcal{I}_T \times \mathcal{R}_T \times \mathcal{P}(\mathcal{M}_T)$, *donde* $\mathcal{M}_T$ *es un conjunto de* mensajes (operaciones) *aplicables a instancias del tipo y* $\mathcal{P}(\mathcal{M}_T)$ *es el conjunto potencia de* $\mathcal{M}_T$.

Cuando el usuario define un nuevo tipo en el sistema debe

1. Especificar el nombre del tipo (identificador de tipo).
2. Asociar una representacion unica al tipo.
3. Especificar un conjunto de operaciones para el tipo.

Dejaremos para mas adelante todo lo referente a la definicion de operaciones asociadas a los tipos.

4 Valores y objetos

4.1 Valores

Sea $\mathcal{I}_O$ un conjunto infinito numerable de simbolos definidos por el usuario a los que llamaremos *identificadores de objetos* (e.g. juan, poesia, todd).

Definicion 4.1 *El conjunto $\mathcal{V}$ de* valores *se define como*

1. Valores atomicos o elementales: *si un valor* v *pertenece al conjunto $\mathcal{D}$ o a $\mathcal{I}_O$, entonces* v *pertenece a $\mathcal{V}$.*
2. Valores de tuplas: *si $a_1, \ldots, a_n$ (distintos entre si) pertenecen a $\mathcal{A}$ y $v_1, \ldots, v_n$ pertenecen a $\mathcal{V}$, entonces $< a_1 : v_1, \ldots, a_n : v_n >$ pertenece a $\mathcal{V}$.*
3. Valores de conjuntos: *si $v_1, \ldots, v_n$ (distintos entre si) pertenecen a $\mathcal{V}$, entonces $\{v_1, \ldots, v_n\}$ pertenece a $\mathcal{V}$.*

Ejemplo 4.1 `Valores`

```
'John Keating'
36
nil
<nombre:'John Keating',edad:36,hijos:nil>
{ <c:poesia,pts:10>,<c:ingles,pts:5> }
```

4.2 Objetos

Definicion 4.2 *Definiremos un* objeto *como una pareja formada por un identificador de objeto y un valor. Es decir, un objeto* o *es un elemento del producto cartesiano $\mathcal{I}_O \times \mathcal{V}$. Definiremos un* conjunto de objetos *$\mathcal{O} \subseteq \mathcal{I}_O \times \mathcal{V}$ como aquellas parejas del producto cartesiano tales que sus primeras componentes (identificador de objeto) son todas distintas entre si.*

Es importante observar que cada objeto (*id-objeto,valor*) tiene una *identidad* que le es propia y que es independiente del valor asociado al objeto. Decimos que dos objetos son *identicos* cuando son el mismo objeto, y son *iguales* cuando tienen el mismo valor. Es decir, dos objetos son siempre objetos diferentes aun cuando tengan el mismo valor. Resulta obvio que dos objetos identicos son iguales.

La asociacion entre un identificador de objeto y un valor *(asignacion)* la realiza el usuario, tal como se ve en el siguiente

Ejemplo 4.2 `Asignaciones de valores a objetos`

```
juan := <nombre:'John Keating',edad:36,hijos:nil>.
ingles := <nomCurso:'Idioma Ingles',docente:juan>.
poesia := <nomCurso:'Poesia',docente:juan>.
todd := <nombre:'Todd Anderson',edad:15,hijos:nil,
         aprobaciones:{ <c:poesia,pts:10>,<c:ingles,pts:5> } >.
```

Definicion 4.3 *En forma adicional definiremos una funcion $\theta(\mathcal{I}_O) : \mathcal{V}$ que devuelve el valor asociado a cada identificador de objeto, o sea la segunda componente de una pareja (*id-objeto,valor*).*

Para los casos del ejemplo anterior, tenemos los siguientes valores para la funcion θ:

$\theta(juan) =< nombre :' JohnKeating', edad : 36, hijos : nil >$

$\theta(ingles) =< nomCurso :' IdiomaIngles', docente : juan >$

$\theta(poesia) =< nomCurso :' Poesia', docente : juan >$

$\theta(todd) =< nombre :' ToddAnderson', edad : 15, hijos : nil, aprobaciones : \{< c : poesia, pts : 10 >, < c : ingles, pts : 5 >\} >$

4.3 Representacion de valores

Cada valor de $\mathcal{V}$ tiene una representacion unica, es decir existe una funcion que a cada valor le asocia su representacion. A esta funcion la llamaremos *representacion de valor*.

Definicion 4.4 *Definimos la funcion $\tau(\mathcal{V}) : \mathcal{R}_T$ de la siguiente manera:*

1. Representacion de valores atomicos:

   si v *pertenece a $\mathcal{Z}$ entonces* $\tau(\mathrm{v}) = \mathcal{Z}$.

   si v *pertenece a $\mathcal{R}$ entonces* $\tau(\mathrm{v}) = \mathcal{R}$.

   si v *pertenece a $\mathcal{S}$ entonces* $\tau(\mathrm{v}) = \mathcal{S}$.

   si v *pertenece a $\mathcal{L}$ entonces* $\tau(\mathrm{v}) = \mathcal{L}$.

   si v *pertenece a $\mathcal{I}_O$ entonces* $\tau(\mathrm{v}) = \tau(\theta(\mathrm{v}))$.

2. Representacion de valores de tuplas: *si* $a_1, \ldots, a_n$ *(distintos entre si) pertenecen a* $\mathcal{A}$*,* $v_1, \ldots, v_n$ *pertenecen a* $\mathcal{V}$*, entonces* $\tau(< a_1 : v_1, \ldots, a_n : v_n >) =< a_1 : \tau(v_1), \ldots, a_n : \tau(v_n) >$.

3. Representacion de valores de conjuntos: *si* $v_1, \ldots, v_n$ *pertenecen a* $\mathcal{V}$*, entonces* $\tau(\{v_1, \ldots, v_n\}) = \{\tau(v_1), \ldots, \tau(v_n)\}$

5 Clases

Sea $\mathcal{I}_C$ un conjunto infinito numerable de simbolos definidos por el usuario a los que llamaremos *identificadores de clases* (e.g. Persona, Estudiante, Curso).

Una *clase* va a representar a la coleccion de objetos existentes de un mismo tipo en el sistema. Cada clase va a tener un unico identificador del conjunto $\mathcal{I}_C$ y va a estar asociada a un unico identificador de tipo del conjunto $\mathcal{I}_T$. En un sistema de base de datos orientado a objetos, el conjunto de tipos definidos representa a la estructura o esquema de la base de datos, mientras que las clases definidas son los entes que almacenan a los objetos existentes en la base de datos. Podemos decir que la nocion de tipo expresa todos los objetos posibles que pueden llegar a existir durante la vida util de la base de datos, mientras que las clases expresan los objetos que realmente existen en un momento dado. En la terminologia clasica de bases de datos, las clases representan a las *extensiones* o *estados* de la base.

5.1 Declaracion de clases

Cuando una nueva clase es definida, debe asociarse a un identificador de clase un unico identificador de tipo. Para ello utilizaremos una funcion $\sigma(\mathcal{I}_C) : \mathcal{I}_T$ definida por el usuario a traves de la expresion *class* $\mathcal{I}_C : \mathcal{I}_T$.

Ejemplo 5.1 `Declaracion de clases`

```
class Persona:TipoPersona.
class Estudiante:TipoEstudiante.
class Curso:TipoCurso.
```

En el ejemplo anterior se han declarado los siguientes valores para la funcion σ:

$\sigma(Persona) = TipoPersona$
$\sigma(Estudiante) = TipoEstudiante$
$\sigma(Curso) = TipoCurso$

5.2 Creacion de instancias de clases

Una *instancia de clase* representa a un objeto cuyos valores son del mismo tipo que el tipo asociado a la clase. En la literatura se manejan los conceptos de instancia y clase como sinonimos. Cuando creamos una instancia de clase mediante la declaracion *object* $\mathcal{I}_O : \mathcal{I}_C$ *new*, la instancia obtiene automaticamente una estructura que es la representacion del tipo asociado a la clase.

Ejemplo 5.2 `Creacion de instancias de clases`

```
object ingles:Curso new.
object poesia:Curso new.
object juan:Persona new.
object todd:Estudiante new.
```

Las instancias de clases que van siendo creadas pueden recibir valores cuyas representaciones deben ser las mismas representaciones de los tipos asociados a las clases usadas para crear las instancias. Todas las instancias de una clase pertenecen a ella. Es decir, la clase se encarga de dos tareas especificas:

1. *Creacion de instancias:* la clase aporta una estructura homogenea a todos los objetos creados en ella. Esta estructura esta dada por la representacion del tipo asociado a la clase.

2. *Almacenamiento de instancias:* la clase recolecta a todas las instancias creadas en ella. Es posible aplicar operadores a todos las instancias de una clase en particular, lo que constituye el lenguaje de manipulacion y consulta a la base de datos. Al conjunto de objetos pertenecientes a una clase lo llamaremos *extension* o *instancia de la clase*.

6 Subtipos

La nocion de *subtipo* es una de las caracteristicas mas destacables de la orientacion a objetos. Si un tipo t es subtipo de un tipo *t'*, entonces cuando en el contexto se espera un valor de tipo *t'* puede sustituirse por un valor de tipo t, pero no viceversa. La relacion *subtipo* puede inferirse automaticamente entre representaciones de tipos, pero debe declararse explicitamente en tipos abstractos mediante la declaracion $\mathcal{I}_T$ *isA* $\mathcal{I}_T$.

6.1 Jerarquia de representaciones

Definicion 6.1 *Entre los elementos de $\mathcal{R}_T$ se establece una relacion de orden* subtipo *definida como $\leq \subseteq \mathcal{R}_T \times \mathcal{R}_T$. Las reglas de inferencia son las siguientes:*

1. $t \leq t, \forall t \in \mathcal{R}_T$

2. $< a_1 : t_1, \ldots, a_{m+1} : t_{m+1}, \ldots, a_n : t_n > \leq < a_1 : t'_1, \ldots, a_m : t'_m >$ *ssi* $t_i \leq t'_i, \forall t_i, t'_i \in \mathcal{R}_T, 1 \leq i \leq m$

3. $\{t\} \leq \{t'\}$, *ssi* $t \leq t', \forall t, t' \in \mathcal{R}_T$

Ejemplo 6.1 $< nombre : String; edad : Integer > \leq < nombre : String >$
$\{< nombre : String; edad : Integer >\} \leq \{< nombre : String >\}$

6.2 Jerarquia entre tipos

La relacion *subtipo* entre tipos abstractos definidos por el usuario debe declararse explicitamente como una relacion de orden $\leq \subseteq \mathcal{I}_T \times \mathcal{I}_T$.

Definicion 6.2 *Siendo* i, i' *elementos de* $\mathcal{I}_T$, $i \leq i'$ *cuando se declare explicitamente usando la declaracion* i isA i'. *La representacion de* i *va a ser la union de la representacion de* i' *mas los atributos nuevos declarados.*

Ejemplo 6.2 `Subtipos`
```
type Curso = <nomCurso:String>
type TipoPersona = <nombre:String; edad:Integer>
type TipoEstudiante isA TipoPersona = <cursando:TipoCurso>
```

En el ejemplo anterior, tenemos que: μ(Curso) = <nomCurso: String>, μ(TipoPersona) = <nombre: String; edad: Integer>. Como caso especial debido a la herencia, tenemos que μ(TipoEstudiante) = <nombre: String; edad: Integer; cursando: TipoCurso>.

En nuestro modelo manejamos unicamente mecanismos de *herencia simple*: un tipo es subtipo de un unico tipo. La resolucion de conflictos de nombres al especificarse la herencia se resuelve por *overriding*: prevalecen los nombres definidos en los tipos mas especializados (i.e. los subtipos). Un tratamiento exhaustivo del tema excede al presente reporte.

7 Subclases

La jerarquia entre tipos establece directamente una jerarquia entre las clases. Esta jerarquia no es declarada por el usuario sino que queda establecida a partir de la jerarquia definida entre los tipos.

Definicion 7.1 *Siendo* c *y* c' *identificadores de clases, decimos que* $c << c'$ (c es subclase de c') *ssi* $\mu(\sigma(c)) \leq \mu(\sigma(c'))$.

Las nociones de subtipo y subclase tienen diferencias claras en nuestro modelo, que se ajustan a las mencionadas en [ACO85]:

1. La nocion de subtipo expresa aspectos estaticos del lenguaje y se utiliza a los efectos de establecer reglas de compatibilidad entre los valores de un tipo y los valores de sus supertipos.

2. La nocion de subclase expresa restricciones de existencia (i.e. aspectos dinamicos) entre los objetos existentes en la base de datos: si $c, c' \in \mathcal{I}_C$ y tenemos que $c << c'$, entonces la extension de c es un subconjunto no estricto de la extension de *c'*.

En todos los casos, cuando un objeto es agregado a una clase, automaticamente se convierte en elemento de su superclase, y transitivamente, de las demas superclases que estan remontando la jerarquia. Analogamente, la eliminacion de un objeto de una clase, lo elimina de todas sus subclases, aunque no de sus superclases.

8 Esquemas e Instancias

8.1 Esquema

El concepto de *esquema* expresa las propiedades intensionales de la base de datos, y lo definiremos de la siguiente manera:

Definicion 8.1 *Un* esquema *es una 4-upla* $(\mathcal{T}, \mathcal{I}_C, \leq, \sigma)$.

Es decir, cuando un usuario define un esquema, debe especificar:

1. El conjunto $\mathcal{T}$ de tipos y su jerarquia $\leq$.
2. El conjunto $\mathcal{I}_C$ de identificadores de clases y la asociacion σ entre cada clase y su tipo.

Por definicion, un esquema es un *esquema consistente:* si $c << c'$ entonces $\sigma(c) \leq \sigma(c')$ [LR89a].

8.2 Instancia de un esquema

El concepto de *instancia de un esquema* o *extension de la base de datos* representa a todos los objetos existentes en la base. Estos objetos son creados en las clases. Dado un esquema, una *instancia del esquema* es el conjunto de clases existentes. El lenguaje de manipulacion de la base de datos dispone de operaciones para agregar objetos a las clases, testear la presencia de un objeto en una clase y quitar un objeto de una clase. El lenguaje de consulta dispone de iteradores que permiten acceder a todos los objetos de una clase y tambien seleccionar un subconjunto de objetos en una clase.

9 Conclusiones y trabajo futuro

El modelo de datos para una base de datos orientada a objetos debe capturar propiedades estaticas y dinamicas. Las propiedades estaticas o intensionales definen el esquema de la base de datos. Hemos definido un esquema en base a un conjunto de tipos complejos definidos por el usuario a partir de tipos basicos. Este conjunto de tipos tiene definida una relacion jerarquica (subtipo) entre sus elementos.

Las propiedades dinamicas o extensionales de la base de datos se reflejan en el concepto de instancia del esquema, un conjunto de clases cuyos elementos son los objetos que constituyen la informacion en la base de datos. Cada clase tiene un tipo asociado. La jerarquia de tipos definida en el esquema establece automaticamente una jerarquia entre las clases (subclase). Si bien no hay consenso general en la literatura referente a los diversos sistemas de bases de datos orientadas a objetos existentes, entendemos que los conceptos de tipo y clase deben ambos existir en un sistema.

El trabajo futuro del Proyecto OO va a enfocar los siguientes temas: operaciones asociadas a los tipos, especificacion de un lenguaje de definicion, manipulacion y consulta, evolucion del esquema de la base de datos y resolucion de conflictos en la herencia simple.

10 Agradecimientos

A Fernando Carpani, porque al intentar explicarle algunos de los conceptos presentados, me genero la suficiente cantidad de dudas como para intentar formalizarlos; a Monica Martinez, por las sostenidas charlas informales y fundamentalmente por el valioso material bibliografico.

Referencias

[AB87] MP Atkinson y PO Buneman. Types and persistence in database programming languages. *ACM Computing Surveys*, 19(2):105–190, Junio 1987.

[ABD*89] MP Atkinson, F Bancilhon, D DeWitt, K Dittrich, D Maier y S Zdonik. The object-oriented database system manifesto. 19 de agosto de 1989.

[ACO85] A Albano, L Cardelli y R Orsini. Galileo: a strongly-typed, interactive, conceptual language. *ACM Transactions on Database Systems*, 10(2):230–260, Junio 1985.

[ADG*89] A Albano, A Dearle, G Ghelli, C Marlin, R Morrison, R Orsini y D Stemple. A framework for comparing type systems for database programming languages. En *2nd. Intl. Workshop on Database Programming Languages*, 1989. Oregon, 4-7 Junio.

[AGOP88] A Albano, F Giannotti, R Orsini y D Pedreschi. The type system of Galileo. En M Atkinson, P Bunemann y D Morrison, editores, *Persistence and Data Types*, pags 101–119, Springer-Verlag, 1988.

[Ban88] F Bancilhon. *Object-Oriented Database Systems.* Invited lecture 7th ACM SIGART-SIGMOD-SIGACT Symposium on Principles of Database Systems, Austin, Texas. Marzo de 1988.

[BM88] F Bancilhon y D Maier. *Multilanguage object oriented database systems: New answers to old database problems?* Reporte tecnico Altair 21-88, 15 de abril de 1988.

[DT88] S Danforth y C Tomlinson. Type theories and object-oriented programming. *ACM Computing Surveys*, 20(1):29–71, Marzo 1988.

[LR89a] C Lecluse y P Richard. *Modeling complex structures in object-oriented databases.* Reporte tecnico Altair 31-89, 22 de agosto de 1989.

[LR89b] C Lecluse y P Richard. *Manipulation of structured values in object-oriented databases.* Reporte tecnico Altair 32-89, 22 de agosto de 1989.

[LR89c] C Lecluse y P Richard. *The O2 database programming language.* Reporte tecnico Altair 26-89, 9 de enero de 1989.