Persistencia: evolución y .net

J. Baltasar García Perez-Schofield

Grupo IMO
Universidad de Vigo

http://webs.uvigo.es/jbgarcia/

Datos, datos, datos ...

- Niklaus Wirth
  - Inventor de los lenguajes de programación Pascal, Modula-2, Oberon
- Autor del libro:
  - Algoritmos + Estructuras de datos = Programas

Un poco de historia ...

- FORTRAN (Formula Translator)
- ALGOL (Algorithmic Language)
- COBOL (Common Bussiness-Oriented Language)
  - Multiplataforma en el 59 ...
- ¿PL/I? (Programming Language One)
  - Fortran + Algol + COBOL + ...
- ¡C!
- ... ¿OO Cobol?

Soporte para datos en C/C++

- Los lenguajes de programación, hasta ahora, se centran alrededor de una estructura minimalista, y la funcionalidad se aporta mediante librerías, que incluso el mismo usuario puede construir.
- Lectura/escritura de texto: `fprintf( f, "%d", 5 )`
- Soporte para datos: `fwrite( &p, sizeof(p), 1, f )`
- Librerías de bases de datos:
  - Soporte del lenguaje.
  - Soporte del sistema operativo.

Trabajando con un vector de enteros

```
int main() {
   int x;
   FILE * f = fopen( “datos.dat”, “rt” );

   vector<int> v1;
   fread( f, “%d\n”, &x );
   do {
      v1.push_back( x );
      fread( f, “%d\n”, &x );
   while( !feof( f ) );
```

```
    // ... continúa ...
    fclose( f );
    for(int i = 0; i < v1.size(); ++i) {
        v[i] *= 2;
    }
    f = fopen( “datos.dat”, “wt” );
    for(int i = 0; i < v1.size(); ++i) {
        fprintf( f, “%d\n”, v[i] );
    }
    fclose( f );
}
```

¿Qué queríamos hacer?

```
// Lógica de negocio
for(int i = 0; i < v1.size(); ++i) {
      v[i] *= 2;
}
```

¿Por qué no serializar?

Serialización

- “Fácil” de utilizar.
- No hay metadatos.
- Dependiente de la arquitectura de la plataforma.
  - Ancho de palabra
  - Byte ordering
- ¿Dependiente del compilador?
  - Padding
  - vtable

dBase III/Clipper

- Ashton Tate publica dBase II, 1982, un potente gestor de base de datos relacionales programable mediante un lenguaje de script.
- Nantucket crea Clipper en 1985, un compilador 100% compatible con ese mismo lenguaje, de manera que no es necesario tener dBase III para crear/ejecutar los programas. El lenguaje es extensible mediante librerías compiladas en C.
- 1997: Clipper Visual Objects.

Clipper

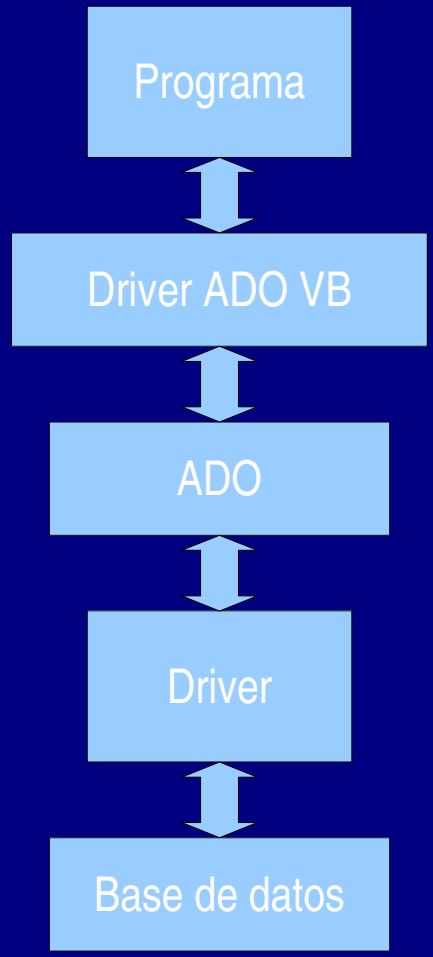
- Manejo de bases de datos “embebido”:

```
print “Convirtiendo nombres ...”
use PERSONA
replace PERSONA->NOMBRE
        with upper( PERSONA->NOMBRE )
        for PERSONA->NOMBRE = “A”
```

Delphi/C++ Builder y Visual Basic

- Lenguaje de programación estándar C++ [Pascal], al que se le añaden librerías que permiten el acceso a bases de datos relacionales.
- Visual Basic: confía en ADO, una librería integrada con el sistema operativo Windows, para el manejo de bases de datos.

ADO




```
"select * from Persona
          where edad>18"
```

Parsing de la sentencia SQL a comandos

C#.NET

C# 2.0

- Proporciona ADO.NET
- También proporciona, mediante librería, de un completo parser XML con interacción de XSLT y XPath.
- Como Java, permite la posibilidad de serializar un objeto.
- Soporta introspección.

C# 3.0

- Soporta Linq, un sublenguaje de manejo de datos embebido en C# 2.0
  - Linq: Objetos derivados de Collection
  - DLinq: Datos que provienen de bases de datos relacionales.
  - XLinq: XML

C# y Linq

```
public static void ejLinq() {
    int[] nums = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    var lowNums =
        from n in numbers where n < 5 select n
    ;

    Console.WriteLine( "Numbers < 5:" );
    foreach (var x in lowNums) {
        Console.WriteLine( x );
    }
}
```

C# y Linq

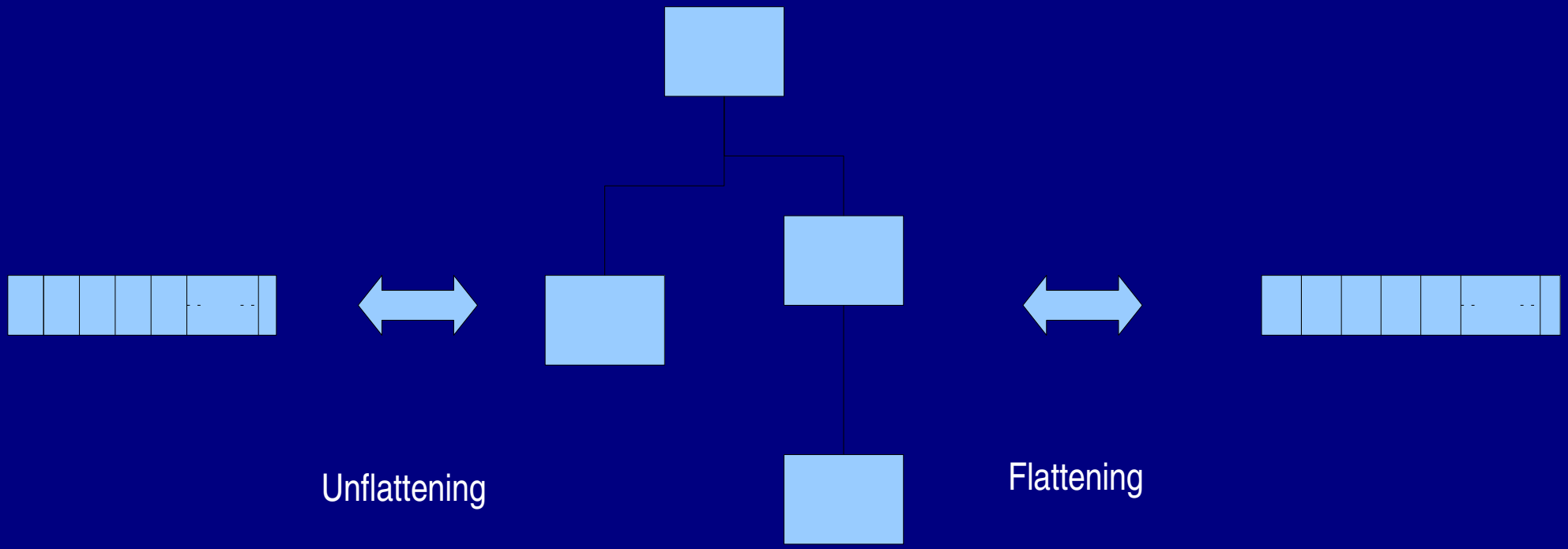
```
public static void ejLinq() {
    int[] nums = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    var lowNums = nums.Where(
         ( digit, index ) => digit < 5 )
    ;

    Console.WriteLine( "Numbers < 5:" );
    foreach (var x in lowNums) {
        Console.WriteLine( x );
    }
}
```

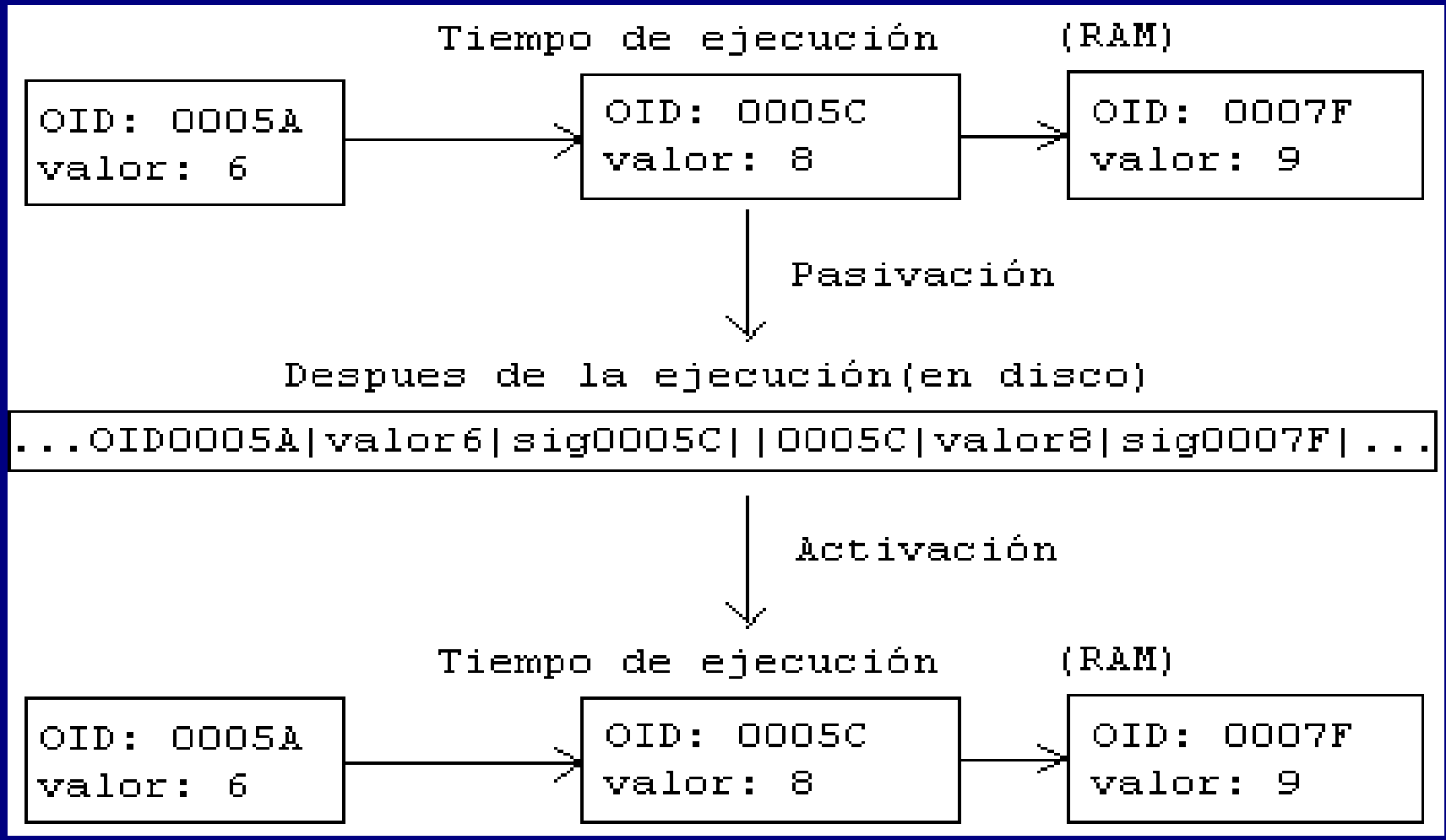
Demo C# 3.0

Persistencia

Modelo de ejecución



Cierre persistente y Swizzling



nHibernate

- Funciona contra ADO.NET
- Es necesario establecer en un fichero XML la asociación entre clases y tablas de la base de datos.
  - La configuración inicial puede ser bastante farragosa.
- Los objetos precisan de un constructor por defecto.
  - Puede no tener sentido.
- En resumen, no muy transparente, si bien es una gran herramienta.

db4Objects

- Guarda atributos de objetos en un formato propio: YAP (Yet another protocol).
- Soporta Java y C#.
- Si no existe la clase, devuelve un *GenericObject* (Map entre los miembros guardados y sus valores).
- Es necesario llamar a *set()* para almacenar un objeto, y a *commit()* para asegurar que estén guardados los últimos cambios.

db4Objects

- Los objetos deben tener un constructor por defecto.
  - Puede no ser una buena idea.
- Es mejor utilizar las dbList, en lugar de ArrayList y otros.
- Soporta ingeniosas formas de consulta de la base de datos de objetos.
  - Query by example.

db4Objects

- Creación de una base de datos de objetos:

```
ObjectContainer db = Db4o.openFile(
  “db.yap” );
Pilot pilot1 = new Pilot( "Michael
  Schumacher",100 );

db.set(pilot1);
```

db4Objects

- *Query by example*:

```
ObjectSet result =
        db.get( new Pilot( "Michael
      Schumacher",0 ) );
```

- Otros métodos de consulta:

```
IList<Pilot> pilots = db.Query<Pilots>(
    delegate(Student student)   {
      return student.Age < 20; } );
```

db4Objects Demo

Persistencia Ortogonal

Persistencia ortogonal

- Transparencia 100%: al tipo, al trato e identificación.
  - Un objeto debe poder ser persistente sin importar su clase.
  - Un objeto debe ser manejado de la misma forma, sea persistente o no.
  - La identificación de objetos persistentes se realiza por alcance desde una raiz persistente.

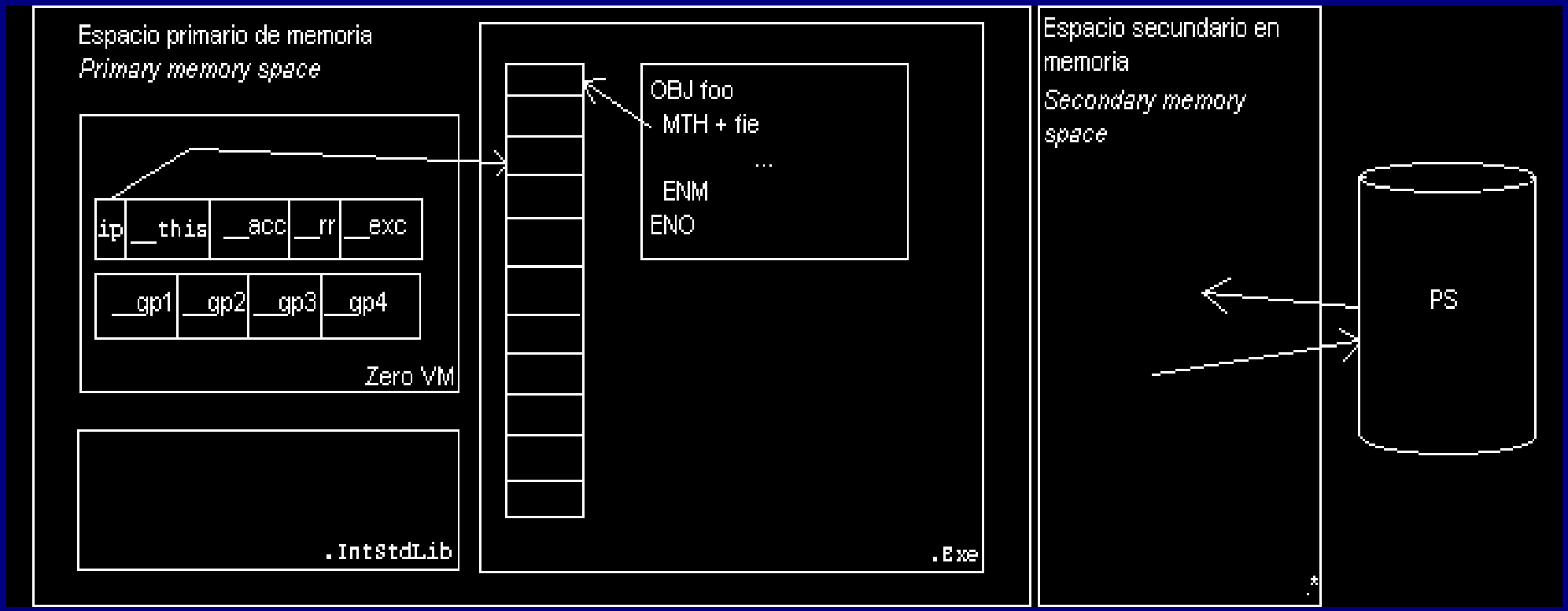
Persistencia Ortogonal

- Algunas desventajas:
  - No soporta cualquier lenguaje (debe ser seguro respecto al tipo).
  - Existe un conocido problema llamado *Spagghetti Pointers*.
  - ¿Organización de la base de datos de objetos?

Zero

Zero

- Máquina virtual orientada a objetos pura, persistente y basada en prototipos.



¡Hola, Mundo! en Zero

```
object HolaMundo : ConsoleApplication
   method + doIt()
      System.console.write("¡Hola, Mundo!")
      System.console.lf()
      return
   endMethod
endObject
```

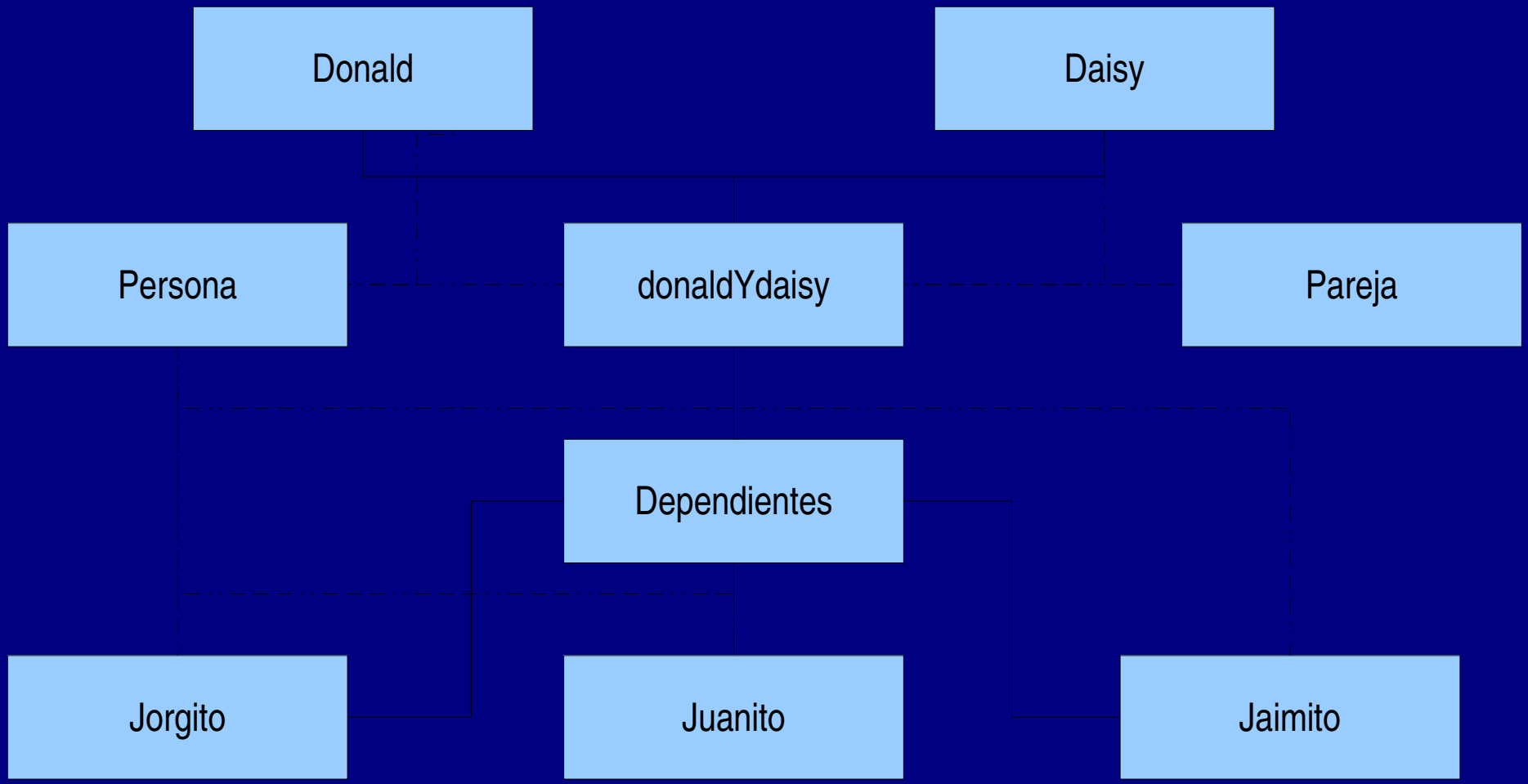
Persistencia transparente en Zero

```
object PersVector : ConsoleApplication
   method + doIt()
      reference x = VectorInstance.copy("")
      x.add( 5 )
      x.add( 7 )
      psRoot.addRenamedObject( "v1", x )
      return
   endMethod
endObject
```

Persistencia transparente en Zero

```
object VerPersVector : ConsoleApplication
   method + doIt()
      reference elem1 = psRoot.v1.get( 0 )
      System.console.write( psRoot.v1 )
      System.console.lf()
      System.console.write( elem1 )
      System.console.lf()
      return
   endMethod
endObject
```

Un ejemplo más complicado



Persistencia transparente en Zero

```
Disney.add( donaldYdaisy )

psRoot.add( Disney )
```

Demo Zero

Persistencia: evolución y .net

J. Baltasar García Perez-Schofield

Grupo IMO
Universidad de Vigo

http://webs.uvigo.es/jbgarcia/