E.T.S de ingenieros de Telecomunicacion

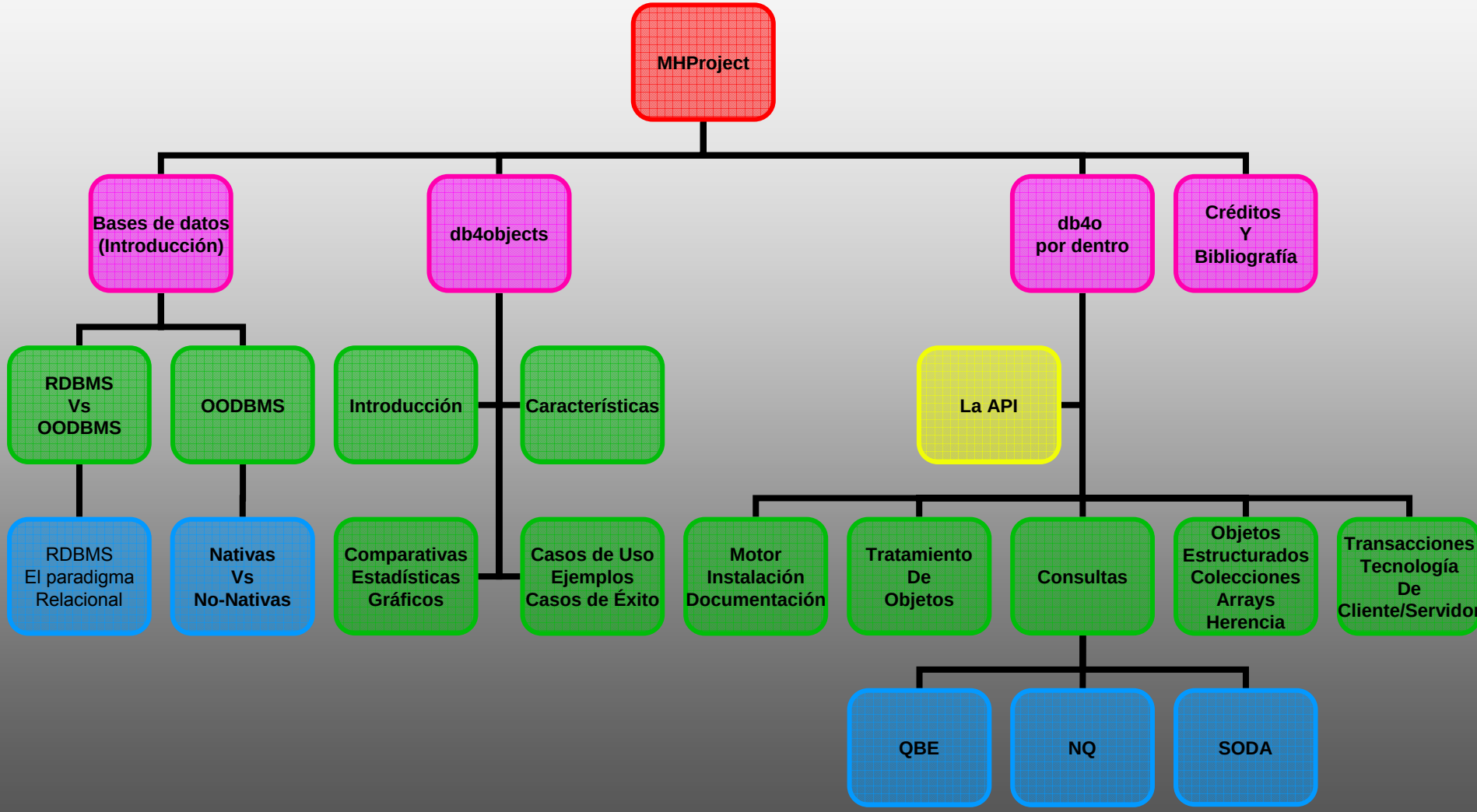
Digitally signed by Alejandro Fanjul
DN: cn=Alejandro Fanjul, c=ES, o=MHProject, ou=Proyectando, email=fanjul.35858@unavarra.es
'Date: 2005.12.21 21:46:21 +01'00

MHProject

db4objects

BASE DE DATOS ORIENTADA A OBJETOS

v1.0

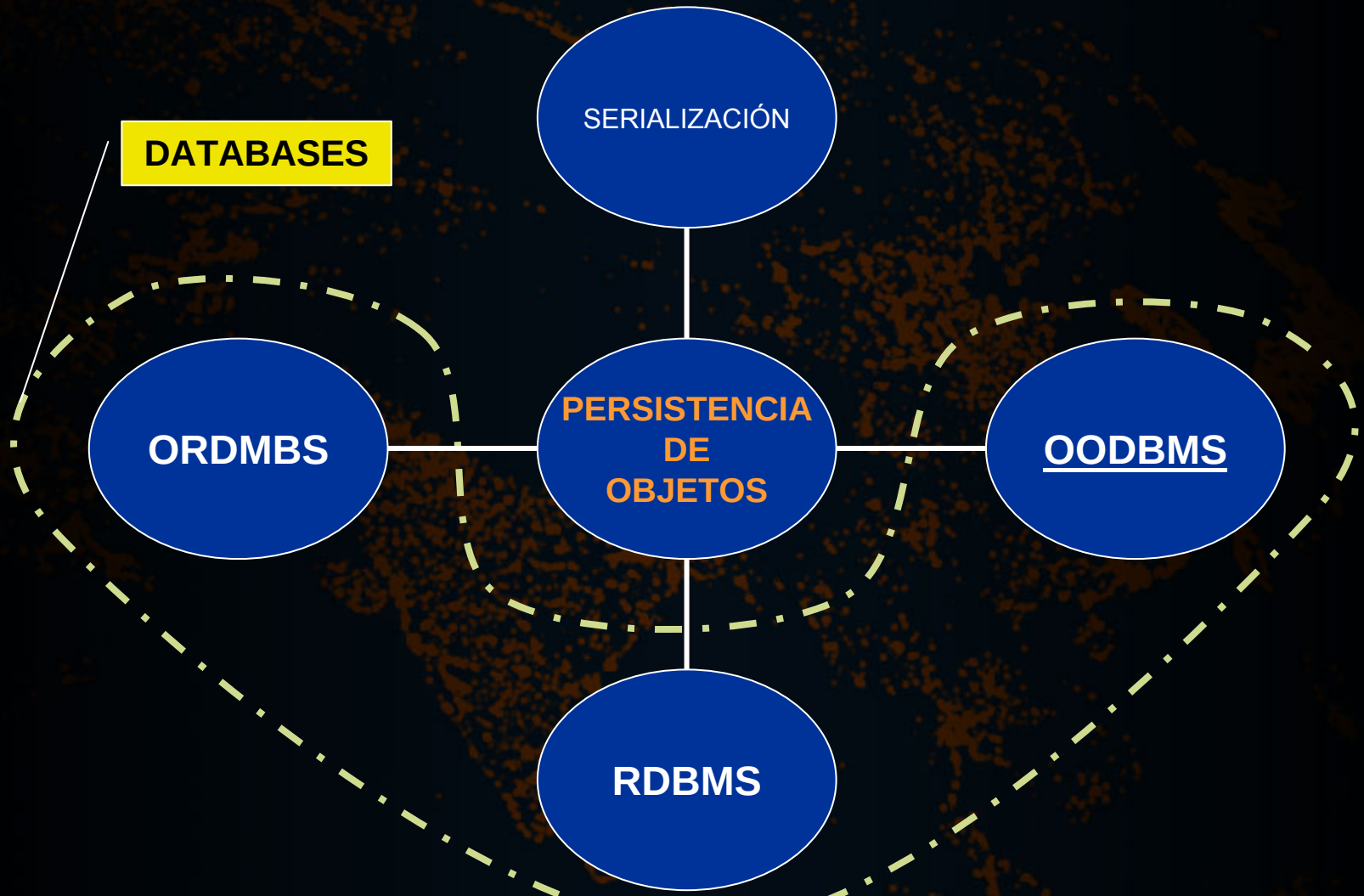

MHProject
Bases de datos (Introducción)
db4objects
db4o por dentro
Créditos Y Bibliografía
RDBMS Vs OODBMS
OODBMS
Introducción
Características
La API
RDBMS El paradigma Relacional
Nativas Vs No-Nativas
Comparativas Estadísticas Gráficos
Casos de Uso Ejemplos Casos de Éxito
Motor Instalación Documentación
Tratamiento De Objetos
Consultas
Objetos Estructurados Colecciones Arrays Herencia
Transacciones Tecnología De Cliente/Servido
QBE
NQ
SODA

BASES DE DATOS
INTRODUCCIÓN

PERSISTENCIA DE DATOS

Persistencia

- Una de las tareas más críticas en la programación es salvar y recuperar datos
- La **persistencia** es el almacenamiento de los datos en memoria, para una posterior recuperación de los mismos
- En sistemas orientados a objetos, existen varios métodos para hacer los objetos persistentes.
- La elección del método es de vital importancia.


DATABASES
SERIALIZACIÓN
ORDMBS
PERSISTENCIA DE OBJETOS
OODBMS
RDBMS

E.T.S de ingenieros de Telecomunicacion

BASES DE DATOS
INTRODUCCIÓN

RDBMS vs OODBMS

RDBMS vs OODBMS

- 2 Tecnologías cara a cara
  - ✓Tecnología Relacional
    - Orientada al uso de funciones
    - Centrada en los datos
  - ✓Tecnología de Objetos
    - Orientada a Objetos
    - Centrada en servicios

RDBMS vs OODBMS

- Tecnología Relacional
  - ✓Los datos constituyen un ente propio y van completamente separados de las funciones que los manejan.
  - ✓De hecho uno puede existir perfectamente sin la existencia del otro.
  - ✓Esto crea una complejidad añadida a la hora de manejar esos datos.

RDBMS vs OODBMS

- Tecnología de Objetos
  - ✓Los datos coexisten junto a los procesos que los tratan.
  - ✓Estas entidades son los **Objetos**
  - ✓Simplifica el tratamiento de los datos.

E.T.S de ingenieros de Telecomunicacion

BASES DE DATOS
INTRODUCCIÓN

RDBMS
EL PARADIGMA RELACIONAL

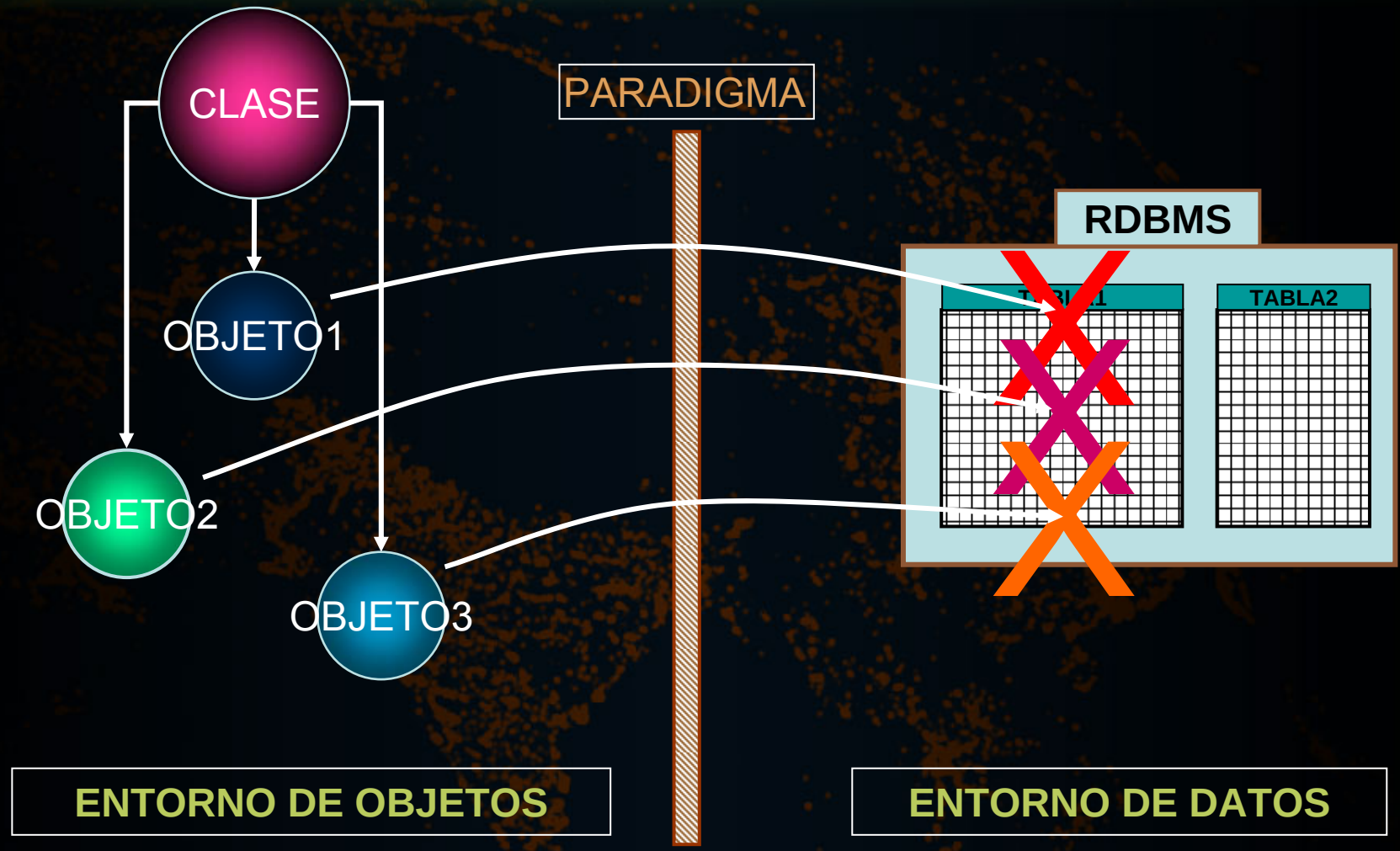
El Paradigma Relacional

- Los datos son independientes de las entidades que los procesan
- Dichos datos deben ser almacenados en bases de datos relacionales compuestas principalmente de Tablas, Filas (registros), y Campos.
- Dichos datos deben mantener una relación coherente entre ellos (Relación impuesta por la Base de datos).

El paradigma relacional

- ¿Qué pasa si queremos almacenar Objetos creados por un lenguaje orientado a Objetos en una base de datos Relacional?
- En este instante se produce lo que vamos a llamar la **Inadaptación de Impedancia** (Entre Objetos y Datos Relacionales)

Esquema del Paradigma Relacional

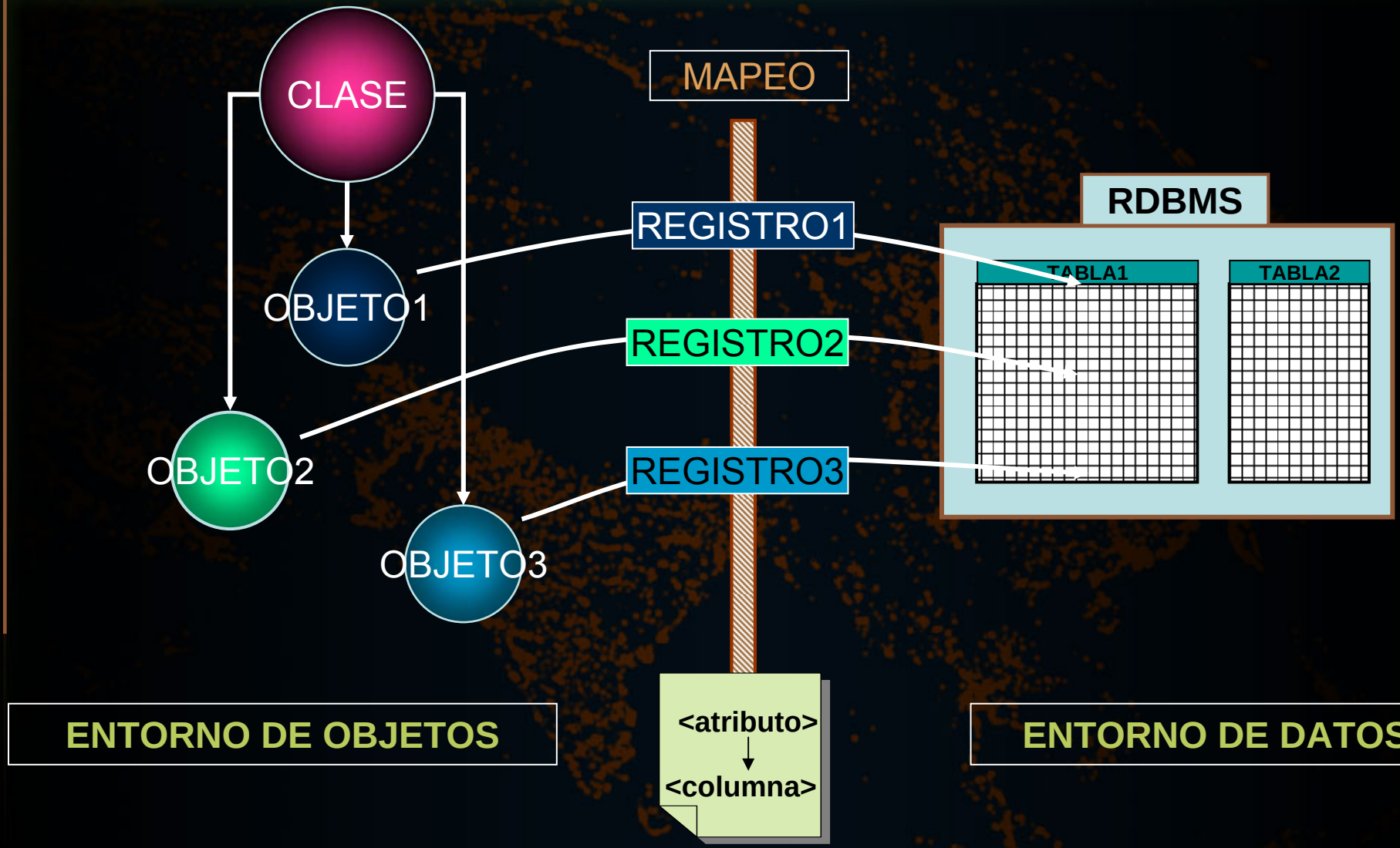


El paradigma relacional
Solución Adoptada

- ¿Qué pasa si queremos almacenar Objetos creados por un lenguaje orientado a Objetos en una base de datos Relacional?
- La solución comunmente aceptada pasa por lo que llamaremos

Mapeo Objeto - Relacional

Esquema del Paradigma Relacional (Solución)



Esquema del Paradigma Relacional (Solución)

- Mapeo (Objeto – Relacional)
  - ✓Nos permite mapear los objetos a registros en las tablas de base de datos.
  - ✓Esto se realiza mediante una hoja de mapeo en la que “convertimos” cada clase en una tabla, cada objeto en un registro (fila) de la base de datos y cada atributo en una columna de la misma.
  - ✓Tenemos que manejar las relaciones entre Clases (Tablas) mediante las típicas claves principales, primarias y/o externas.

mhproject

E.T.S de ingenieros de Telecomunicacion

BASES DE DATOS
INTRODUCCIÓN

OODBMS
(Object oriented Data Base Manager System)

OODBMS

- Nos permiten almacenar Objetos directamente (no hay registros ni tablas).
- Nos dan transparencia a la hora de almacenarlos.
- Nos ofrecen mayor flexibilidad ante los posibles cambios. a1
- Los datos coexisten con los objetos. a2
- Velocidad OODBMS ≥ RDBMS

OODBMS Standards

- **The Object Oriented Database Manifesto (1989)**

- **Mandatory features:**
- **Complex objects** (OO feature)
  - ✓ objects can contain attributes which are themselves objects.
- **Object identity** (OO)
- **Encapsulation** (OO)
- **Classes** (OO)
- **Inheritance** (OO): class hierarchies
- **Overriding, Overloading, Late Binding** (OO)
- **Computational completeness** (OO
- **Persistence** (DB)
  - ✓ data must remain after the process that created it has terminated
- **Secondary Storage Management** (DB)
- **Concurrency** (DB)
- **Recovery** (DB)
- **Ad hoc query facility** (DB)
  - ✓ not necessarily a query language – could be a graphical query tool

OODBMS Standards

- ***The ODMG Proposed Standard***
- One of the crucial factors in the commercial success of RDBMSs is the relative standardisation of the data model
- The Object Data Management Group (ODMG) was formed by a group of industry representatives to define a proposed standard for the object data model.
- It is still far from being as widely recognised as the relational database standards.
- The ODMG proposed standard defines the following aspects of an OODBMS:
  - ✓ basic terminology
  - ✓ data types
  - ✓ classes and inheritance
  - ✓ objects
  - ✓ collection objects (including sets, bags, lists, arrays)
  - ✓ structured objects (Date, Interval, Time, Timestamp – similar to SQL)
  - ✓ relationships
  - ✓ object definition language (ODL)
  - ✓ object query language (OQL)

OODBMS
Ventajas

- Objetos y Relaciones Complej@s
- Jerarquía de clases
- Sin desadaptación de impedancia
- Sin necesidad de claves primarias
- Un Modelo de datos
- Un Lenguaje de programación
- Sin necesidad de lenguaje de Conslutas
- Alto rendimiento en ciertas tareas

OODBMS
Desventajas

- Cambios de esquemas
- Falta de consenso en standards
- Falta de ad-hoc querying a7

  - ✓En General RDMBS es más adecuando para bases de datos con variedad de consultas y requerimientos de interface de usuario (ej. sistema de gestión de business). Mientras que OODBMS es más propio de aplicaciones con complejos y/o irregulares datos donde se siguen patrones previsibles (ej. CAD/CAM)

OODBMS
Java Data Objects (JDO)

- Aplicaciones escritas para uso de JDO pueden tratar con cualquier Base de datos que implemente JDO
- Las consultas (queries) están escritas en un lenguaje parecido-a-Java (JDOQL)
- El mapeo de objetos a la base de datos están definidos en descriptores XML
- Algunos proovedores de OODBMS basan sus productos en JDO (Lo hace db4o?)

OODBMS
Nativas vs. No-Nativas

- OODBMS No-Nativas
  - ✓ La interface nos permite un tratamiento transparente de los objetos como tales.
  - ✓ La base de datos se encarga de transformar esos objetos por nosotros.

- OODBMS Nativas
  - ✓ La interface permite el tratamiento nativo de los objetos como tales.
  - ✓ La base de datos almacena esos objetos como tales.

E.T.S de ingenieros de Telecomunicacion

db4OBJECTS

www.db4o.com

INTRODUCCIÓN

Db4o: Introducción

- Base de datos **orientada a objetos** completamente nativa de alto rendimiento.
- Desarrollada en el corazón de Sillicon Valley.
- Compacta y válida como Base de datos embedida en la aplicación.
- Soporta aplicaciones Standalone así como Cliente/Servidor (Aplicaciones distribuidas).
- Disponible para entornos Java o .Net

Db4o: Características

- Mínimo consumo de recursos
  - ✓Pequeña FootPrint (Huella)
- Alto rendimiento
- Fácil Implementación (Mínimo código)
- Portabilidad
  - ✓Corre sobre Java 1.x hasta 5.0, en .Net sobre J2EE, J2SE, J2ME:CDC, Symbian…
- Confiabilidad (ACID)
- Transparencia

Db4o: Características II

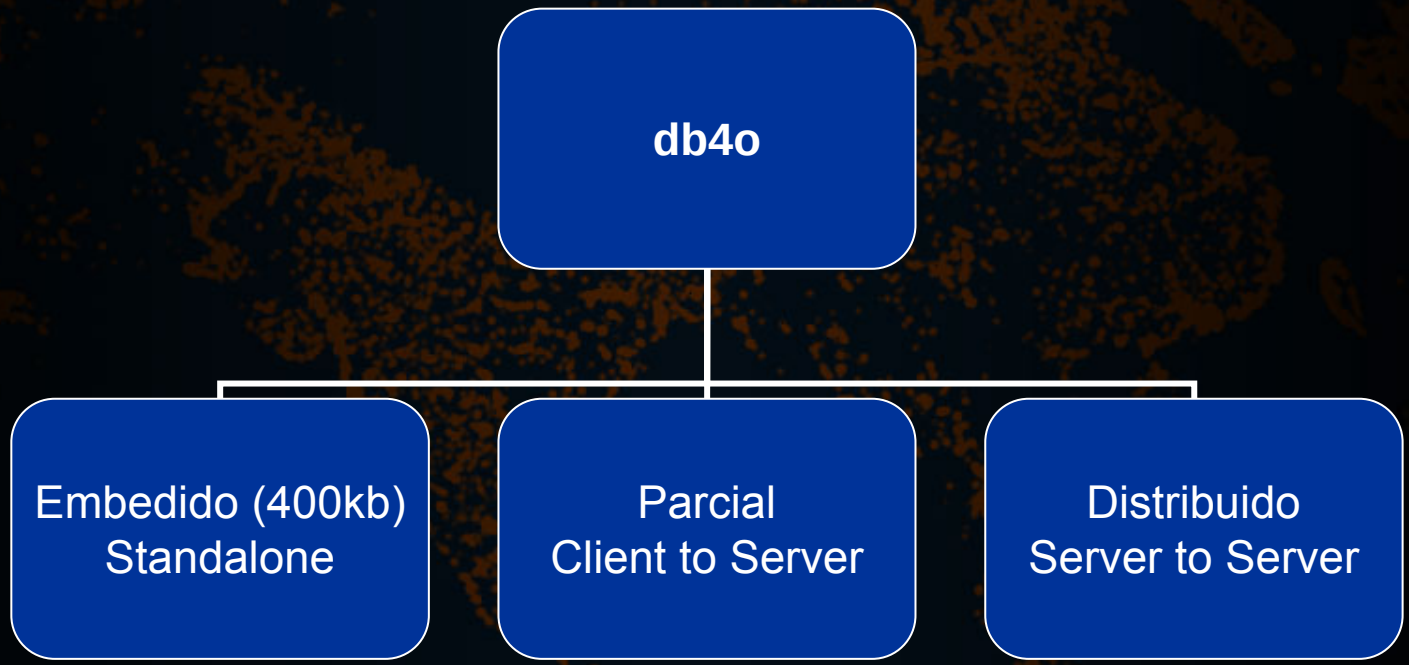
- Velocidad: hasta 44 veces más rápido que Hibernate/MySQL
- Soporte al cambio de Versiones
- Administración Nula
- Soporte a JSP / Servlets
- Código Abierto (Bajo 2 licencias)

Db4o: Características III

- Con db4o eliminamos el proceso de diseño, implementación y mantenimiento de la base de datos pues:
  - ✓El modelo de clases es el esquema de base de datos.

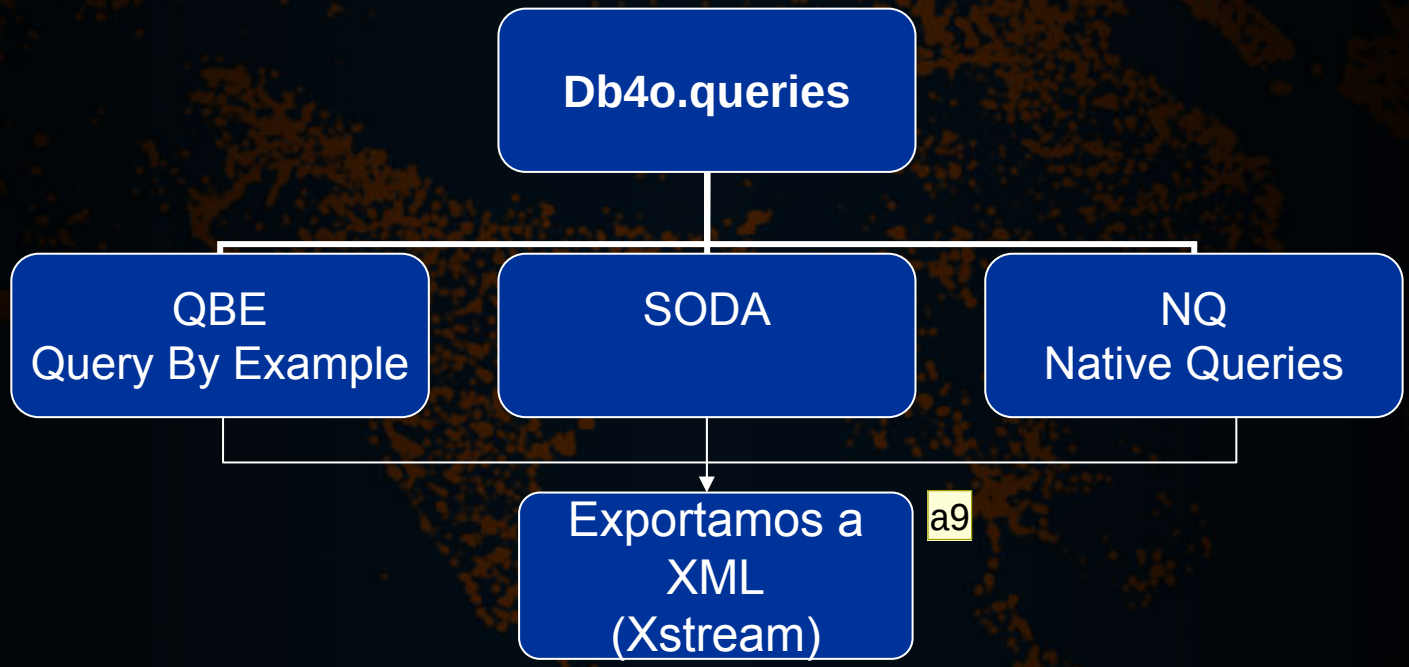
Db4o: Características III

- Db4o nos permite:
  - ✓ Embeder la base de datos (applicaciones Standalone)
  - ✓ Conexión parcial Cliente a Servidor
  - ✓ Conexión distribuida entre servidores (Sincronización de datos)



Db4o: Características IIII

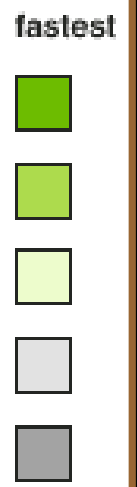
- Db4o nos ofrece 3 tipos de consultas:
  - ✓ Query by Example (QBE): Consultas por ejemplo o prototipo
  - ✓ Simple Object Database Access (SODA): Consultas dinámicas basadas en nodos
  - ✓ Native Queries (NQ): Consultas a datos con lenguaje nativo
    - Con todo lo que supone esto (TypeSafe, No-Strings, No Learn…)



Db4o: Comparativas, Estadísticas y Gráficos

Velocidad de escritura, lectura, consulta y borrado.

barcelona benchmarks	read	write	query	delete
db4o/4.5.200	1.0	1.0	1.0	1.0
Hibernate / MySQL	20.8	32.2	6.7	17.3
Hibernate / HSQLDB	10.4	5.4	536.0	3.9
JDBC / MySQL	10.8	14.6	1.7	6.5
JDBC / HSQLDB	0.4	1.7	677.8	0.7
JDBC / Derby	3,696.0	12.9	1,299.7	7.1
JDO/VOA/MySQL	4.4	14.8	3.0	2.4



Estadísticas realizadas con PolePosition.

Db4o: Comparativas, Estadísticas y Gráficos

Otros Bancos de Pruebas.

other benchmarks	melbourne	bahrain	sepang	imola
db4o / 4.5.200	1.0	1.0	1.0	1.0
Hibernate / MySQL	7.2	1.6	7.9	14.0
Hibernate / HSQLDB	2.8	0.5	2.2	4.8
JDBC / MySQL	3.1	0.7	2.3	5.9
JDBC / HSQLDB	0.4	0.1	0.8	0.1
JDBC / Derby	1.6	0.8	42.5	2.1
JDO / VOA / MySQL	3.4	0.9	4.1	10.2

fastest

slowest

Estadísticas realizadas con PolePosition.

Db4o: Casos de Uso

- Podemos utilizar db4objects para:
  - ✓Dispositivos móviles
    - (Móviles,Pda's,Tablet Pc's…)
  - ✓Dispositivos médicos y biotecnología
  - ✓Industria del transporte
  - ✓Software enlatado
  - ✓Aplicaciones Web (JSP / Servlets)
  - ✓Sistemas en tiempo real

Db4o: Casos de Uso II

- Podemos utilizar db4objects para:
  - ✓Instituciones educativas y de enseñanza en Colegios y/o Universidades.
    - (Conseguimos con ello centrarnos en el lenguaje Java o .Net sin distraernos en la BD)
  - ✓Juegos Standalone y Online
  - ✓Televsión Interactiva? (MHP) ?

Db4o: Ejemplos de Clientes

- Novell
- BMW Car IT
- Indra Systems
- Massie Labas
- Bosch
- TMT
- Eastern Data
- Electrabel
- Web Radiance
- Hertz

Db4objects
POR DENTRO

MOTOR
INSTALACION
DOCUMENTACION

Db4o por dentro
Motor, Instalación, Documentación

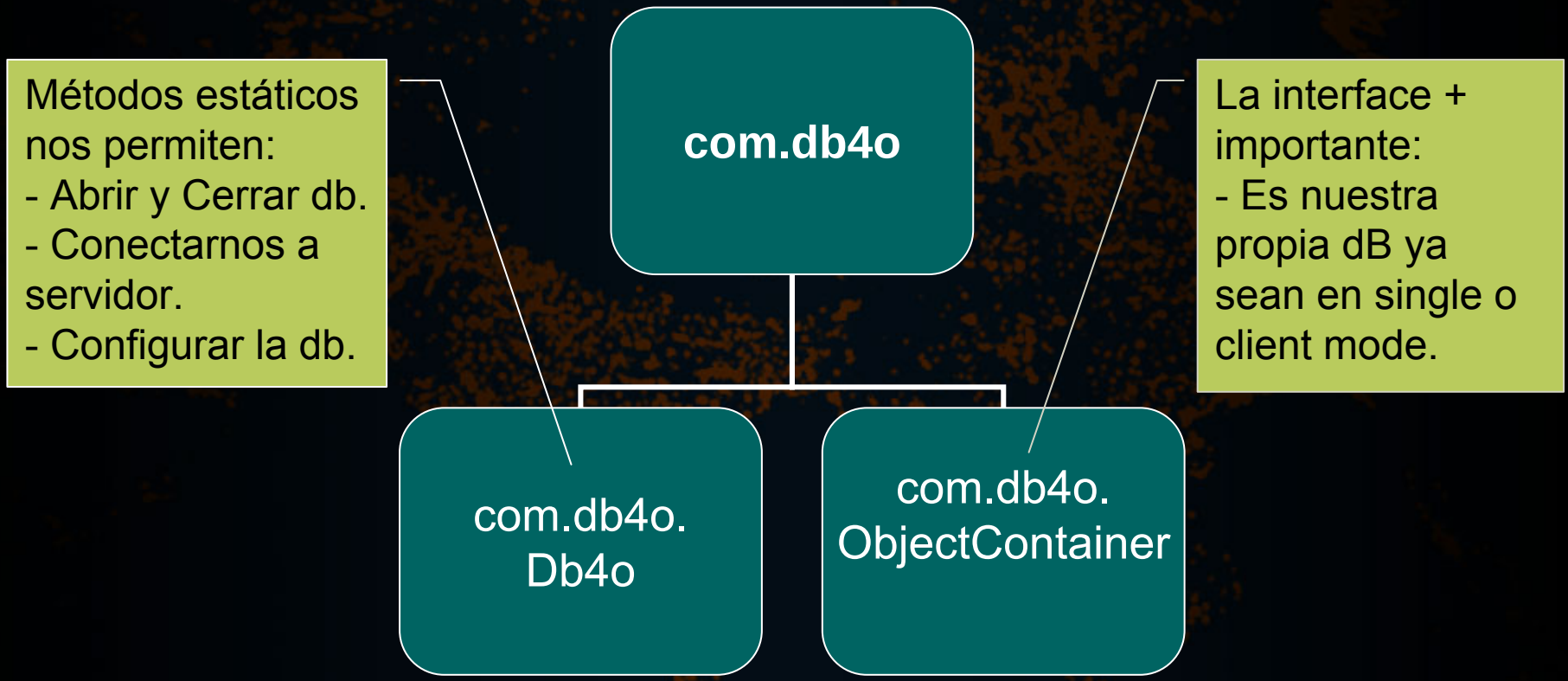
- El motor de la base de datos consiste tan solo en un pequeño archivo .jar de menos de 400kb.
- La instalación supone agregar el motor de la base de datos (db4o-.jar) a nuestro CLASSPATH.
- La documentación de la base de datos está creada mediante JavaDoc y viene incluida en el mismo paquete.

Db4o POR DENTRO

La API

Db4o Por dentro: La API

- En principio solo los paquetes com.db4o y com.db4o.query son necesarios para un correcto funcionamiento.

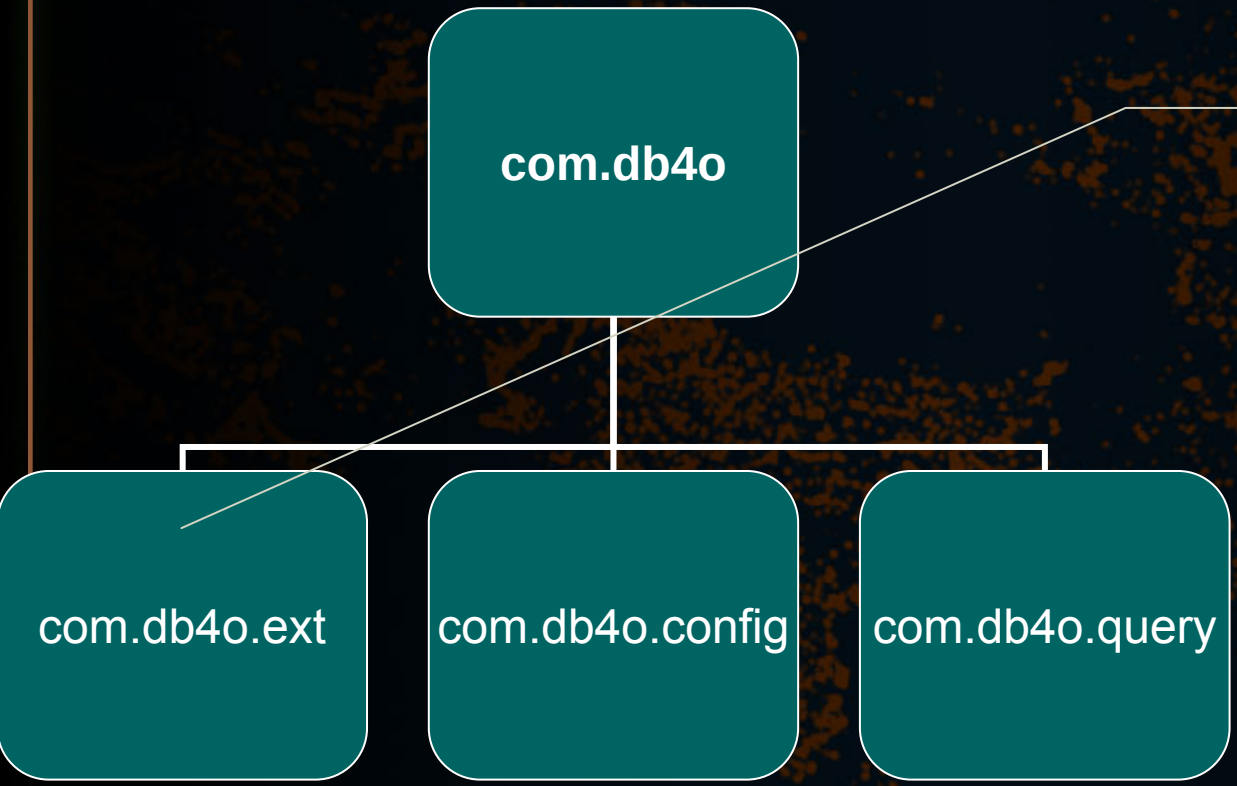


Db4o Por dentro: La API

- com.db4o.ObjectContainer.
  - ✓ Puede ser una dB tanto en single-mode como en client to server connection.
  - ✓ Todos los ObjectContainer manejan una transacción. Todo trabajo es transaccional.
  - ✓ Cada ObjectContainer mantiene sus propias referencias a objetos almacenados e instanciados.
  - ✓ Un ObjectContainer esta diseñado para permanecer abierto mientras se trabaja con ellos, cuando cerramos una dB todas las referencias a objetos desaparecen de la memoria RAM.

Db4o Por dentro: La API

- com.db4o.ext nos extiende y proporciona las funciones avanzadas de com.db4o. (2 pasos)

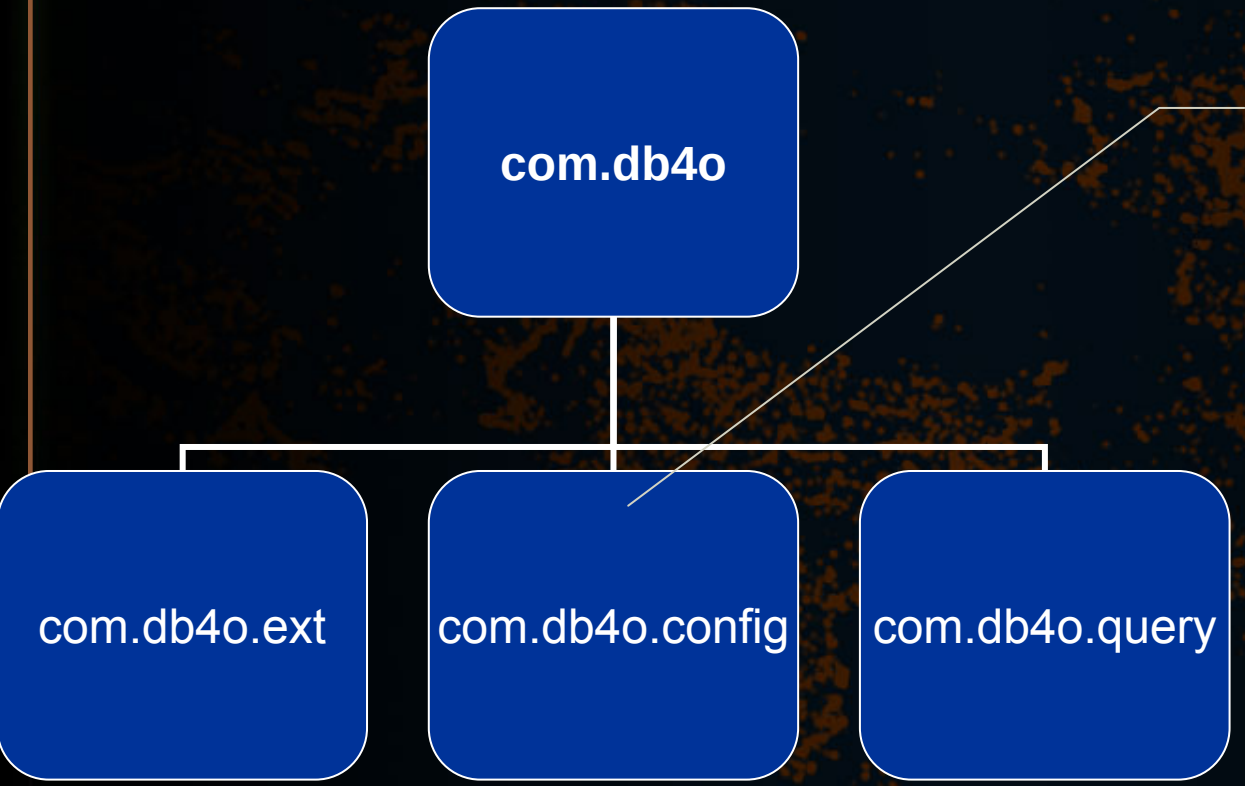


Razones:

-Es más fácil y rápido empezar a utilizar la db.
-Será más fácil para otros productos copiar la interface básica de la db.
-Es un sencillo ejemplo de cómo de ligera puede llegar a ser nuestra dB.

Db4o Por dentro: La API

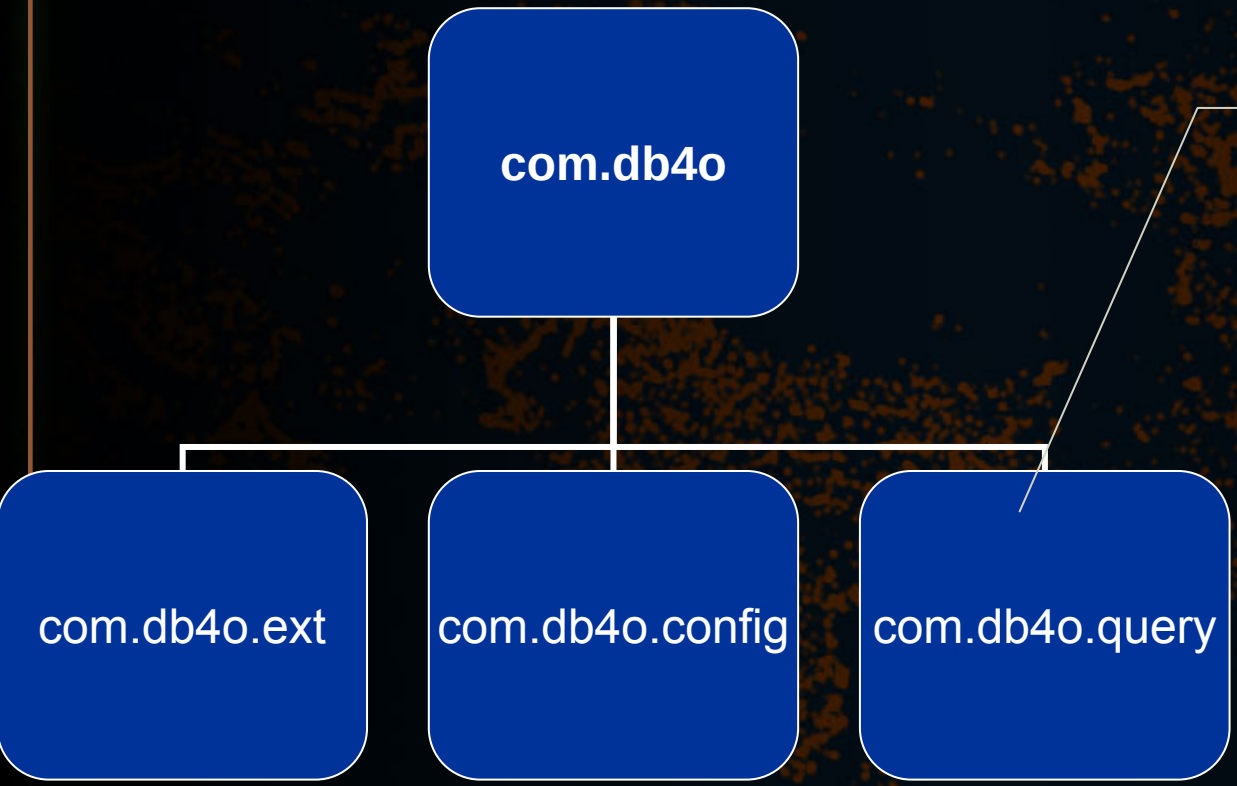
- com.db4o.config nos permite configurar y/o tunear la base de datos a nuestro gusto



-Paquete que contiene tipos y clases que nos permiten configurar y/o tunear la base de datos a nuestro gusto.
-La configuración de la base de datos se hace por norma general antes de abrir la sesión en la misma.

Db4o Por dentro: La API

- com.db4o.query nos ofrece el interface de consultas mediante 3 tipos de consultas



-Paquete que alberga la clase predicado para realizar consultas mediante Native Query (NQ).
-NQ permite mejor y más rápida implementación pero reduce el rendimiento de consultas.

E.T.S de ingenieros de Telecomunicacion

Db4o POR DENTRO

TRATAMIENTO DE OBJETOS

dbo4: Tratamiento de Objetos

- Abrir y cerrar la base de datos
  - ✓Simplemente llamamos a ***.openFile( )***
  - ✓Cerramos con una llamada a ***.close()***

```
ObjectContainer db = Db4o.openFile("Archivo.yap");
try {
                    // hacer algo con la db

finally {

   db.close();      // cerrar la base de datos antes de salir
}
```

Representa la base de datos

dbo4: Tratamiento de Objetos

- Almacenar objetos
  - ✓Simplemente llamamos a ***.set( )*** pasando cualquier objeto como parámetro

```
Pilot pilot1 = new Pilot("Michael Schumacher",100);

db.set(pilot1);


System.out.println("Stored "+pilot1);
```

Almacenamos el objeto con un simple set

dbo4: Tratamiento de Objetos

- Recuperar objetos
  - ✓Lo intentamos con la sencilla interface QBE

```
Pilot proto = new Pilot(null,0);

ObjectSet result=db.get(proto);

listResult(result);
```

Listamos todos ! Creamos un objeto prototipo con new() (con null's y 0's)

Consultamos con .get()

Listamos resultados

dbo4: Tratamiento de Objetos

- Recuperar objetos con QBE
  - ✓Lo intentamos con la sencilla interface QBE

```
Pilot proto = new Pilot("Michael Sumacher",0);

ObjectSet result=db.get(proto);

listResult(result);
```

Listamos por nombre
Creamos un objeto prototipo con new()

Consultamos con .get()

Listamos resultados

dbo4: Tratamiento de Objetos

- Actualizar objetos
  - ✓Lo intentamos con la sencilla interface QBE

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


found.addPoints(11);

db.set(found);

retrieveAllPilots(db);
```

Recuperamos un objeto de la db

Lo tratamos (Actualizamos)

Lo almacenamos En la misma sesión

dbo4: Tratamiento de Objetos

- Borrar objetos
  - ✓Simplemente llamamos a ***.delete( )*** pasando cualquier objeto como parámetro

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

db.delete(found);

retrieveAllPilots(db);
```

Borramos el objeto conocido

E.T.S de ingenieros de Telecomunicacion

Db4o POR DENTRO

CONSULTAS DE OBJETOS

dbo4: Consulta de Objetos

- Quering by Example (QBE) ¿Qué?
  - ✓ Son consultas extremadamente fáciles y rápidas
  - ✓ Tenemos limitaciones en las consultas
    - No podemos ejecutar consultas con expresiones avanzadas como (AND, OR, NOT, etc.)
  - ✓ No se pueden imponer valores 0 o null.
  - ✓ Necesitas un constructor para los objetos

dbo4: Consulta de Objetos

- Quering by Example (QBE) ¿Cómo?
  1. Creamos un **objeto prototipo** (Mediante atributos por defecto nulls y 0's)
  2. Utilizamos el método **.get( )** pasándole el objeto prototipo.
  3. Nos devuelve un ObjectSet que contiene los objetos de la consulta realizada

```
Pilot proto=new Pilot("null",0);
ObjectSet result=db.get(proto);
listResult(result);
```

Consultamos con .get()

dbo4: Consulta de Objetos

- Quering by Example (QBE) (Ejemplo 2)
  - ✓Hacemos una consulta por nombre

```
Pilot proto = new Pilot("Michael Sumacher",0);

ObjectSet result=db.get(proto);

listResult(result);
```

Listamos por nombre
Creamos un objeto prototipo con new()

Consultamos con .get()

Listamos resultados

dbo4: Consulta de Objetos

- Native Queries (NQ) ¿Qué?
  - ✓ Son consultas realizadas con el mismo lenguaje de programación.
  - ✓ Por tanto son consultas de Escritura-Segura, comprobadas en Tiempo de Compilación, y 100 % Factorizables
  - ✓ Podemos incluso llamar a métodos dentro de las propias consultas.
  - ✓ Están perfectamente standarizadas y son una vía de futuro seguro

dbo4: Consulta de Objetos

- Native Queries (NQ) ¿Cómo?
  1. Creamos un predicado (o expresión) con lenguaje de código nativo. [new predicate()]
  2. Implementamos el método #match() perteneciente a la clase predicate()
  3. Devuelve true para marcar específicas instancias como parte del conjunto resultado
  4. Utilizamos el método **.query( )** pasándole esa expresión de consulta.

dbo4: Consulta de Objetos

- Native Queries (NQ) (Ejemplo)
  - ✓Hacemos una consulta por puntos == 100

```
List pilots = db.query(new Predicate( ) {

    public boolean match(Pilot pilot) {

        return pilot.getPoints() == 100;

    }
});
```

Creamos una expresión en leguaje nativo con nuevo Predicate()

Implementamos el método .match() que hace la comparación y devuelve true

Consultamos esa expresión mediante .query()

dbo4: Consulta de Objetos

- Native Queries (NQ) (Ejemplo 2)
  - ✓ Hacemos una consulta por puntos > 99 (AND) <199

```
List pilots = db.query( new Predicate() {
   public boolean match(Pilot pilot) {

      return pilot.getPoints() > 99
      && pilot.getPoints() < 199
      || pilot.getName().equals("Rubens Barrichello");

         }
});
```

Creamos una expresión en leguaje nativo con nuevo Predicate()

Implementamos el método .match() que hace la comparación y devuelve true

Consultamos esa expresión mediante .query()

dbo4: Consulta de Objetos

- S.O.D.A Query API ¿Qué?
  - ✓ Son consultas de nodo dinámicas de bajo nivel que permiten directamente recorrer la jerarquía de clases.
  - ✓ Al contrario que NQ, SODA utiliza Strings para identificar los campos
  - ✓ Por lo tanto no tiene las características de Escritura-Segura, comprobadas en Tiempo de Compilación, y 100 % Factorizables
  - ✓ Sin embargo es más rápido que NQ
  - ✓ Al contrario que QBE permite consultas a 0.

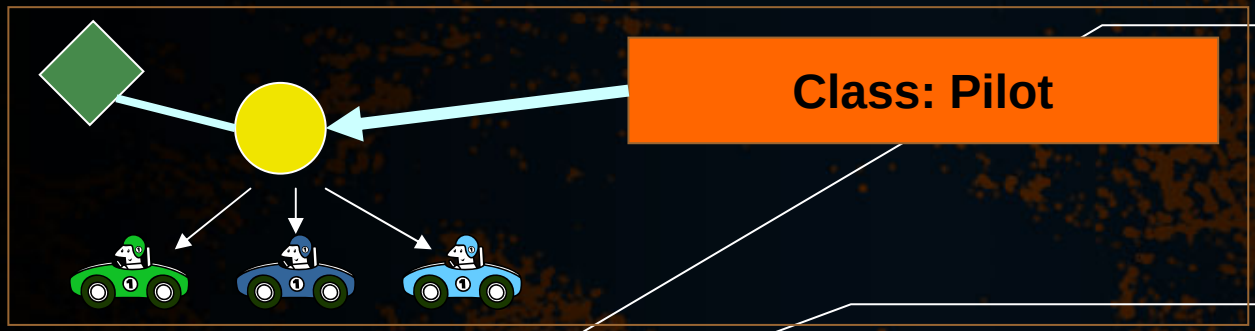
dbo4: Consulta de Objetos

- S.O.D.A Query API ¿Cómo?
  1. Creamo un nuevo objeto (nodo) a través del metodo .query() del ObjectContainer
  2. Descendemos por árbol jerárquico de clases
  3. Imponemos alguna o varias condiciones con .constrain(), (a uno o varios atributos)
  4. Ejecutamos esa consulta con .execute() y nos devuelve un ObjectSet con los objetos consultados.

dbo4: Consulta de Objetos

S.O.D.A Query API (Ejemplo)

- ✓Hacemos una consulta por todos los pilotos




```
Query query = db.query();

query.constrain(Pilot.class);

ObjectSet result = query.execute();

listResult(result);
```

Creamos nuevo nodo de consulta con **.query()**

Imponemos ese nodo a 'Toda la clase pilot' con **.constrain()** sobre Pilot.class

Ejecutamos la consulta mediante **.execute()**

dbo4: Consulta de Objetos

S.O.D.A Query API (Ejemplo 2)

- ✓Hacemos una consulta por nombre de piloto


```
Query query=db.query();

query.constrain(Pilot.class);

query.descend("name").constrain("MS");

ObjectSet result=query.execute();
```

Creamos nuevo nodo de consulta con **.query()**

Imponemos ese nodo a 'Toda la clase pilot

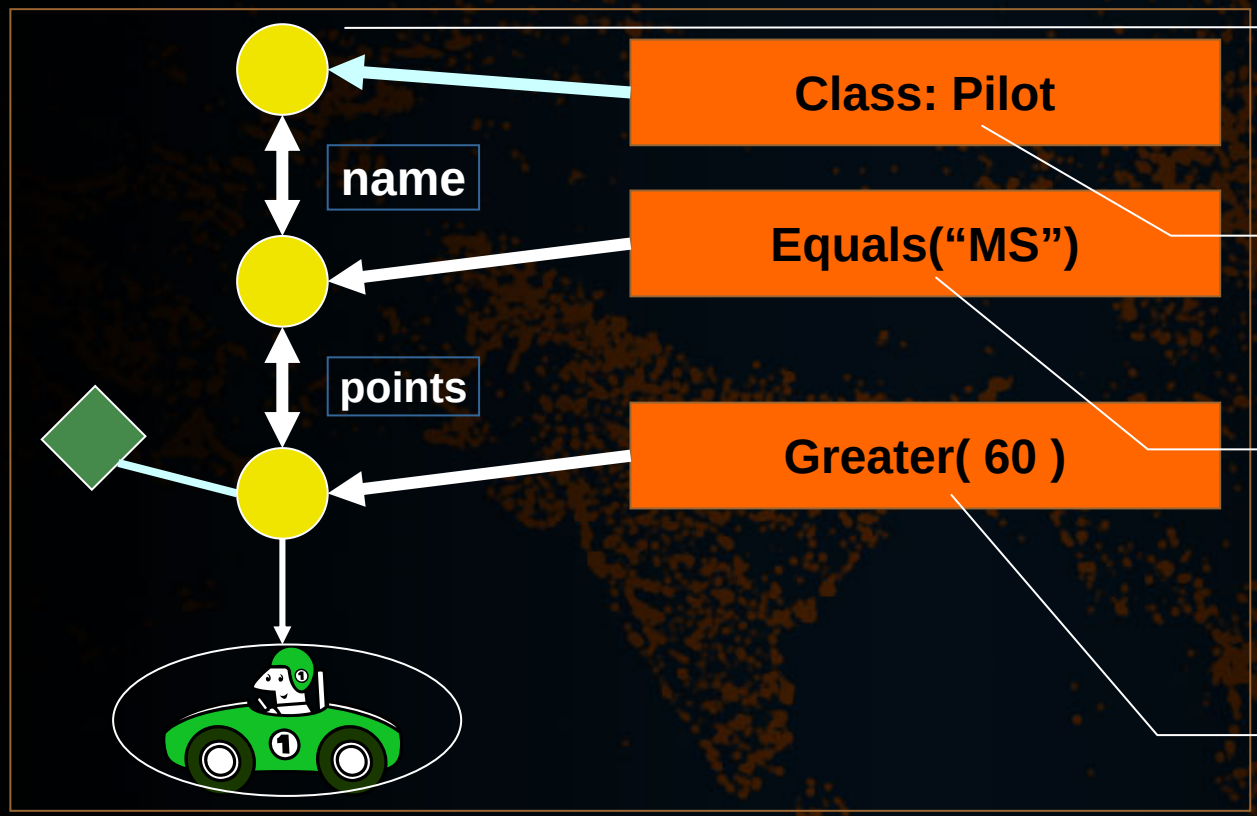
Descendemos por el nodo a **name** e imponemos "**MS**" con **.constrain()**

Ejecutamos la consulta mediante **.execute()**

dbo4: Consulta de Objetos

S.O.D.A Query API (Ejemplo 2)

- ✓Hacemos una consulta por nombre y puntos



Creamos nuevo nodo de consulta con **.query()**

Imponemos ese nodo a 'Toda la clase pilot

Descendemos por el nodo a **name** e imponemos "**MS**" con **.constrain()**

Descendemos a points e imponemos **>60**

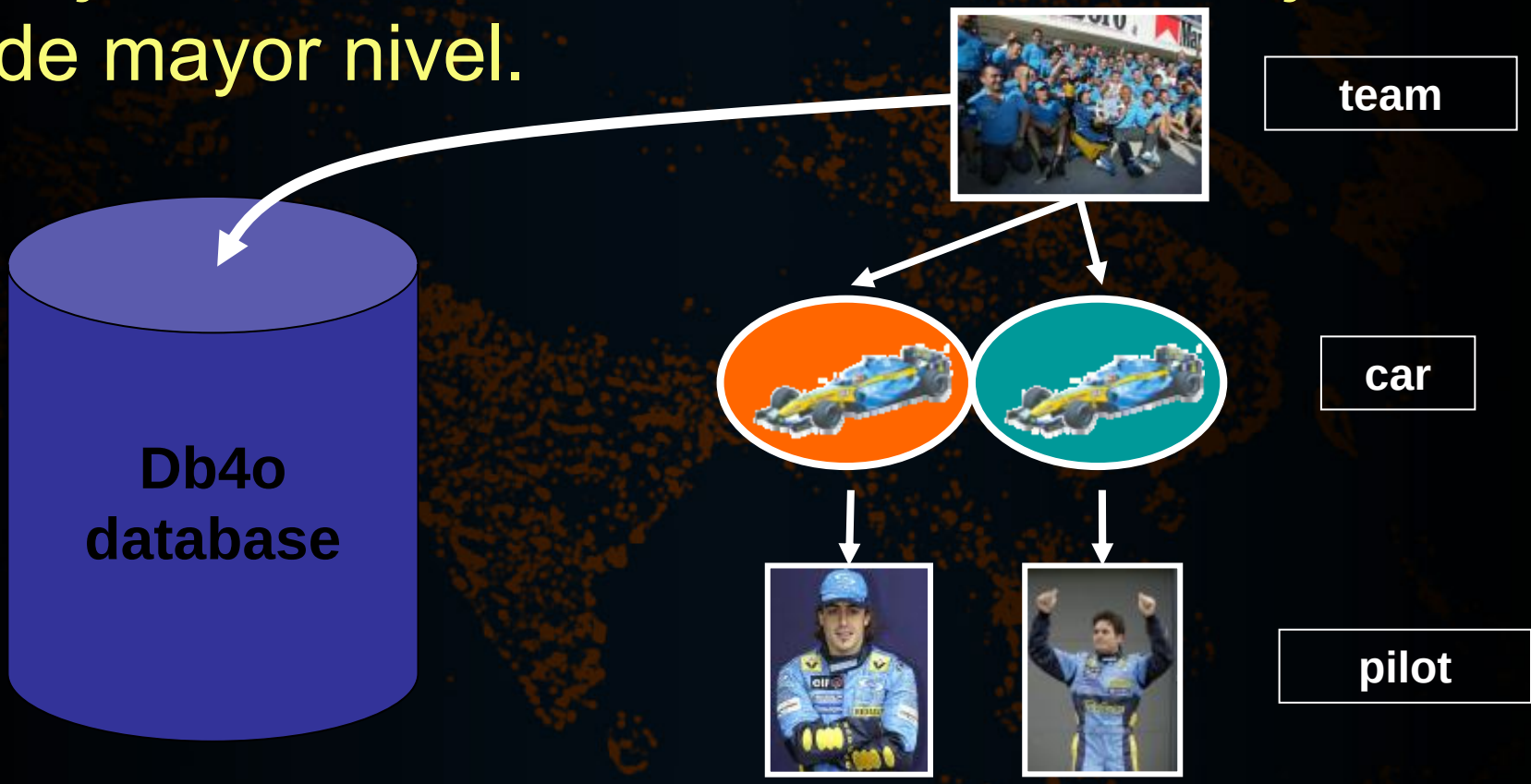
E.T.S de ingenieros de Telecomunicacion

Db4o POR DENTRO

Objetos Estructurados
Colecciones y Arrays
Herencia

dbo4: Objetos Estructurados

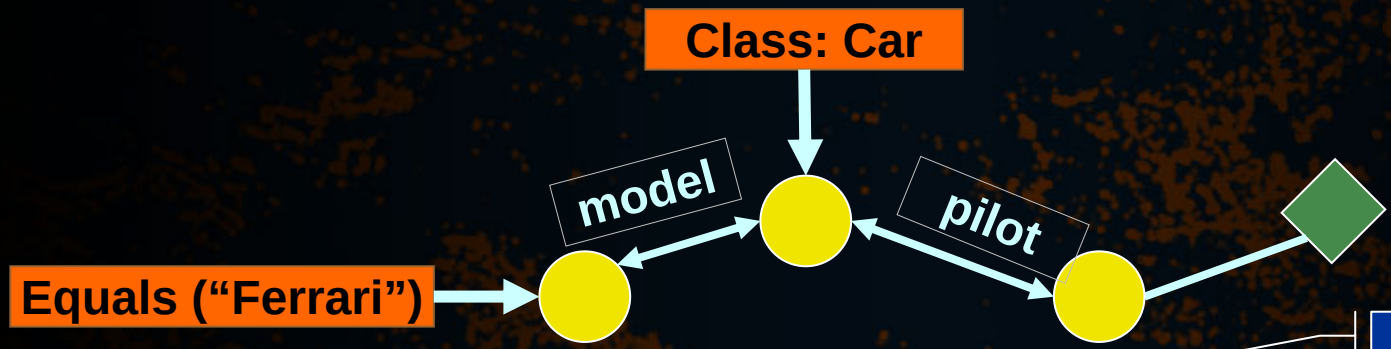
- No tenemos por qué almacenar todos los objetos subordinados si no sólo el objeto de mayor nivel.



dbo4: Objetos Estructurados

Consultas estructuradas

- ✓Consultar un piloto por su modelo de coche




```
Query carquery=db.query();

carquery.constrain(Car.class);
carquery.descend("model").constrain("Ferrari");

Query pilotquery=carquery.descend("pilot");
ObjectSet result=pilotquery.execute();
```

Creamos nuevo nodo de consulta con **.query()**

Descendemos por model imponemos Ferrari

Descendemos por pilot.

dbo4: Objetos Estructurados

- Actualización **de profundidad**
  - ✓Si queremos actualizar un objeto subordinado salvando (almacenando) su objeto superior en la dB debemos configurar la db para actualización de profundida con .**cascadeOnUpdate()** antes de abrir la db.
  - ✓Por defecto la profundidad es 1 lo cual permite actualizar atributos primitivos o Strings.

dbo4: Objetos Estructurados

- Borrado **recursivo**
  - ✓ De forma análoga a la actualización el borrado recursivo debe ser configurado mediante .cascadeOnDelete() antes de la apertura de la dB.
  - ✓ Con esto conseguimos borrar los objetos subordinados borrando únicamente el objeto de mayor nivel.
  - ✓ Pero que pasa si un objeto subordinado está referenciado por otro objeto? → Cuidado!!!

dbo4: Objetos Estructurados

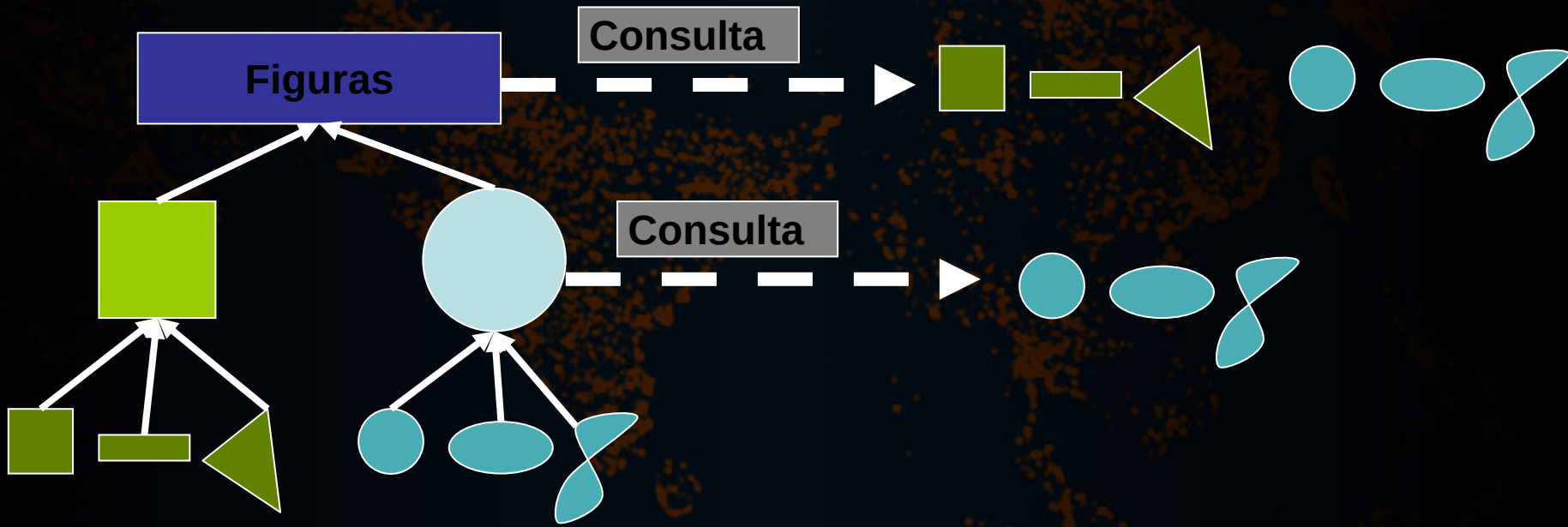
- **Sin .cascadeOnDelete()**
- **Con** .cascadeOnDelete()

db4o: Colecciones y Arrays

- Podemos consultar arrays y almacenarlos de la misma forma que datos primitivos.
  - ✓Mediante QBE el orden de los valores consultados es irrelevante
  - ✓Con NQ simplemente consultamos como si hiciéramos búsquedas nativas en arrays.
- Nada nuevo en consulta y borrado de arrays, simplemente tener en cuenta la profundidad.

db4o: Herencia

- Db4o nos devuelve los objetos del tipo consultado es decir:
  - ✓ Consultando una SuperClase nos devuelve todos los objetos padre e hijos.
  - ✓ Consultando una SubClase nos devuelve los objetos de esa SubClase.



db4o: Herencia

- ¿Qué pasa con QBE si la clase a consultar es Abstracta o Interface?
  - ✓No podemos utilizar un constructor para hacer el objeto prototipo.
  - ✓Solución: Utilizamos 'MiClase'**.class** lo que nos devuelve la clase entera.

E.T.S de ingenieros de Telecomunicacion

Db4o POR DENTRO

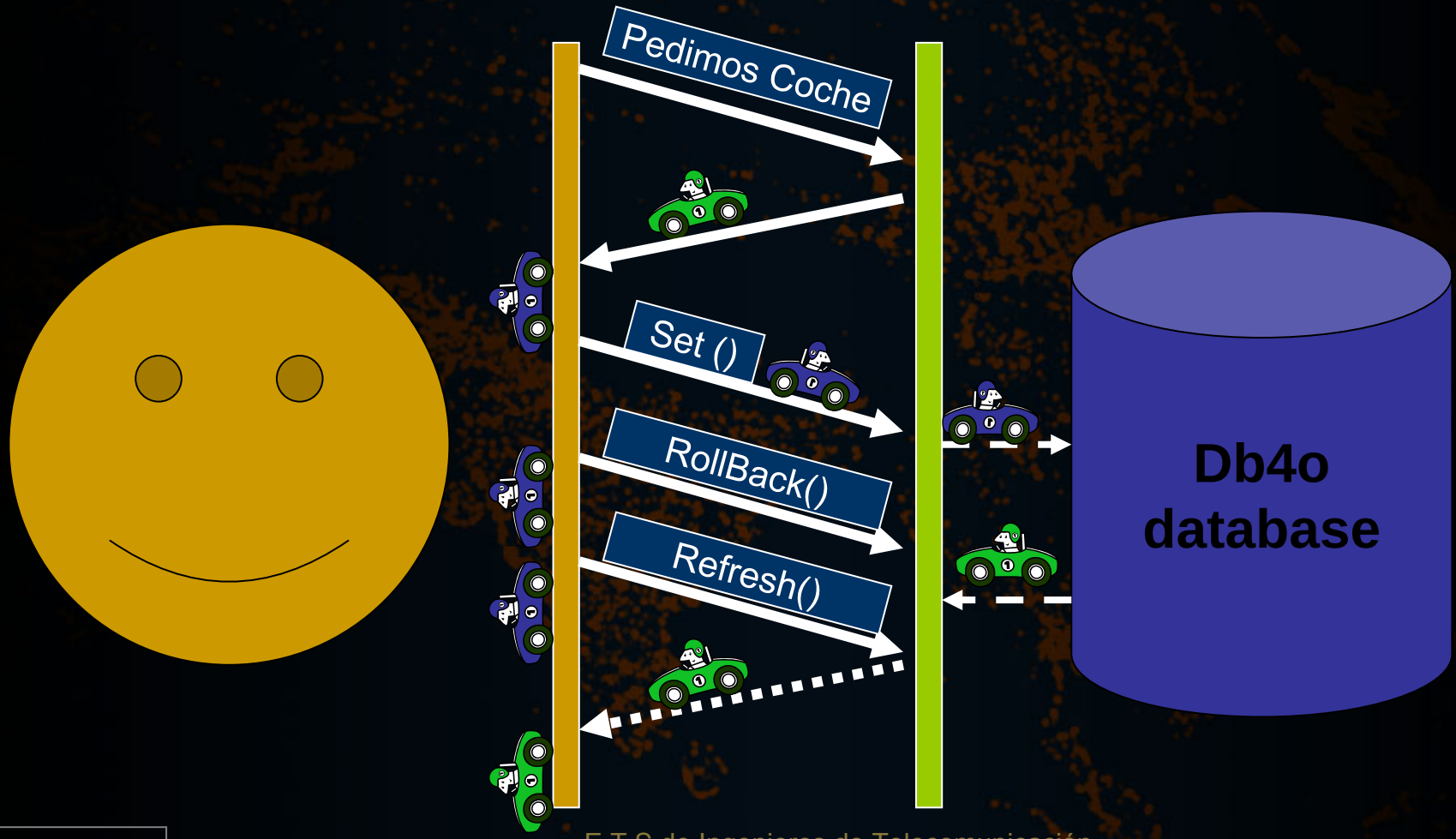
TRANSACCIONES
TECNOLOGÍA
CLIENTE/SERVIDOR

db4o: Transacciones Simples

- db4o nos ofrece dos métodos para realizar transacciones
  - ✓.commit() Finaliza una transacción
  - ✓.rollback() Deshace una transacción
- La transacción es implícitamente cerrada por defecto cuando cerramos una db.
- Sin embargo debemos tener cuidado a la hora de hacer .rollback() con Live Objects.
  - ✓Se deben refrescar con .ext().refresh(obj,depth)

db4o: Transacciones Simples

- Proceso commit – rollback (Client/Server)



db4o: Cliente / Servidor Local

- De cara a la API no existen diferencias reales entre ejecutar transacciones concurrentes dentro de la misma VM y transacciones ejecutadas contra un server
  - ✓Para abrir una db en modo servidor dentro de la misma VM utilizamos **.openServer('dbfile.yap',0)** con puerto 0.
  - ✓Para acceder a esa base de datos en modo cliente Local tan solo usamos **.openClient()**

db4o: Cliente / Servidor Local

- Cada contenedor cliente mantiene su propio cache de referencias débiles de los objetos ‘ya conocidos’.
- Para hacer que los cambios hechos por un cliente sean ‘cometidos’ por todos los clientes inmediatamente debemos refrescar los objetos conocidos explícitamente con **.ext().refresh(obj, depth)**

db4o: Cliente / Servidor Remoto

- Desde aquí hay un pequeño paso ya para configurar nuestra db como Client/Server sobre TCP/IP.
  - ✓Tan solo especificamos un puerto de servidor mayor que 0 y abrimos los clientes garantizandoles acceso con USER Y PASS.
  - ✓Abrimos con **.openServer(db.yap,PORT)**
  - ✓Garantizamos con **.grantAccess(USER,PASS)**
  - ✓Conectamos con **.openClient(“Server",PORT,USER,PASS)**

db4o: Cliente / Servidor Remoto

- Algunas veces necesitamos enviar ciertos mensajes tipo ‘kill’ al servidor entonces:
  - ✓El servidor se pone como receptor de mensajes con .**setMessageRecipient()** pasándole como parámetro el mensaje obj.
  - ✓El mensaje es recivido y procesado por el método **processMessage()**
  - ✓El cliente se pone como Mensajero con .getMessageSender() y manda un mensaje con **.send() . Ejemplo msg: new StopServer()**

db4o: Conclusión

- Hemos visto como db4o nos permite hacer un sin fin de cosas con muy poco código y esfuerzo y eso que tan solo hemos visto una parte de db4o. Aún nos quedaría por mirar temas como:
  - Evaluaciones SODA
  - Constructores
  - Intérpretes
  - Configuración y Tuneado
  - Indexación
  - Object Manager
  - ClassLoaders
  - ServLets
  - Encriptación
  - Refactorizado de clases
  - Optimización de consultas
  - Replicaciones

E.T.S de ingenieros de Telecomunicacion

Créditos y Bibliografía

db4o – Base de datos Orientada a Objetos

[---------- Proyecto ----------]

MHProject v1.0

www.mhproject.org

E.T.S de Ingenieros de Telecomunicación
Universidad Pública de Navarra

[---------- Autor ----------]

Alejandro Fanjul

fanjul.35858@e.unavarra.es

afanjul@mhproject.org

[---------- Tutores ----------]

Mikel Sagues

mikel.sagues@unavarra.es

Javier Navallas

javier.navallas@unavarra.es

[---------- Bibliografía ----------]

Db4o.com: Tutorial and Presentations

Bell College: Presentations [Object Persistence]