Aplicações em Redes de Grande Escala

Persistência

Prof. Paulo Ferreira
IST/INESC

http://mega.ist.utl.pt/~ic-arge

Índice

- Modelo de persistência (por acessibilidade, explícita, etc.)
- Falta de objecto (que está em disco)
- Transparência
- *Persistent State Service* (arquitectura, PDL)
- Suporte de persistência (ficheiros, bases de dados)
- Serialização (ex.: CORBA, Java, .Net)
- Gestão de versões dos dados serializados
- Acesso a bases de dados (ex.: JDO, ADO)
- Mapeamento objecto-relacional (ex.: SQL)
- Transacções (ex. CORBA, Java, .Net)

Introdução

- **Capacidade do sistema suportar a existência de objectos independentemente do tempo de vida dos processos que os utilizam:**
  - Escrita/leitura para disco (serialização)
  - Activação (falta de objecto) e desactivação dos objectos
- **Permite que seja guardado estado de forma perene:**
  - Partilha ao longo do tempo
  - Quando combinada com sistema transacional permite maior robustez

Modelo de Persistência

- **Quais os objectos que são guardados de forma persistente**
- **Explícita:**
  - O programador decide quais os objectos que devem ser guardados de forma persistente indicando esse facto no seu programa
  - Não é transparente para o programador
- **Tipos:**
  - Os objectos que são guardados de forma persistente devem herdar de uma interface/classe específica
  - Não é transparente para o programador
- **Acessibilidade:**
  - Os objectos que são guardados de forma persistente são aqueles que são acessíveis directa ou indirectamente a partir de uma raiz de persistência
  - É transparente para o programador

Falta de Objecto

- **Explícita:**
  - Cabe ao programador obter o objecto antes de o invocar evitando assim a falta do objecto
  - Cabe ao programador tratar a excepção que resulta da invocação de um objecto não existente em memória principal
- **Implícita:**
  - Cabe ao sistema detectar que o objecto invocado não está em memória principal e carregá-lo em antecipação
  - Cabe ao sistema tratar a excepção que resulta da invocação de um objecto não existente em memória principal

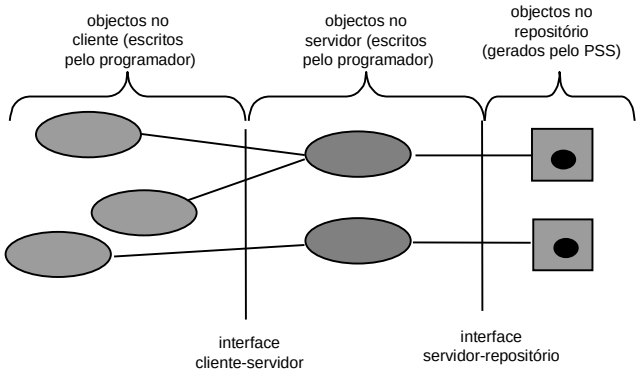
Transparência de Persistência

- **Transparência:**
  - Quais os objectos que devem ser guardados de forma persistente:
    - Para o programador dos servidores pode não ser um requisito
    - Para o programador dos clientes deve ser um requisito
  - Qual o suporte usado para guardar os objectos de forma persistente:
    - Para o programador dos servidores deve ser um requisito
    - Para o programador dos clientes deve ser um requisito
- **Claramente, a aplicação deve usar a mesma API independentemente do:**
  - Suporte usado para guardar os objectos (data storage-type transparency e data storage-vendor transparency)

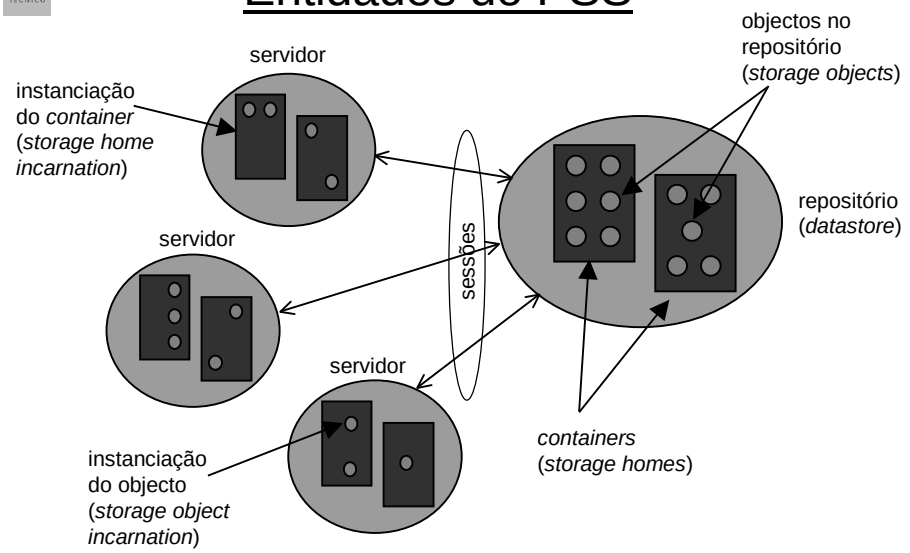
Transparência de Persistência (cont.)

- **Suporte usado para guardar os objectos de forma persistente:**
  - Transparência assegurada através de Persistente State Services (PSS)
  - PSS são usados através de um Persistente State Service Definition Language (PSSDL)
  - PSSDL permite especificar quais os objectos (ou parte deles) que devem ser guardados de forma persistente e como voltar a mapear em memória esses mesmos objectos
  - PSS efectua a serialização de forma automática
  - PSS permite automatizar, tanto quanto possível, o mapeamento entre os objectos no servidor e no repositório (ficheiros, bases de dados relacionais, bases de dados orientadas a objectos)

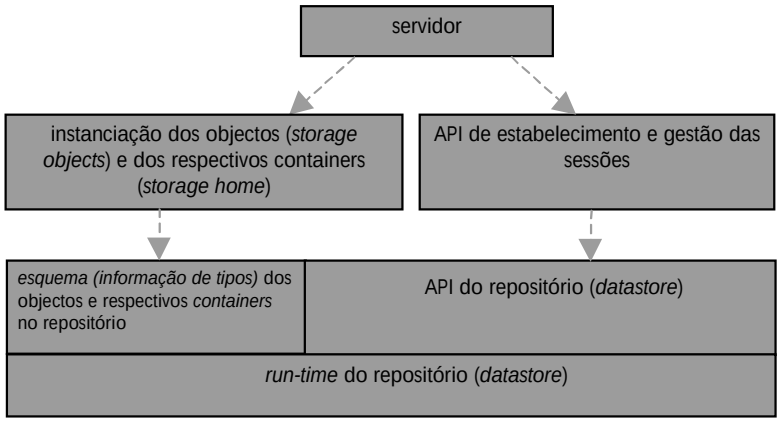
Persistence State Service



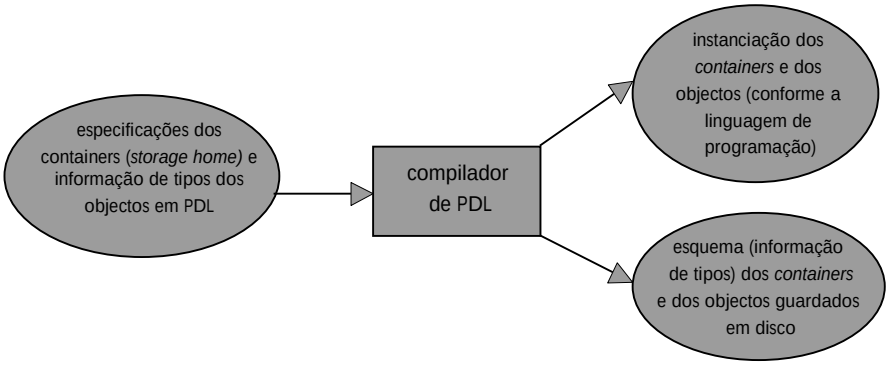
Entidades do PSS



Arquitectura do PSS



Compilador de PDL



Suporte de Persistência

- **Sistemas de ficheiros:**
  - Não suporta dados complexos (implica conversão do formato dos dados explícito para cada tipo):
    - exs.: CORBA Externalization, Java Serialization, COM Structured Storage, .Net Serialization
  - Não suporta transacções
  - Independente da linguagem de programação

Suporte de Persistência (cont.)

- **Bases de dados relacionais:**
  - Simples de usar
  - Interacção via RPC ou código na BD
  - Suporte de transacções limitado
  - Dependente da linguagem de programação
  - Não suporta dados complexos (implica conversão do formato dos dados explícito para cada tipo):
    - *Object-Relational-Mappings* - JDBC/ODBC

Suporte de Persistência (cont.)

- **Bases de dados orientadas a objectos:**
  - Simples de usar
  - Limitado a ambientes *single-vendor/single-server*
  - Não há um standard que seja de facto utilizado
  - Suporta dados complexos
  - Dependente da linguagem de programação

Soluções Actuais

- **The available persistence layer solutions divide into the following groups:**
  - Roll your own, using perhaps the JDBC API
  - Proprietary object-relational mapping tools or object databases (ODBMS)
  - J2EE/entity bean CMP (container-managed persistence) solutions
  - Java Data Objects (JDO)-based solutions
  - Microsoft ADO-based solutions

Roll-your-own Approach

- **In the past, most developers turned to the roll-your-own approach for projects that use relational databases for persistent storage:**
  - The two technologies used by this approach (e.g. JDBC) require a sound knowledge of the relational SQL technology and,
  - provide no transparency.
  - this approach works fine for projects with a relatively small object model
  - it can quickly lead to hard-to-maintain code because the mapping tends to be implicit/hardcoded in multiple locations in the source code.
- **Several companies have attempted to develop in-house proprietary persistence layers:**
  - such a project is a nontrivial, expensive task
  - why should we reinvent the wheel?

Proprietary Object-Relational Mapping Tools

- **Object-relational mapping tools:**
  - have been around for quite some time, are quite mature,
  - are available from many vendors (e.g TopLink from WebGain, CocoBase by Thought, Inc., Visual Business Sight Framework by ObjectMatter, etc.)
  - they all have proprietary APIs (no problem as long as they conform to an established multivendor interface; that is not the case for the products mentioned above)
  - mapping techniques vary greatly from vendor to vendor, so porting between vendors would probably be a significant issue
  - some proprietary O/R mapping tools have prohibitive deployment licensing costs in addition to development licenses.
- **If you are a middleware application architect using these tools:**
  - you and the project are locked into a single mapping tool vendor.
  - moreover, you cannot replace the whole data storage system easily.

Serialização

Serialização

- **Transformar objectos num stream de bytes de modo a serem:**
  - escritos para um ficheiro em disco,
  - enviados pela rede
- **Ler o stream de bytes existente num ficheiro em disco ou recebido via rede:**
  - mapear o objecto correspondente (implica re-construir todas as estruturas de dados necessárias em memória)
- **Formato utilizado para os dados serializados:**
  - Binário, XML, etc.

Serialização (cont.)

- **Que objectos devem ser serializados:**
  - apenas os que são directamente referenciados
  - o grafo de objectos é percorrido e todos os objectos encontrados são serializados
  - detecção de objectos já serializados (e.g.: ciclos)
- **Informação guardada de forma perene:**
  - estado do(s) objecto(s) em causa (tradução de referências)
  - informação de tipos
- **Gestão de versões:**
  - evolução dos tipos implica alterações na serialização/des-serialização

Objectos a Serializar

- It is probably better to mark all classes as serializable unless:
  - They will never cross an application domain
  - The class stores special pointers that are only applicable to the current instance of the class:
    - If a class contains unmanaged memory or file handles, for example, ensure these files are marked as NonSerialized or don't serialize the class at all.
  - Some of the data members contain sensitive information:
    - mark the class itself as serializable, but
    - mark the individual variables containing the sensitive information as non-serializable

Versões de Objectos Serializados

Versioning of Serializable Objects

- **Imagine:**
  - you create a class, instantiate it, and write it out to an object stream.
  - that flattened object sits in the file system for some time.
  - meanwhile, you update the class file, perhaps adding a new field.
  - what happens when you try to read in the flattened object?
  - an Exception will be thrown
- **E.g. in Java, java.io.InvalidClassException is thrown:**
  - all persistent-capable classes are automatically given a unique identifier
  - if the identifier of the class does not equal the identifier of the flattened object, the exception is thrown

Versioning of Serializable Objects (cont.)

- **When objects are serialied to save state in files:**
  - the potential arises that the version of a class reading the data is different than the version that wrote the data.
- **Versioning raises some fundamental questions about the identity of a class, including what constitutes a compatible change:**
  - A compatible change is a change that does not affect the contract between the class and its callers.

Versioning of Serializable Objects (cont.)

- **The goals are to:**
  - Support bidirectional communication between different versions of a class operating in different virtual machines by:
    - Defining a mechanism that allows classes to read streams written by older versions of the same class.
    - Defining a mechanism that allows classes to write streams intended to be read by older versions of the same class.
  - Provide default serialization for persistence and for remote invocation
  - Perform well and produce compact streams in simple cases, so that remote method invocation can use serialization.
  - Be able to identify and load classes that match the exact class used to write the stream.
  - Keep the overhead low for nonversioned classes.

Responsibility for Versioning of Streams

- **In the evolution of classes, it is the responsibility of the evolved (later version) class to maintain the contract established by the non-evolved class:**
  - the evolved class must not break the existing assumptions about the interface provided by the original version,
  - so that the evolved class can be used in place of the original.
  - when communicating with the original (or previous) versions, the evolved class must provide sufficient and equivalent information to allow the earlier version to continue to satisfy the nonevolved contract.
- **For the purposes of the discussion here:**
  - each class implements and extends the interface or contract defined by its supertype.
  - new versions of a class, for example foo', must continue to satisfy the contract for foo and may extend the interface or modify its implementation.

Responsibility for Versioning of Streams (cont.)

- **When objects are serialized and used by applications:**
  - communication between objects via serialization does occur
  - however, such communication is not part of the contract defined by these interfaces.
- **Serialization is a private protocol between the implementations:**
  - It is the responsibility of the implementations to communicate sufficiently to allow each implementation to continue to satisfy the contract expected by its clients.

Incompatible Changes

- **Incompatible changes to classes are those changes for which:**
  - the guarantee of interoperability cannot be maintained.
- **Deleting fields:**
  - if a field is deleted in a class, the stream written will not contain its value.
  - when the stream is read by an earlier class, the value of the field will be set to the default value because no value is available in the stream.
  - however, this default value may adversely impair the ability of the earlier version to fulfill its contract.
- **Moving classes up or down the hierarchy:**
  - this cannot be allowed since the data in the stream appears in the wrong sequence.

Incompatible Changes (cont.)

- **Changing a nonstatic field to static or a nontransient field to transient:**
  - when relying on default serialization, this change is equivalent to deleting a field from the class.
  - this version of the class will not write that data to the stream, so it will not be available to be read by earlier versions of the class.
  - as when deleting a field, the field of the earlier version will be initialized to the default value, which can cause the class to fail in unexpected ways.
- **Changing the declared type of a primitive field:**
  - each version of the class writes the data with its declared type.
  - earlier versions of the class attempting to read the field will fail because the type of the data in the stream does not match the type of the field.
- **Changing the methods that write/read objects to/from disk (e.g. writeObject or readObject in Java) so that:**
  - it no longer writes or reads the default field data or changing it so that it attempts to write it or read it when the previous version did not.
  - the default field data must consistently either appear or not appear in the stream.

Incompatible Changes (cont.)

- In Java:
  - changing a class from Serializable to Externalizable or visa-versa is an incompatible change since:
    - the stream will contain data that is incompatible with the implementation in the available class.
  - Removing either Serializable or Externalizable is an incompatible change since:
    - when written it will no longer supply the fields needed by older versions of the class.
  - Adding the writeReplace or readResolve method to a class is incompatible:
    - if the behavior would produce an object that is incompatible with any older version of the class.

Compatible Changes

- **Adding fields:**
  - When the class being reconstituted has a field that does not occur in the stream, that field in the object will be initialized to the default value for its type.
  - If class-specific initialization is needed, the class may provide a readObject method that can initialize the field to nondefault values.
- **Adding classes:**
  - The stream will contain the type hierarchy of each object in the stream.
  - Comparing this hierarchy in the stream with the current class can detect additional classes.
  - Since there is no information in the stream from which to initialize the object, the class's fields will be initialized to the default values.

Compatible Changes (cont.)

- **Removing classes:**
  - Comparing the class hierarchy in the stream with that of the current class can detect that a class has been deleted.
  - In this case, the fields and objects corresponding to that class are read from the stream.
  - Primitive fields are discarded, but the objects referenced by the deleted class are created, since they may be referred to later in the stream.
  - They will be garbage-collected when the stream is garbage-collected or reset.
- **Adding methods that write/read objects to/from disk (e.g. writeObject or readObject in Java):**
  - If the version reading the stream has these methods then readObject is expected, as usual, to read the required data written to the stream by the default serialization.
  - It should call defaultReadObject first before reading any optional data.
  - The writeObject method is expected as usual to call defaultWriteObject to write the required data and then may write optional data.

Compatible Changes (cont.)

- **Removing writeObject/readObject methods:**
  - If the class reading the stream does not have these methods, the required data will be read by default serialization, and the optional data will be discarded.
- **Adding java.io.Serializable:**
  - This is equivalent to adding types.
  - There will be no values in the stream for this class so its fields will be initialized to default values.
  - The support for subclassing nonserializable classes requires that the class's supertype have a no-arg constructor and the class itself will be initialized to default values. If the no-arg constructor is not available, the InvalidClassException is thrown.
- **Changing the access to a field:**
  - The access modifiers public, package, protected, and private have no effect on the ability of serialization to assign values to the fields.

Compatible Changes (cont.)

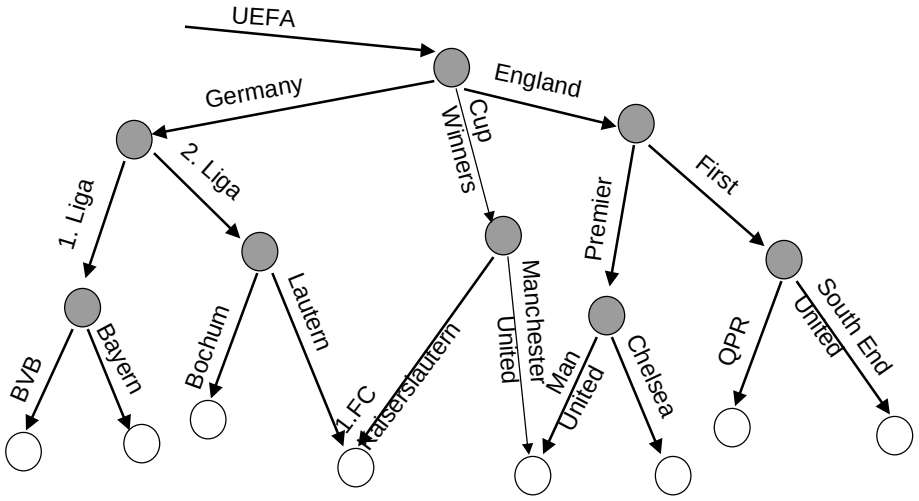
- **Changing a field from static to nonstatic or transient to nontransient:**
  - When relying on default serialization to compute the serializable fields, this change is equivalent to adding a field to the class.
  - The new field will be written to the stream but earlier classes will ignore the value since serialization will not assign values to static or transient fields.

Mapeamento Objecto-Relacional

Mapeamento Objecto-Relacional

- **Bases de dados relacionais armazenam conjuntos de tabelas**
- **O esquema da base de dados:**
  - Define as tabelas existentes
  - Cada tabela tem várias colunas
  - Define o tipo de dados que é guardado em cada uma dessas colunas
- **Mapeamento objecto-relacional (*impedance mismatch*) implica mapear os dados dos objectos nas colunas da base de dados:**
  - Para cada tipo de objecto existe uma tabela
  - Na tabela existe uma coluna para cada atributo dos objectos do tipo em causa
  - O mapeamento dos tipos básicos é feito de forma automática pela base de dados
  - Uma chave primária identifica um objecto
  - Chaves secundárias são usadas como referências para objectos na base de dados

Exemplo: serviço de nomes persistente



Tabelas na Bases de Dados

NameBindings

nctx	bind
7	10
8	11
8	12
4	13
4	14
5	15
5	16
6	17
6	18
1	2
1	3
1	4
3	5
3	6
2	7
2	8
7	9

Bindings

id	id	kind	objref
1	UEFA	NCtx	lasd098asjasfd08lasd
10	Bayern	Team	asldhf2y4395-qFDOh
11	Bochum	Team	kasdughf92ry5u-0AS
12	Lautern	Team	kasjsdhf923r5-00OSA
13	Kaiserslautern	Team	kasjsdhf923r5-00OSA
14	Manchester United	Team	lasasaskjhasldjfadfh
15	Man United	Team	lasasaskjhasldjfadfh293ruasld
16	Chelsea	Team	ksajdfgw4y250-2=0sdhfkasjgs
17	QPR	Team	aksdhf9245-qSIDHFOASFaslh
18	South End United	Team	33i4y50wswfahsdkfjh29345yoa
2	Germany	NCtx	kljsadpoweurasldkfjasofduqels
3	England	NCtx	aslkjdksahdfksgdf9qeijksadhfs
4	Cup Winners	NCtx	ashkw09we4ty0slajdfhisafsgfs
5	Premier	NCtx	saldkjfdasldkjfalsdkfh0828lsdj
6	First	NCtx	sosdihfw9y4t02-1jwohfkasjdnf
7	1. Liga	NCtx	lsadkfh924357109ejaojkasbja
8	2. Liga	NCtx	asldkjfh29y35r-q[ofdhdasldgj2
9	BVB	Team	ksaldfhqweoiry[pa\sonasglsas

NamingCtxts

id	objref
1	lasd098asjasfd08lasdsfhasdf0
2	kljsadpoweurasldkfjasofduqels
3	aslkjdksahdfksgdf9qeijksadhfs
4	ashkw09we4ty0slajdfhisafsgfs
5	saldkjfdasldkjfalsdkfh0828lsdj
6	sosdihfw9y4t02-1jwohfkasjdnf
7	lsadkfh924357109ejaojkasbja
8	asldkjfh29y35r-q[ofdhdasldgj2

Acesso a Objectos Armazenados em Bases de Dados Relacionais

- **SQL embebido:**
  - Macros que efectuam *queries* embebidos em programas escritos em C++, Java, etc.
  - Estas macros são depois expandidas por pre-processadores fornecidos com as bases de dados
  - SQL é um standard mas as macros e as APIs usadas não são
- **ODBC (*Open Database Connectivity*) da Microsoft:**
  - API standard para acesso a base de dados SQL
- **JDBC (*Java Database Connectivity*) da Sun:**
  - API standard para acesso a base de dados SQL a partir de programas escritos em Java

Bases de Dados Orientadas a Objectos

- **Suportam o armazenamento persistente:**
  - De objectos (que são instâncias de linguagens de programação OO)
  - Sem a desvantagem do impedance mismatch
- **Sistemas de gestão de bases de dados orientadas a objectos foram standardizados pela ODMG (*Object Database Management Group*):**
  - Linguagem para definição do esquema (ODL)
  - ODL é um *super-set* do CORBA-IDL
  - *Bindings* para linguagens de programação (C++, Java, Smalltalk)
  - OQL (*Object Query Language*)

Ficheiros vs. Bases de Dados

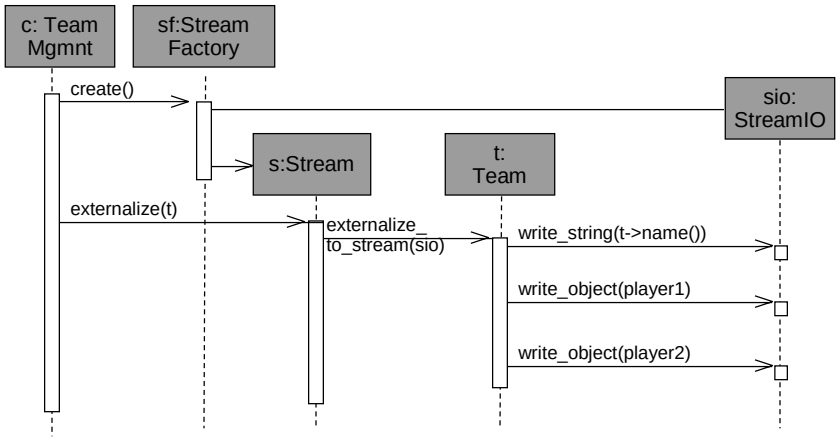
- **Ficheiros:**
  - Serialização/*externalization*
- **Bases de dados relacionais:**
  - *Impedance mismatch*
- **Bases de dados orientadas a objectos:**
  - Mesmo paradigma que as linguagens de programação orientadas a objectos

Serialização em CORBA

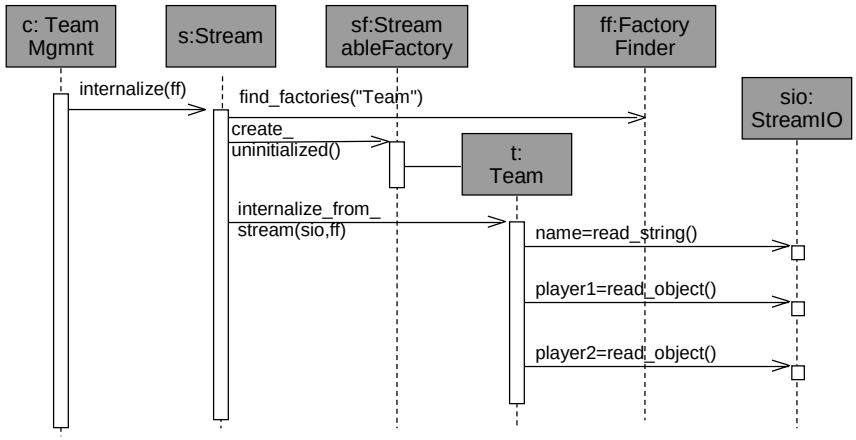
Serialização no CORBA

- **Técnica que permite:**
  - Transformar objectos num *stream* de bytes de modo a serem escritos para um ficheiro em disco (*externalization*)
  - Ler o *stream* de bytes existente num ficheiro em disco e mapear o objecto correspondente; implica re-construir todas as estruturas de dados necessárias em memória (*internalization)*
- **É mantido em disco:**
  - Estado do objecto em causa (tradução de referências)
  - Informação de tipos

Externalization no CORBA (cont.)



Internalization no CORBA (cont.)



Serialização no .Net

Serialização no .Net

- **Binary serialization:**
  - preserves type fidelity, which is useful for preserving the state of an object between different invocations of an application
  - examples:
    - share an object between different applications by serializing it to the clipboard
    - serialize an object to a stream, disk, memory, over the network, and so forth
  - remoting uses serialization to pass objects "by value" from one computer or application domain to another
- **XML serialization:**
  - serializes only public properties and fields and does not preserve type fidelity
  - useful when we want to provide or consume data without restricting the application that uses the data
  - because XML is an open standard, it is an attractive choice for sharing data across the Web
  - SOAP is an open standard, which makes it an attractive choice

Serialização no .Net (cont.)

- **The common language runtime:**
  - manages how objects are laid out in memory, and
  - provides an automated serialization mechanism by using reflection
- **The serialization engine:**
  - keeps track of all referenced objects already serialized to ensure that the same object is not serialized more than once
  - correctly handles object graphs and circular references automatically
- **The only requirement placed on object graphs is:**
  - all objects referenced by the object that is being serialized must also be marked as Serializable
  - if this is not done, an exception will be thrown when the serializer attempts to serialize the unmarked object

Serializable e MarshalByRefObject

- **Objects are only valid in the application domain where they are created:**
  - any attempt to pass the object as a parameter or return it as a result will fail, unless
  - the object derives from MarshalByRefObject or is marked as Serializable
- **If the object is marked as Serializable:**
  - the object will automatically be serialized, transported from one application domain to the other, and then
  - deserialized to produce an exact copy of the object in the second application domain.
  - this process is typically referred to as marshal by value
- **When an object derives from MarshalByRefObject:**
  - an object reference will be passed from one application domain to another, rather than the object itself

Objectos a Serializar

- **It is probably better to mark all classes as serializable unless:**
  - They will never cross an application domain:
    - If serialization is not required and the class needs to cross an application domain, derive the class from MarshalByRefObject.
  - The class stores special pointers that are only applicable to the current instance of the class:
    - If a class contains unmanaged memory or file handles, for example, ensure these files are marked as NonSerialized or don't serialize the class at all.
  - Some of the data members contain sensitive information:
    - mark the class itself as serializable, but
    - mark the individual variables containing the sensitive information as NonSerialized
    - another alternative is to implement ISerializable and serialize only the required fields

Exemplo de Serialização Binária

```
[Serializable] public class MyObject {
        public int n1 = 0;
        public int n2 = 0;
        public String str = null;
}
/* serialização */
MyObject obj = new MyObject();
obj.n1 = 1; obj.n2 = 24;
obj.str = "Some String";
IFormatter formatter = new  BinaryFormatter();
Stream stream = new FileStream("MyFile.bin",
        FileMode.Create, FileAccess.Write,
        FileShare.None);
formatter.Serialize(stream, obj);
stream.Close();
```

all member variables of a class will be serialized – even variables marked as private; in this aspect, binary serialization differs from the XML Serializer, which only serializes public fields

the Serializable attribute cannot be inherited; if we derive a new class from MyObject, the new class must be marked with the attribute as well, or it cannot be serialized

Exemplo de Serialização Binária (cont.)

```
/* des-serialização */
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin", FileMode.Open,
                               FileAccess.Read, FileShare.Read);
MyObject obj = (MyObject) formatter.Deserialize(stream);
stream.Close();

Console.WriteLine("n1: {0}", obj.n1);
Console.WriteLine("n2: {0}", obj.n2);
Console.WriteLine("str: {0}", obj.str);
```

constructors are not called when an object is deserialized

Exemplo de Serialização XML

```
<SOAP-ENV:Envelope
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP- ENC=http://schemas.xmlsoap.org/soap/encoding/
xmlns:SOAP- ENV=http://schemas.xmlsoap.org/soap/envelope/
SOAP-ENV:encodingStyle=
        "http://schemas.microsoft.com/soap/encoding/clr/1.0
         http://schemas.xmlsoap.org/soap/encoding/"
xmlns:a1="http://schemas.microsoft.com/clr/assem/ToFile">
<SOAP-ENV:Body>
  <a1:MyObject id="ref-1">
    <n1>1</n1>
    <n2>24</n2>
    <str id="ref-3">Some String</str>
  </a1:MyObject>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

replacing the formatter in the previous code with SoapFormatter

Serialização XML

- **XML serialization converts:**
  - the public fields and properties of an object, or
  - the parameters and return values of methods,
  - into an XML stream that conforms to a specific XML Schema definition language (XSD) document.
  - XML serialization results in strongly typed classes with public properties and fields that are converted to a serial format (in this case, XML) for storage or transport.
- **Because XML is an open standard:**
  - the XML stream can be processed by any application, as needed, regardless of platform.
  - for example, XML Web services created using ASP.NET use the XmlSerializer class to create XML streams that pass data between Web service applications throughout the Internet or on intranets.
  - conversely, deserialization takes such an XML stream and reconstructs the object.

Serialização XML (cont.)

- XML serialization can also be used to:
  - serialize objects into XML streams that conform to the SOAP specification.
  - SOAP is a protocol based on XML, designed specifically to transport procedure calls using XML.
- To serialize or deserialize objects:
  - use the XmlSerializer class.
  - to create the classes to be serialized, use the XML Schema Definition tool.

Serialização Selectiva

- A class often contains fields that should not be serialized:
  - assume a class stores a thread ID in a member variable
  - when the class is deserialized, the thread we stored the ID for when the class was serialized might not be running anymore
  - serializing this value does not make sense
  - we can prevent member variables from being serialized by marking them with the NonSerialized attribute as follows

```
[Serializable] public class MyObject {
    public int n1;
    [NonSerialized] public int n2;
    public String str;
}
```

Serialização "Costumizada"

- **This is particularly useful in cases where:**
  - the value of a member variable is invalid after deserialization, but
  - you need to provide the variable with a value in order to reconstruct the full state of the object
- **Customize the serialization process by implementing the ISerializable interface on an object:**
  - involves implementing the GetObjectData method (called during serialization), and
  - a special constructor that will be used when the object is deserialized.
- **When you derive a new class from one that implements ISerializable:**
  - the derived class must implement both the constructor as well as the GetObjectData method if it has any variables that need to be serialized

Exemplo de Serialização "Costumizada"

```
[Serializable] public class MyObject : ISerializable {
        public int n1; public int n2; public String str;

        public MyObject() { }
        protected MyObject (SerializationInfo info,
                                    StreamingContext context) {
                n1 = info.GetInt32("i");
                n2 = info.GetInt32("j");
                str = info.GetString("k");
        }
        public virtual void GetObjectData (SerializationInfo info,
                            StreamingContext context) {
                info.AddValue("i", n1);
                info.AddValue("j", n2);
                info.AddValue("k", str); }
    }
```

Serialização no Java

Serialização no Java

- **Técnica que permite:**
  - Transformar objectos num *stream* de bytes de modo a serem escritos para um ficheiro em disco
  - Ler o *stream* de bytes existente num ficheiro em disco e mapear o objecto correspondente (implica re-construir todas as estruturas de dados necessárias em memória
  - O grafo de objectos é percorrido de forma automática
- **Objectos em causa têm de implementar a classe *Serializable***
- **Os atributos dos objectos que não precisam de ser guardados de forma persistente:**
  - são declarados `transient` na definição da classe respectiva

Serialização no Java (cont.)

- É mantido em disco:
  - Estado do objecto em causa (tradução de referências)
  - Informação de tipos
- The defaultReadObject method is used to read the fields and object from the stream:
  - Any field of the object that does not appear in the stream is set to its default value
  - Values that appear in the stream, but not in the object, are discarded:
    - This occurs primarily when a later version of a class has written additional fields that do not occur in the earlier version.

Exemplo de Serialização

```
// Serialize today's date to a file.
FileOutputStream f = new FileOutputStream("tmp");
ObjectOutput s = new ObjectOutputStream(f);
s.writeObject("Today");
s.writeObject(new Date());
s.flush();

// Deserialize a string and date from a file.
FileInputStream in = new FileInputStream("tmp");
ObjectInputStream s = new ObjectInputStream(in);
String today = (String)s.readObject();
Date date = (Date)s.readObject();
```

Serializable e Externalizable

- **To be stored in an Object Stream:**
  - each object must implement either the Serializable or the Externalizable interface
- **For a Serializable class:**
  - Object Serialization can automatically save and restore fields of each class of an object, and
  - automatically handle classes that evolve by adding fields or supertypes.
  - A serializable class can declare which of its fields are saved or restored, and write and read optional values and objects.
- **For an Externalizable class:**
  - Object Serialization delegates to the class complete control over its external format, and
  - how the state of the supertype(s) is saved and restored.

Serializable Fields for a Class

- **Default serializable fields of a class are defined to be the non-transient and non-static fields.**
- **This default computation can be overridden by declaring a special field in the Serializable class, serialPersistentFields:**
  - This field must be initialized with an array of ObjectStreamField objects that list the names and types of the serializable fields

```
class List implements Serializable {
        List next;
        private static final ObjectStreamField[] serialPersistentFields =
                {new ObjectStreamField("next", List.class)};
```

  - By using serialPersistentFields to define the Serializable fields for a class, there no longer is a limitation that a serializable field must be a field within the current definition (i.e. a new version) of the Serializable class:
    - The writeObject and readObject methods of the Serializable class can map the current implementation of the class to the serializable fields of the class

Serializable Interface

- **A Serializable class must do the following:**
  - Implement the java.io.Serializable interface
  - Identify the fields that should be serializable
- **Use the:**
  - serialPersistentFields member to explicitly declare them serializable
  - transient keyword to denote nonserializable fields
- **The class can optionally define the following methods:**
  - A writeObject method to control what information is saved or to append additional information to the stream
  - A readObject method either to read the information written by the corresponding writeObject method or to update the state of the object after it has been restored
  - A writeReplace method to allow a class to nominate a replacement object to be written to the stream
  - A readResolve method to allow a class to designate a replacement object for the object just read from the stream

Externalizable Interface

- **For Externalizable objects:**
  - only the identity of the class of the object is saved by the container
  - the class must save and restore the contents
- **The Externalizable interface is defined as follows:**

```
package java.io;
public interface Externalizable extends Serializable {
    public void writeExternal(ObjectOutput out) throws IOException;
    public void readExternal(ObjectInput in) throws IOException,
      java.lang.ClassNotFoundException;
    }
```

- **There is no difference in how a class that implements Externalizable is used:**
  - just call writeObject() or readObject and those externalizable methods will be called automatically.

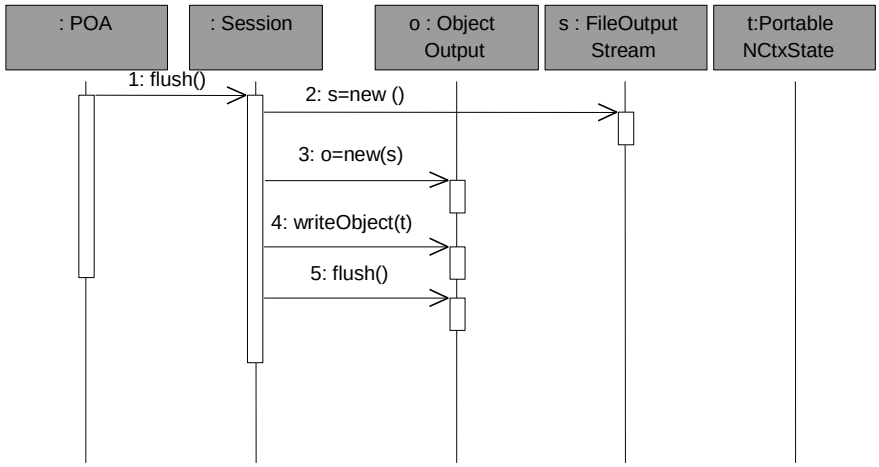
Externalizable Interface (cont.)

- **The class of an Externalizable object must do the following:**
  - Implement the java.io.Externalizable interface
  - Implement a writeExternal method to save the state of the object
  - It must explicitly coordinate with its supertype to save its state
  - Implement a readExternal method to read the data written by the writeExternal method from the stream and restore the state of the object
  - Have the writeExternal and readExternal methods be solely responsible for the format, if an externally defined format is written
  - Have a public no-arg constructor
- **An Externalizable class can optionally define the following methods:**
  - A writeReplace method to allow a class to nominate a replacement object to be written to the stream
  - A readResolve method to allow a class to designate a replacement object for the object just read from the stream

Externalizable Interface (cont.)

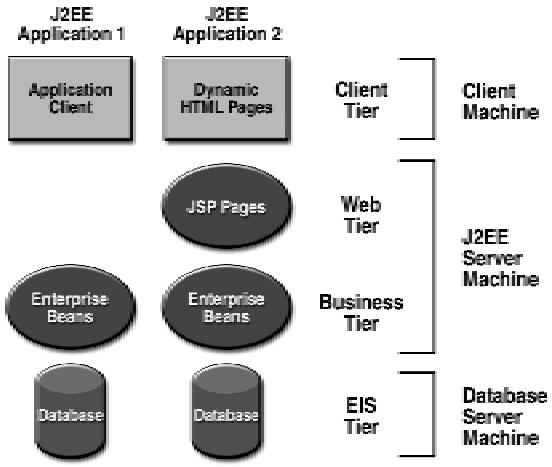
- **The writeReplace method:**
  - is called when ObjectOutputStream is preparing to write the object to the stream.
  - the ObjectOutputStream checks whether the class defines the writeReplace method.
  - if the method is defined, the writeReplace method is called to allow the object to designate its replacement in the stream.
  - the object returned should be:
    - either of the same type as the object passed in,
    - or an object that when read and resolved will result in an object of a type that is compatible with all references to the object.
  - if it is not, a ClassCastException will occur when the type mismatch is discovered.

Serialização no Java (cont.)



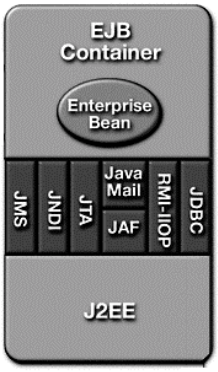
Enterprise Java Beans

Arquitectura Global



Enterprise Java Bean

- Componente de software que se executa no servidor, em ambiente distribuído
- Pode ser constituído por um ou mais objectos mas os clientes só invocam a sua interface
- Expõe a sua interface de acordo com a especificação Enterprise Java Beans

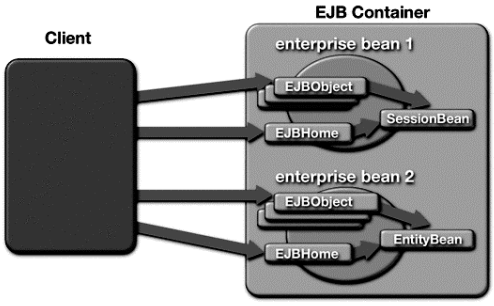


Tipos de Beans

- **Session Beans**
  - representam processos de negócio
  - executam acções
- **Entity Beans**
  - representam dados de negócio
  - chache de informação em BD
- **Message Driven Beans**
  - recebem mensagens para executar acções

EJB Container

- **O Container cria e gere as instâncias em run-time, interceptando todas as chamadas ao bean.**



EJB Object

- O EJB Object faz parte do container e replica os métodos de negócio que o bean expõe.
- Intercepta todos os pedidos ao bean para executar lógica intermédia requerida pelo container antes que esses métodos sejam executados no bean.
- Este mecanismo permite acesso implícito a:
  - Gestão de Transações (acesso a um serviço transaccional exposto através da JTA)
  - Segurança (autenticação e autorização transparentes)
  - Gestão de recursos e componentes (egstão de threads, sockets e ligações a bases de dados, etc.)
  - Persistência (assistência na salvaguarda de dados dos objectos)
  - Suporte multi-cliente (tratamento automático de pedidos concorrentes)

EJB Remote Interface

- Como é que as ferramentas que fazem a geração automática dos EBJObjects sabem quais os métodos que devem replicar?
  - O autor do Bean fornece uma interface que duplica os métodos que o bean expõe, designada por remote interface

EJB Home Object

- O código cliente lida com EJB Objects e nunca com o código do bean directamente
- Como é que os clientes obtêm referências para EJB Objects?
  - O cliente "pede" um EJB Object a uma factory, designada por HomeObject cujas principais reposnsabilidades são:
  - Criar EJBObjects
  - Procurar EJBObjects existentes (para Entities)
  - Remover EBJObjects
- Porque é que não podemos instanciar o EJB Object directamente?
  - Podem estar em máquinas diferentes.
  - A localização é transparente

Home Interface

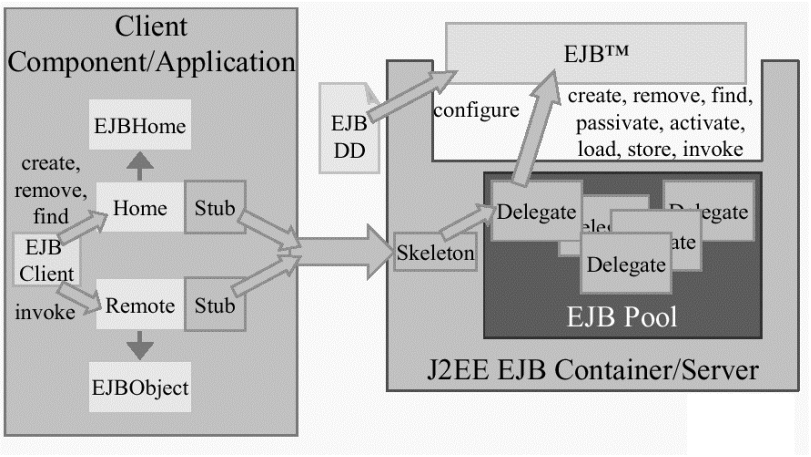
- Como é que o Home object sabe o modo como devem ser inicializados os EJB Objects?
  - O Container necessita de saber informação sobre os parâmetros para gerar HomeObjects.
  - O autor do bean fornece essa informação especificando a Home Interface:
    - Define métodos para Criar, Destruir e Procurar EBJObjects
- O EBJ Home Object implementa a home interface

Exemplo

- Cliente:
  - Procurar um Home Object
  - Usar o Home Object para criar um EBJ Object
  - Invocar métodos no EBJ Object
  - Remover o EBJ Object

```
/* Get System properties for JNDI initialization */
Properties props = System.getProperties();
/* Form an initial context */
Context ctx = new InitialContext(props);
/* Get reference to the home object, the factory for EJB objects. */
CartHome home = (CartHome) ctx.lookup("CartHome");
Cart ejbObject = home.create();
ejbObject.add("Laptop computer");
ejbObject.remove();
```


Arquitectura



EJB Design

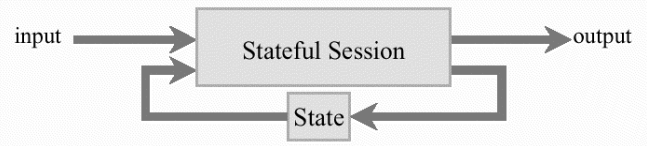
- Três aproximações para a construção de sistemas distruibuídos:
  - Servidor sem estado
  - Orientada à sessão
  - Orientada a objectos persistentes
- Na especificação EJB traduzem-se como:
  - Stateless Session e Message Driven Beans
  - Stateful Session Bean
  - Entity Bean

Stateless Session EJB

- Fornecem uma utilização única de um serviço
- Não mantêm o estado do cliente
- Só podem ser usados sincronamente
- Não sobrevivem a quebras do servidor
- Ciclo de vida relativamente curto
- Quaisquer duas instâncias do mesmo tipo são sempre idênticas

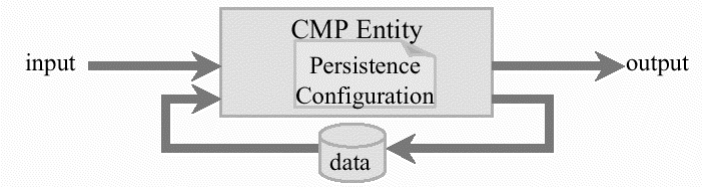
Statefull Session EJB

- Fornecem interacção conversacional
- Guardam o estado do cliente
- São síncronos
- Não sobrevivem a quebras do servidor
- Ciclo de vida relativamente curto
- Cada instância pode ser usada por uma única thread
- O container pode torná-los Activos/ Passivos



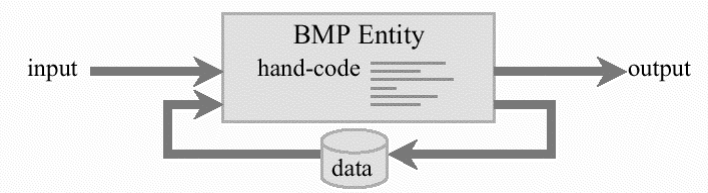
Entity EJB

- São representações de dados persistentes
- Sobrevivem a quebras do servidor
- Múltiplos clientes podem usar EJBs que representam os mesmos dados
- Funcionam como cópias em memória dos dados em armazenagem persistente
- Possuem uma Primary Key que é um identificador único para os dados



Entity Bean - Bean Managed Persistence

- Os autores dos beans são responsáveis por codificar os acessos à base de dados (SQL via JDBC, Stored Procedures, SQLJ, Java Data Objects…)
  - ebjCreate(), ejbPostCreate(), ejbRemove()
  - ejbLoad() ebjStore()
  - ebjFindByPrimaryKey()
  - ejbFind<…>(<…>)

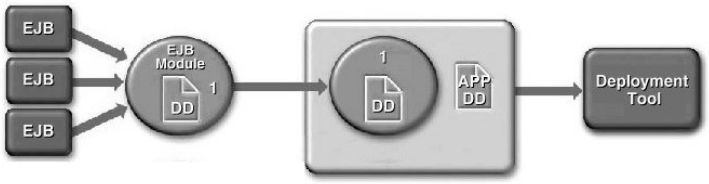


Entity Bean - Container Managed Persistence

- O EBJContainer executa implicitamente todas as operações de base de dados
- A metadata que representa o mapeamento do EJB para a base de dados encontra-se no ficheiro de Deployment que o container lê.
- É uma boa opção quando:
  - O desenho dos EJB mapeia facilmente com o da base de dados
  - As relações entre os Entity Beans (e/ou objectos dependentes são simples) e pouco profundas (até 3/4 classes)
- Torna-se complexa quando a integridade referencial, gestão de relações e cascading deletes têm que ser tomados em consideração

Deployment

- Criar o XML que descreve o EJB
  - gestão de componentes
  - requisitos de persistência
  - requisitos para transacções
  - requisitos de segurança
- Criar um jar com o XML DD e o bean
- Gerar os ficheiros do container para o EJB
- Configurar propriedades do servidor para o EJB



ADO .Net

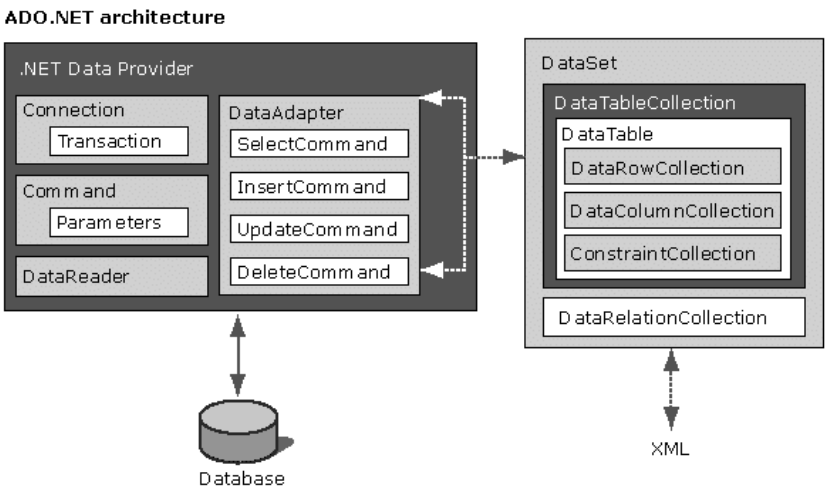
ADO .Net

- ADO.NET was designed to meet the needs of a new programming model:
  - disconnected data architecture,
  - tight integration with XML,
  - common data representation with the ability to combine data from multiple and varied data sources, and
  - optimized facilities for interacting with a database, all native to the .NET Framework.
- ADO.NET provides consistent access to data sources such as:
  - Microsoft SQL Server, as well as
  - data sources exposed via OLE DB and XML.

ADO .Net (cont.)

- Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data:
  - The ADO.NET components have been designed to factor data access from data manipulation.
- ADO.NET includes:
  - .NET data providers for connecting to a database,
  - executing commands, and
  - retrieving results
- There are two central components of ADO.NET that accomplish this:
  - the DataSet, and
  - the .NET data provider, which is a set of components including the Connection, Command, DataReader, and DataAdapter objects.

ADO .Net Architecture



DataSet

- **Results are either processed directly, or placed in an ADO.NET DataSet object in order to:**
  - be exposed to the user in an ad-hoc manner,
  - combined with data from multiple sources, or
  - remoted between tiers.
- **The ADO.NET DataSet object can also be used independently of a .NET data provider:**
  - to manage data local to the application or sourced from XML.
- **The DataSet is explicitly designed for data access independent of any data source:**
  - It contains a collection of one or more DataTable objects
  - DataTable are made up of rows and columns of data, as well as primary key, foreign key, constraint, and relation information about the data in the DataTable objects

DataSet (cont.)

- The DataSet is:
  - a memory-resident representation of data that
  - provides a consistent relational programming model regardless of the data source.
  - It can be used with multiple and differing data sources:
    - used with XML data, or
    - used to manage data local to the application.
  - The DataSet represents a complete set of data including related tables, constraints, and relationships among the tables
- **The methods and objects in a DataSet are consistent with those in the relational database model.**

ADO .Net (cont.)

- **.NET data provider:**
  - Its components are explicitly designed for data manipulation and fast, forward-only, read-only access to data
- **A .NET data provider is used for connecting to a database, executing commands, and retrieving results:**
  - Those results are either processed directly, or
  - placed in an ADO.NET DataSet
- **The DataSet can also persist and reload its contents as XML and its schema as XML Schema definition language (XSD) schema.**

ADO .Net (cont.)

- **.NET provider objects:**
  - **Connection** object provides connectivity to a data source.
  - **Command** object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information
  - **DataReader** provides a high-performance stream of data from the data source
  - **DataAdapter** provides the bridge between the **DataSet** object and the data source.
- **The DataAdapter uses Command objects to execute SQL commands at the data source to:**
  - both load the DataSet with data, and
  - reconcile changes made to the data in the DataSet back to the data source.

XML and ADO.NET

- **ADO.NET and the XML classes in the .NET Framework converge in the DataSet object:**
  - The DataSet can be populated with data from an XML source, whether it is a file or an XML stream.
  - The DataSet can be written as World Wide Web Consortium (W3C) compliant XML, including its schema as XML Schema definition language (XSD) schema,
  - regardless of the source of the data in the DataSet.
  - Because the native serialization format of the DataSet is XML, it is an excellent medium for moving data between tiers
  - making the DataSet an optimal choice for remoting data and schema context to and from an XML Web service.

DataSet vs. DataReader

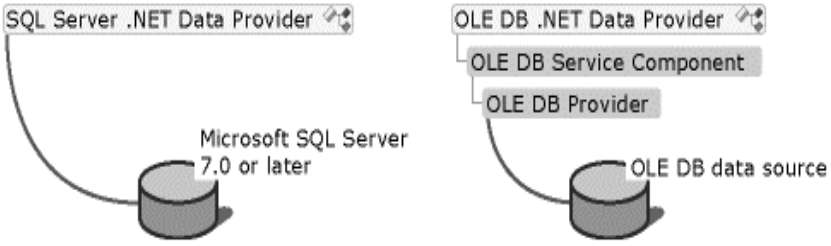
- **Use a DataSet to do the following:**
  - Remote data between tiers or from an XML Web service.
  - Cache data locally in your application.
  - Perform extensive processing on data without requiring an open connection to the data source, which frees the connection to be used by other clients.
- **If you do not require the functionality provided by the DataSet:**
  - you can improve the performance of your application by using the DataReader to return your data in a forward-only read-only fashion.
  - Although the DataAdapter uses the DataReader to fill the contents of a DataSet, by using the DataReader you can receive performance gains because you will save memory that would be consumed by the DataSet, as well as saving the processing required to create and fill the contents of the DataSet.

.NET Data Provider

- **The .NET Framework includes:**
  - the SQL Server .NET Data Provider (for Microsoft SQL Server version 7.0 or later), and
  - the OLE DB .NET Data Provider.
  - An Open Database Connectivity (ODBC) .NET Data Provider is available as a separate download
- **The SQL Server .NET Data Provider uses its own protocol to communicate with SQL Server:**
  - it is lightweight and performs well because it is optimized to access a SQL Server directly without adding an OLE DB or Open Database Connectivity (ODBC) layer.
- **The OLE DB .NET Data Provider communicates to an OLE DB data source through:**
  - both the OLE DB Service component, which provides connection pooling and transaction services, and
  - the OLE DB Provider for the data source.

.NET Data Provider (cont.)

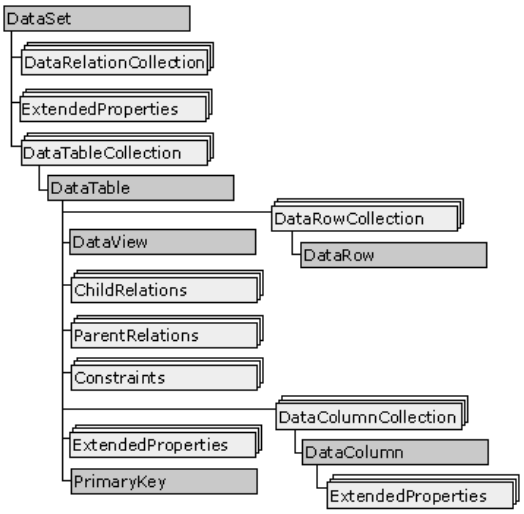
Comparison of the SQL Server .NET Data Provider and the OLE DB .NET Data Provider



Choosing a .NET Data Provider

Provider	Notes
SQL Server .NET Data Provider	Recommended for middle-tier applications using Microsoft SQL Server 7.0 or later. Recommended for single-tier applications using Microsoft Data Engine (MSDE) or Microsoft SQL Server 7.0 or later. Recommended over use of the OLE DB Provider for SQL Server (SQLOLEDB) with the OLE DB .NET Data Provider. For Microsoft SQL Server version 6.5 and earlier, you must use the OLE DB Provider for SQL Server with the OLE DB .NET Data Provider.
OLE DB .NET Data Provider	Recommended for middle-tier applications using Microsoft SQL Server 6.5 or earlier, or any OLE DB provider that supports the OLE DB interfaces listed in OLE DB Interfaces Used by the OLE DB .NET Data Provider (OLE DB 2.5 interfaces are not required). For Microsoft SQL Server 7.0 or later, the SQL Server .NET Data Provider is recommended. Recommended for single-tier applications using Microsoft Access databases. Use of a Microsoft Access database for a middle-tier application is not recommended. Support for the OLE DB Provider for ODBC (MSDASQL) is disabled. For access to Open Database Connectivity (ODBC) data sources, an ODBC .NET Data Provider is available as a separate download at http://msdn.microsoft.com/downloads.

DataSet Object Model



DataTableCollection

- **An ADO.NET DataSet contains a collection of zero or more tables represented by DataTable objects:**
  - The DataTableCollection contains all the DataTable objects in a DataSet.
  - A DataTable is defined in the System.Data namespace and represents a single table of memory-resident data.
  - it contains a collection of columns represented by a DataColumnCollection, and constraints represented by a ConstraintCollection, which together define the schema of the table.
  - A DataTable also contains a collection of rows represented by the DataRowCollection, which contains the data in the table.
  - Along with its current state, a DataRow retains both its current and original versions to identify changes to the values stored in the row.

DataRelationCollection

- **A DataSet contains relationships in its DataRelationCollection object:**
  - A relationship is represented by the DataRelation object,
  - It associates rows in one DataTable with rows in another DataTable.
  - It is analogous to a join path that might exist between primary and foreign key columns in a relational database.
  - A DataRelation identifies matching columns in two tables of a DataSet.
- **Relationships enable navigation from one table to another within a DataSet:**
  - The essential elements of a DataRelation are the name of the relationship, the name of the tables being related, and the related columns in each table.
  - Relationships can be built with more than one column per table by specifying an array of DataColumn objects as the key columns.
  - When a relationship is added to the DataRelationCollection, it may optionally add a UniqueKeyConstraint and a ForeignKeyConstraint to enforce integrity constraints when changes are made to related column values.

ExtendedProperties

- **The DataSet (as well as the DataTable and DataColumn) has an ExtendedProperties property:**
  - ExtendedProperties is a PropertyCollection where you can place customized information,
  - such as the SELECT statement that was used to generate the resultset, or
  - a date/time stamp of when the data was generated.
  - The ExtendedProperties collection is persisted with the schema information for the DataSet (as well as the DataTable and DataColumn).

Example

```
using System; using System.Data; using System.Data.SqlClient;
class Sample {
    public static void Main() {
      SqlConnection nwindConn = new SqlConnection("Data Source=localhost;
                    Integrated Security=SSPI; Initial Catalog=northwind");
    SqlCommand catCMD = nwindConn.CreateCommand();
    catCMD.CommandText = "SELECT CategoryID, CategoryName FROM Categories";
    nwindConn.Open();
    SqlDataReader myReader = catCMD.ExecuteReader();
    while (myReader.Read()) {
      Console.WriteLine("\t{0}\t{1}", myReader.GetInt32(0), myReader.GetString(1));
    }
    myReader.Close();
    nwindConn.Close();
    }
}
```


.Net Provider Classes

Object	Description
Connection	Establishes a connection to a specific data source.
Command	Executes a command against a data source.
DataReader	Reads a forward-only, read-only stream of data from a data source.
DataAdapter	Populates a **DataSet** and resolves updates with the data source.

Object	Description
Transaction	Enables you to enlist commands in transactions at the data source.
CommandBuilder	A helper object that will automatically generate command properties of a **DataAdapter** or will derive parameter information from a stored procedure and populate the **Parameters** collection of a **Command** object.
Parameter	Defines input, output, and return value parameters for commands and stored procedures.
Exception	Returned when an error is encountered at the data source. For an error encountered at the client, .NET data providers throw a .NET Framework exception.
Error	Exposes the information from a warning or error returned by a data source.
ClientPermission	Provided for .NET data provider code access security attributes.

Connection

- In ADO.NET you use a Connection object to connect to a specific data source
- To connect to Microsoft SQL Server version 7.0 or later:
  - use the SqlConnection object of the SQL Server .NET Data Provider.

```
SqlConnection nwindConn = new SqlConnection("Data Source = localhost;
        Integrated Security = SSPI; " + "Initial Catalog = northwind");
nwindConn.Open();
```

- To connect to an OLE DB data source, or to Microsoft SQL Server version 6.*x* or earlier using the OLE DB Provider for SQL Server (SQLOLEDB):
  - use the OleDbConnection object of the OLE DB .NET Data Provider.

Pool Creation and Assignment

- Pooling connections can significantly enhance the performance and scalability of your application:
  - The SQL Server .NET Data Provider provides connection pooling automatically for your ADO.NET client application.
- When a connection is opened:
  - a connection pool is created based on an exact matching algorithm that associates the pool with the connection string in the connection.
  - Each connection pool is associated with a distinct connection string.
  - When a new connection is opened, if the connection string is not an exact match to an existing pool, a new pool is created.

Example

- **Three new SqlConnection objects are created, but only two connection pools are required to manage them**

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Integrated Security=SSPI; Initial Catalog = northwind";
conn.Open(); // Pool A is created.
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Integrated Security=SSPI; Initial Catalog=pubs";
conn.Open(); // Pool B is created because the connection strings differ.
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Integrated Security=SSPI;Initial Catalog=northwind";
conn.Open(); // The connection string matches pool A.
```


Connection Addition

- **A connection pool is created for each unique connection string:**
  - When a pool is created, multiple connection objects are created and added to the pool so that the minimum pool size requirement is satisfied.
  - Connections are added to the pool as needed, up to the maximum pool size.
- **When a SqlConnection object is requested:**
  - it is obtained from the pool if a usable connection is available.
  - To be usable, the connection must:
    - currently be unused,
    - have a matching transaction context or not be associated with any transaction context, and
    - have a valid link to the server.
- **If the maximum pool size has been reached and no usable connection is available:**
  - the request is queued.
  - The object pooler satisfies these requests by reallocating connections as they are released back into the pool.
  - If the time-out period (determined by the Connect Timeout connection string property) elapses before a connection object can be obtained, an error occurs.

Connection Removal

- **The object pooler will remove a connection from the pool:**
  - if the connection lifetime has expired, or
  - if the pooler detects that the connection with the server has been severed.
  - Note that this can be detected only after attempting to communicate with the server.
  - If a connection is found that is no longer connected to the server, it is marked as invalid.
  - The object pooler periodically scans connection pools looking for objects that have been released to the pool and are marked as invalid.
  - These connections are then permanently removed.
- **If a connection exists to a server that has disappeared:**
  - it is possible for this connection to be drawn from the pool even if the object pooler has not detected the severed connection and marked it as invalid.
  - When this occurs, an exception is generated.
  - However, you must still close the connection in order to release it back into the pool.

Transaction Support

- **Connections are drawn from the pool and assigned based on transaction context:**
  - The context of the requesting thread and the assigned connection must match.
  - Therefore, each connection pool is actually subdivided into:
    - connections with no transaction context associated with them, and into
    - N subdivisions that each contain connections with a particular transaction context.
- **When a connection is closed:**
  - it is released back into the pool and into the appropriate subdivision based on its transaction context.
  - Therefore, you can close the connection without generating an error, even though a distributed transaction is still pending.
  - This allows you to commit or abort the distributed transaction at a later time.

XML and the DataSet

- **With ADO.NET you can fill a DataSet from an XML stream or document:**
  - You can use the XML stream or document to supply to the DataSet either data, schema information, or both.
  - The information supplied from the XML stream or document can be combined with existing data or schema information already present in the DataSet.
- **ADO.NET also allows you to:**
  - create an XML representation of a DataSet, with or without its schema,
  - in order to transport the DataSet across HTTP for use by another application or XML-enabled platform.
- **In an XML representation of a DataSet:**
  - the data is written in XML and the schema, if it is included inline in the representation, is written using the XML Schema definition language (XSD).
  - XML and XML Schema provide a convenient format for transferring the contents of a DataSet to and from remote clients.

Java Data Objects (JDO)

JDO

- **The Java Community Process developed Java Data Objects (JDO):**
  - a high-level API specification and a reference implementation
  - Sun's JDO specification defines a simple, transparent interface between application objects and transactional data stores
- **The spec covers:**
  - Persistence semantics with respect to transactions
  - Interactions of transactional objects with J2EE
  - Data selection (queries) based on Java expressions
- **Specification's three fundamental concepts:**
  - JDO instance
  - First-class objects
  - Second-class objects

JDO Instance

- **JDO instance:**
  - is a Java class instance that implements the application functions, and
  - represents data in an enterprise data store. A
- **JDO instance has an important limitation:**
  - its class must always implement the **PersistenceCapable** interface (defined below),
  - either explicitly by the class writer or implicitly by the enhancer's results.

JDO First-Class Objects

- **First-class objects:**
  - Instances of the **PersistenceCapable** classes that have JDO identity represent first-class objects;
  - these objects are stored in a data store with their associated second-class objects (if any) and primitive values.
  - First-class objects are unique:
    - when a **PersistentManager** (defined below) instantiates one into memory, that same **PersistentManager** manages an instance representing that first-class object,
    - though other **PersistentManager**s might manage other instances of that same class.

JDO Second-Class Objects

- **Also PersistenceCapable instances:**
  - second-class objects differ from first-class objects in that they have no JDO identity of their own.
  - Second-class objects notify their first-class objects of their modification;
  - that modification reflects as a change to that first-class object.
  - Second-class objects are stored in the data store as part of a first-class object only.
- **Example of first/second-class objects:**
  - an object of class Order with that has one or more instances of OrderLine items.
  - Order is a first-class object example
  - OrderLine is a second-class object example.

JDO Interfaces

- **Major public interfaces:**
  - PersistenceManagerFactory
  - PersistenceManager
  - PersistenceCapable
  - Transaction
  - Query

PersistenceManagerFactory

- **The PersistenceManagerFactory:**
  - creates PersistenceManager instances for application use.
  - It also lets application developers configure the persistence layer behavior:
    - set transaction options, perform connection pool administration, etc.
- **In a managed environment, the application uses JNDI (Java Naming and Directory Interface):**
  - lookup to retrieve an environment-named object,
  - which is then cast to javax.jdo.PersistenceManagerFactory.

PersistenceManager

- **The JDO PersistenceManager:**
  - provides the primary interface for JDO-aware application components.
  - administers persistent instances' lifecycles, and
  - provides transaction and cache management.
  - It also acts as the Query interface's factory.

PersistenceCapable

- **Any user domain class must implement the PersistenceCapable interface.**
- **There are no special methods for persistence:**
  - the PersistenceCapable interface is really empty.
  - You can implement this interface in one of three ways:
    - through source code,
    - an enhancer, or
    - generation (tool-based).

Transaction

- A one-to-one relationship exists between the PersistenceManager and the Transaction.
- In managed environments:
  - the container provides actual transaction services, but
  - the Transaction interface provides methods for managing transaction options.
- In a standalone environment:
  - the Transaction implementation, provided by the JDO software vendor, must ensure a successful transaction -- commit or rollback.

Cache Management and JDO Instance Lifecycle

- Every JDO object (instance) goes through a series of state changes in its lifetime:
  - The Sun JDO specification defines 10 JDO instance states.
- Transient:
  - When an instance is transient, the object lacks persistent identity.
  - A transient instance changes its state to persistent-new (see below) in one of two ways:
    - when passed as an argument to the makePersistent() method or
    - when referenced by a persistent instance's persistent field after that same instance commits.
- Persistent-new:
  - An instance that is newly persistent in the current transaction.
  - When an application component requests an instance to become persistent, that instance assumes the persistent-new state and receives a persistent identity.

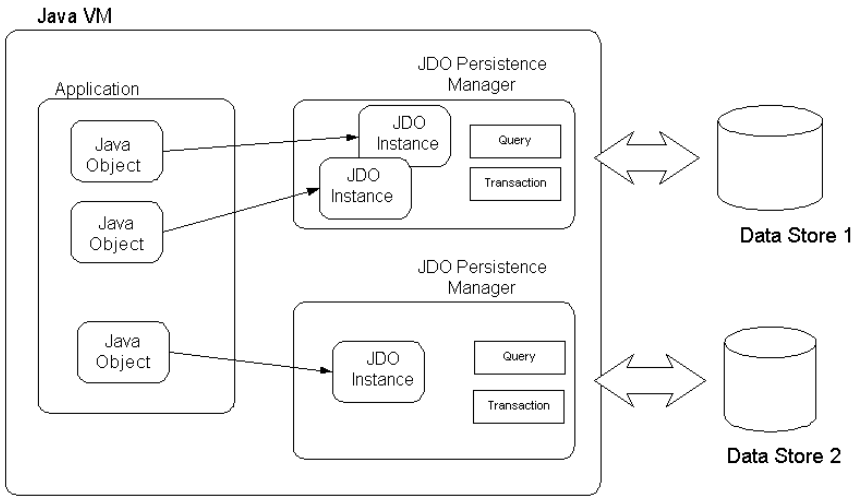
Cache Management and JDO Instance Lifecycle (cont.)

- **Persistent-dirty:**
  - An instance's state when one or more of its attributes have changed (within the current transaction), but not yet persisted.
- **Hollow:**
  - A JDO instance that represents specific persistent data in the data store, but whose values are not in the JDO instance.
- **Persistent-clean:**
  - A JDO instance that represents specific transactional persistent data in the data store, and
  - whose values have not changed (within the current transaction).
- **Persistent-deleted:**
  - JDO instances that represent specific persistent data in the data store and
  - have been deleted in the current transaction.
- **Persistent-new-deleted:**
  - JDO instances that represent new persistent instances deleted from the current transaction.

Queries

- **The** PersistentManager **instance:**
  - is a factory for query instances, and
  - queries execute in the context of the **PersistentManager** instance.
- **Queries must conform to the Object Query Language (OQL) grammar:**
  - defined in the Object Management Group (OMG) 3.0 OQL Specification.
  - The JDO OQL resembles SQL, except that it operates on Java classes and objects, not tables.

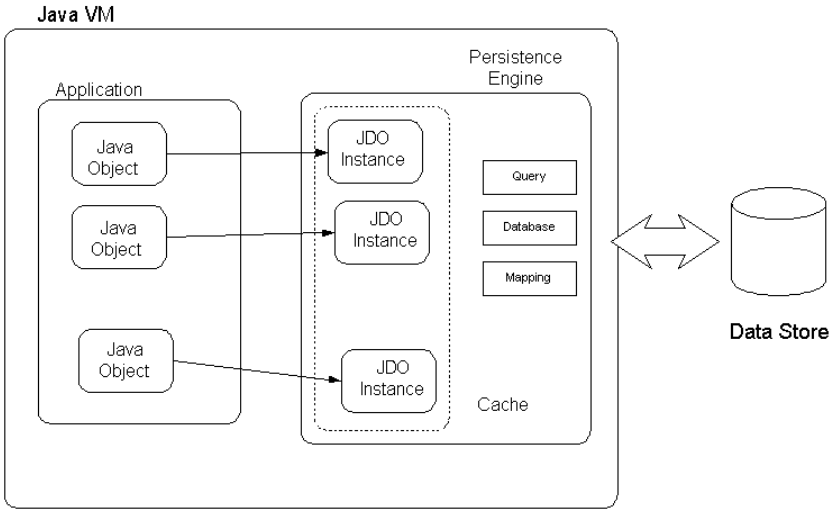
Sun JDO Architecture



Castor JDO

- **The Castor JDO project focuses on the Java object persistence-to-relational data stores:**
  - Despite its name and resemblance, Castor JDO is not compatible with Sun's spec.
  - it concentrates on relational data stores exclusively, thus it does not support data-storage type transparency.

Castor JDO Architecture



JDO vs. Others

- **JDO vs. JDBC:**
  - less code to write and maintain
  - an average of three times less code than JDBC
- **JDO vs. entity beans:**
  - If a developer wants to take advantage of EJB's (Enterprise JavaBeans) lifecycle management, security, and an entity bean's distributed nature, then EJB is the right choice.
  - However, JDO specifications have **much** less overhead and provide more design freedom:
    - data objects are still Java objects, not entity beans.
    - In most cases, developers don't need entity bean's remote capability because they access the entity beans through appropriate session beans.

Transacções

Transacções

- **Motivação:**
  - O que sucede quando ocorre uma falha durante a modificação de um recurso em disco?
  - Quais as operações que não foram completadas?
  - Que operações têm ou não de ser re-executadas?
  - Em que estado se encontram os recursos?
- **Propriedades (ACID):**
  - Atomicidade
  - Consistência
  - Isolamento
  - Durabilidade
- **Transacções:**
  - *Commit* vs. *Abort*
  - *Flat* vs. *nested*
  - Centralizadas vs. distribuídas

Propriedades

- **Atomicidade:**
  - As modificações efectuadas no âmbito de uma transacção ou são todas executadas ou nenhuma o é
  - É sempre possível voltar ao início (*roll-back*) de uma transacção
- **Consistência:**
  - Os dados partilhados estão sempre consistentes
  - Os dados podem estar inconsistentes apenas durante a execução de uma dada transacção:
    - Escondidos de outras transacçõoes concorrentes
    - Resolvidas aquando da terminação da transacção
  - Aplicações são responsáveis pela consistência dos dados
  - Transacção aborta se as modificações dos dados implicam que estes fiquem inconsistentes

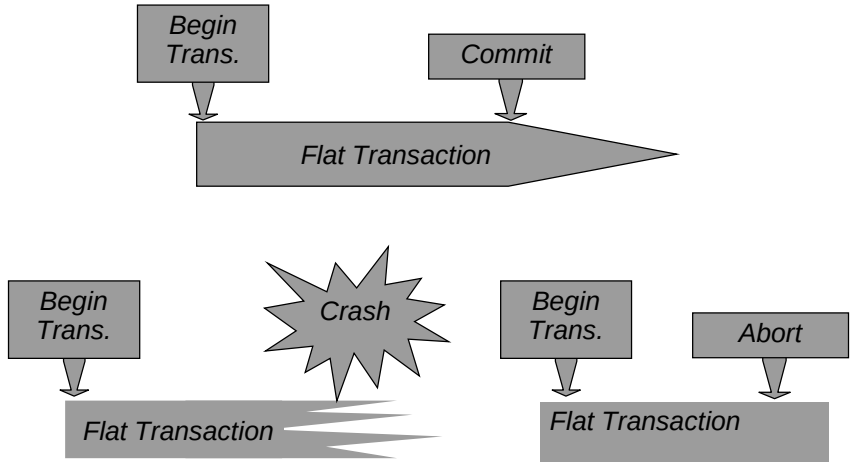
Propriedades (cont.)

- **Isolamento:**
  - Cada transacção acede aos dados como se não houvesse nenhuma transacção concorrente
  - As modificações executadas no âmbito de uma transacção só se tornam visíveis para outras depois de terminar (com sucesso)
  - Implementado através de:
    - *two-phase locking*
    - *optimistic concurrency control*
- **Durabilidade:**
  - As modificações efectuadas no âmbito de uma transacção que terminou com sucesso persistem
  - Os dados modificados devem existir em disco antes da transacção terminar
  - Dados podem ser guardaos em disco, Flash RAM, etc.

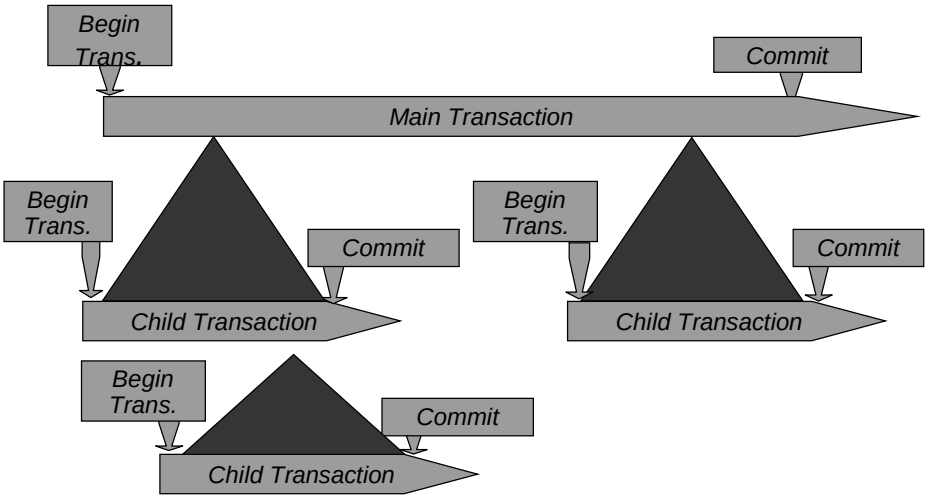
Operações Transaccionais

- **Begin:**
  - Iniciar uma nova transacção
- **Commit:**
  - Terminar uma transacção
  - As modificações efectuadas aos dados são armazenadas
  - A nova versão dos dados são disponibilizadas a outras transacções
- **Abort:**
  - Terminar uma transacção
  - As modificações até aí efectuadas são anuladas

Flat Transactions



Nested Transactions



Transacções Centralizadas vs. Distribuídas

- **Transacções num base de dados:**
  - Centralizado
  - DBMS controla a execução das transacções
  - DBMS implementa o controle de concorrência
  - Processamento transaccional é transparente aos programadores
- **Surgem dificuldades se:**
  - Os dados acedidos encontram-se em bases de dados distintas
  - Dados acedidos não estão em nenhuma base de dados

Sistema Transaccional (TP)

- **A distributed TP system consists of several cooperating entities:**
  - Transaction Processing (TP) Monitors
  - Transaction Managers
  - Resource Managers
  - Resource Dispensers
- **These entities are logical and can reside on the same computer or on different computers.**

TP monitor

- **A TP monitor is software that sits between a transaction-aware application and a collection of resources:**
  - It maximizes operating system activities, streamlines network communications, and connects multiple clients to multiple applications that potentially access multiple data resources.
  - Instead of writing an application that manages a multi-user, distributed environment, you write an application that consists of single transaction requests.
  - The monitor scales your application as required.
  - Example: the Distributed Transaction Coordinator (DTC) is the TP monitor for Microsoft Windows 2000.

Transaction Managers

- **In a distributed transaction, each participating resource has a local transaction manager (TM):**
  - to track incoming and outgoing transactions on that computer.
  - The TP monitor assigns one TM the additional task of coordinating all activities among local TMs.
  - The TM that coordinates transaction activities is called the root or coordinating TM.
- **A TM:**
  - coordinates and manages all transaction processing functions, but
  - it is not equipped to manage data directly.
  - Resource managers handle data-related activities

Resource Managers

- **A resource manager is:**
  - a system service that manages persistent or durable data in databases, durable message queues, or transactional file systems.
  - stores data and performs disaster recovery.
- **Examples:**
  - Both SQL Server and MSMQ provide resource managers that participate in distributed transactions.
  - Oracle, Sybase, Informix, IBM (for IBM DB2), and Ingres also provide compatible resource managers for their database products.

Resource Dispensers

- **A resource dispenser manages nondurable state that can be shared:**
  - For example, the ODBC resource dispenser:
    - manages pools of database connections,
    - reclaiming each connection when it is no longer needed

Distributed Transactions

- **Distributed transaction processing (TP) systems are designed to facilitate transactions that span:**
  - heterogeneous, transaction-aware resources in a distributed environment.
- **Supported by a distributed TP system, your application can combine into a transactional unit such diverse activities:**
  - as retrieving a message from a Microsoft Message Queuing (MSMQ) queue,
  - storing the message in a Microsoft SQL Server database, and
  - removing all existing references to the message from an Oracle Server database.

Transaction Models

- **Manual transaction:**
  - you control transaction boundaries with explicit instructions to begin and end the transaction.
  - From within one transaction boundary, you can begin a second transaction, called a nested transaction.
  - The parent transaction does not commit until all its subordinate transactions commit.
- **Automatic transaction:**
  - manages transaction boundaries for you, based on a declarative attribute set for each component.
  - A transaction automatically flows to objects instructed to participate in a transaction and bypasses objects instructed to execute outside a transaction.
  - You cannot nest transactions when using the automatic transaction model.

Manual Transactions

- **These APIs enable manual transaction processing:**
  - Microsoft ActiveX Data Objects (ADO),
  - OLE DB,
  - Open Database Connectivity (ODBC),
  - Microsoft Message Queuing (MSMQ).
- **A manual transaction allows you to explicitly:**
  - begin a transaction,
  - control each connection and resource enlistment within the transaction boundary,
  - determine the outcome of the transaction (commit or abort), and
  - end the transaction.

Manual Transactions (cont.)

- **Although this manual model offers measured control over a transaction:**
  - it lacks some of the ease built into the automatic transaction model.
  - For example, there is no automatic enlistment and coordination between data stores in a manual transaction.
  - Further, transactions do not flow from object to object, as is the case in automatic transactions.
- **If you choose to control a distributed transaction manually, you must manage:**
  - recovery, concurrency, security, and integrity.
  - In other words, you must apply all the programming techniques necessary to maintain the ACID Properties associated with transaction processing.

Two Phase Locking (2PL)

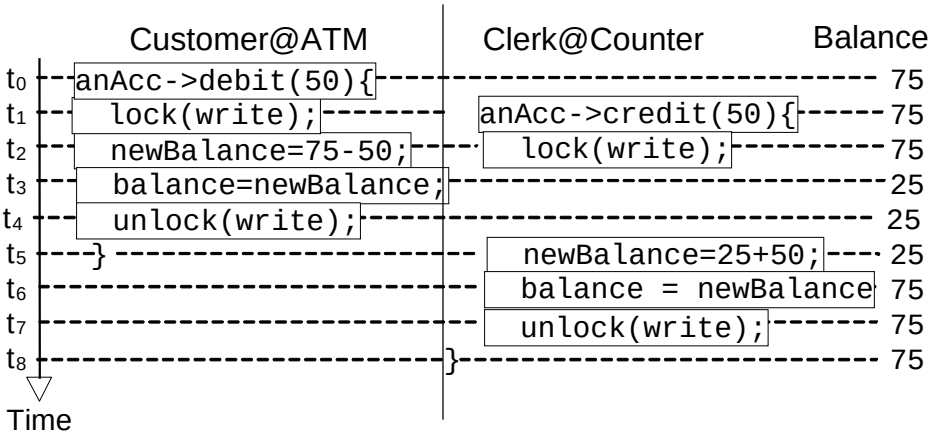
- **Algoritmo de controle de concorrência mais usado:**
  - Bases de dados relacionais (Oracle, Ingres, Sybase, DB/2, etc.)
  - Bases de dados orientadas a objectos (O2, ObjectStore, Versant, etc.)
  - Monitores transaccionais (CICS, etc)
- **Processos que executam as transacções adquirem locks a partir de um processo gestor de locks**
- **Gestor de locks assegura que os locks dados não originam conflitos**
- **Um lock não é mais que um token que permite a um processo aceder aos dados segundo um modo (leitura, escrita)**
- **Processos adquirem locks, acedem aos dados, e depois libertam os locks**
- **2PL:**
  - Fase de aquisição de locks
  - Fase de libertação dos locks (depois de inciada, o processo não adquire mais lock nenhum)

Conflitos de *Locking*

- **Compatibilidade:**
  - *Single-writer multiple readers*
- **Quando um lock não pode ser adquirido:**
  - Processo fica em espera até que o gestor lhe ceda o lock
  - Processo recebe resposta negativa do gestor

	read	write
read	sim	não
write	não	não

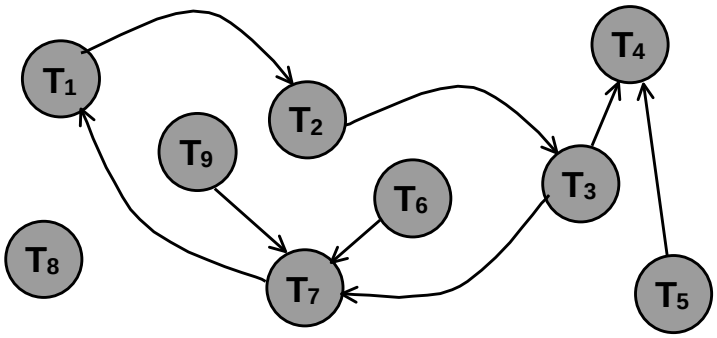
Exemplo



Deadlock

- **Podem surgir situações de *deadlock:***
  - Detectadas pelo gestor de locks
  - Detectadas através da análise do *wait-graph* (se contiver ciclos)
  - Complexidade da detecção é proporcional ao número de transacções concorrentes
- **Resolução:**
  - Abortar uma ou mais transacções em curso:
    - As que já consumiram mais recursos
    - As que têm mais dependências

Exemplo de *Deadlock*



- ***Deadlock***: $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_7 \rightarrow T_1$

Entidades Intervenientes

- **Num sistema distribuído com suporte transaccional há a considerar:**
  - Cliente da transacção
  - Servidor da transacção
  - Coordenador da transacção
- **Coordenador:**
  - Gere a transacção (vários passos: begin, commit, abort)
  - Atribui identificadores únicos às transacções
  - Note-se que transacções distintas podem ter coordenadores diferentes
- **Servidor:**
  - Engloba um dado recurso acedido transacionalmente
  - Conhece o coordenador das transacções em que participa
  - Regista, junto do coordenador, a sua participação em transacções
  - Suporta o protocolo transacional (*two-phase commit*)
- **Cliente:**
  - No que diz respeito a transacções, apenas conhece o coordenador
  - Invoca o coordenador (begin, commit, abort)
  - A implementação das transacções é completamente transparente ao cliente
  - Não consegue distinguir entre coordenador e servidor

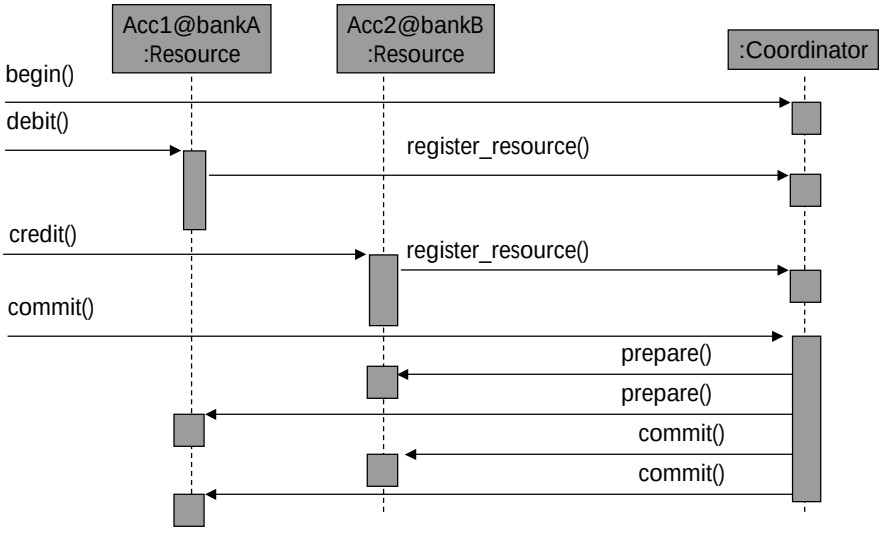
Two-Phase Commit (2PC)

- **Vários servidores distribuídos autónomos:**
  - Todos os servidores envolvidos numa transacção têm de acordar em fazer commit
  - Basta um servidor não acordar em fazer commit para que a transacção (todos os servidores) faça abort
- **Dua fases para chegar a acordo:**
  - Fase 1: votação
  - Fase 2: terminação

2PC

- **Fase 1:**
  - Coordenador pergunta a cada servidor se fazem commit
  - Cada servidor responde (umas das duas):
    - Sim, faço commit se for essa a decisão do coordenador e fico à espera dessa decisão
    - Não, aborto de imediato a transacção
- **Fase 2:**
  - Coordenador recebe todos as respostas e decide em conformidade:
    - commit se todos os servidores responderam afirmativamente
    - abort se pelo menos um respondem negativamente
  - A todos os servidores que responderam afirmativamente é enviada a decisão (uma das duas): commit ou abort
  - Cada servidor faz ACK depois de ter efectuado o commit dos seus dados
- **Nenhum dos servidores pode decidir fazer commit sozinho**

Exemplo: transferência de fundos



Transacções no .CORBA

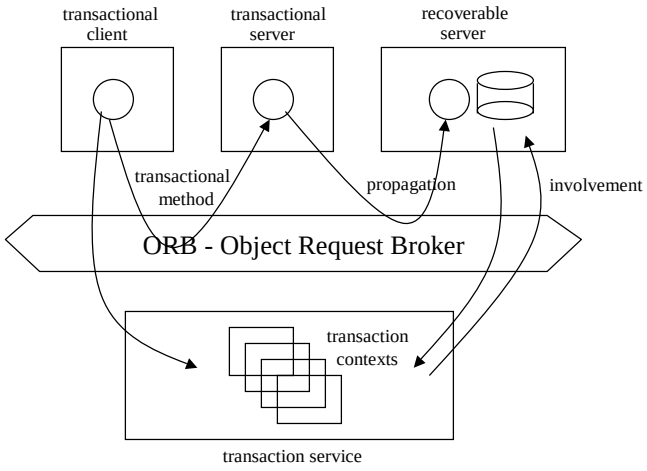
CORBA - motivação para o OTS

- **Suportar crashes do sistema operativo ou dos discos assim como a partilha de dados de forma consistente e robusta:**
  - Atomicidade - todas as modificações são commited ou rolled-back
  - Consistência - uma transacção faz com que os dados passem de um estado consistente para outro
  - Isolamento - transacções executam-se uma forma serializável
  - Durabilidade - efeitos de uma transacção commited não se perdem
- **Transacções que incluem múltiplos objectos que foram definidos separadamente**
- **Transacções que combinam objectos e outros dados**

CORBA - transaction service

- Estende a semântica transaccional em sistemas distribuídos de objectos
- Vários objectos em diferentes ORBs participam em transacções (OTS - Object Transaction Service)
- Aplicações ORB e não-ORB participam numa mesma transacção
- Suporta transacções que abrangem vários ORBs heterogéneos
- Suporta interfaces IDL já existentes (herança da classe transactionalObject)
- Nested transactions são suportadas opcionalmente
- Fortemente relacionado com o concurrency control service
- evolução dos transaction monitors

CORBA - transaction service (cont.)



CORBA - transaction service (cont.)

- **Transactional server:**
  - Colecção de objectos que são afectados pela transacção
  - Não têm estado recuperável associado
  - Não participam no commit mas podem forçar rollback
- **Recoverable server:**
  - Colecção de objectos em que pelo menos um tem estado recuperável
  - Participam no commit (tipicamente 2PC)

CORBA - interfaces do transaction service

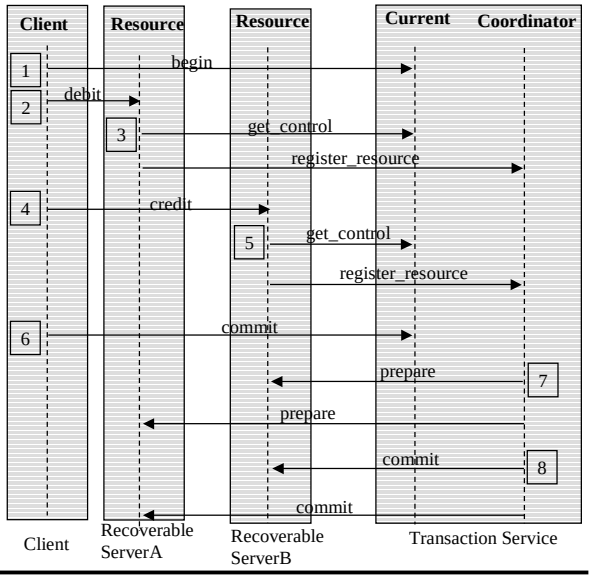
- **Current:**
  - pseudo-objecto CORBA
  - begin, commit, rollback, etc.
- **Coordinator:**
  - implementado pelo transaction service
  - register_resource, create_subtransaction, etc.
- **Resource:**
  - implementado por um objecto no recoverable server
  - prepare, commit, rollback, etc. (2PC)
- **SubtransactionAwareResource:**
  - commit_subtransaction, rollback_subtransaction

CORBA - exemplo de transacção

- Débito da contaA e depósito na contaB
- Os objectos contaA e contaB herdam das classes Resource e TransactionalObject (classe abstracta)
- Os objectos current e coordinator são instâncias das classes homónimas

CORBA - exemplo de transacção

1 - cliente inicia transacção
2 - cliente invoca *debit* no obecto *resource* que foi programado
3 - obtém referência para *coordinator* e regista-se
4 - igual ao ponto 2 mas para o método *credit*
5 - igual ao ponto 3
6 - cliente termina transacção (ORB informa o Transaction Service)
7 - primeira fase do 2PC
8 - segunda fase do 2PC



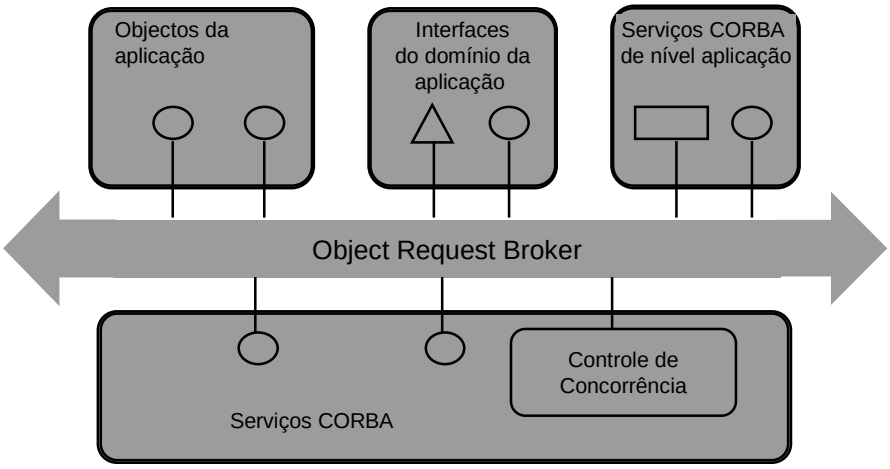
Transaction Boundaries

- A transaction boundary defines the scope of a transaction:
  - Objects inside a transaction boundary share a common transaction identifier.
  - a transaction boundary is an abstraction for managing consistency across process and computer boundaries.
- Your control over a transaction boundary varies depending on the transaction model you select for your application:
  - Manual
  - automatic.

CORBA - concurrency control service

- Permite coordenar o acesso a objectos partilhados
- Baseado no uso de locks (vários modos e granularidade variada)
- Útil quando usado no âmbito de transacções
- Também pode ser usado de forma independente
- É na realidade uma lock managing facility
- Cliente pode adquirir locks em dois modos:
  - No âmbito de uma transacção (locks libertados pelo Transaction Service)
  - Fora de uma transacção (locks libertados pelo cliente)

Exemplo: serviço de controle de concorrência no CORBA



Exemplo: serviço de controle de concorrência no CORBA

- **Suporta 2PL hierárquico**
- **Além de IR, R, IW, W, tem upgrade locks**
- **Upgrade lock (UP) é usado de seguinte modo:**
  - Transacção começa por obter R lock sabendo que mais tarde vai precisar de obter um W lock
  - Quando realmente quiser escrever pede o upgrade lock
- **Locks UP reduzem a possibilidade de deadlock:**
  - Mesma compatibilidade de locks R
  - Locks UP não saõ mutuamente compatíveis
- **Compatibilidade de locks:**

	IR	R	UP	IW	W
IR	+	+	+	+	-
R	+	+	+	-	-
UP	+	+	-	-	-
IW	+	-	-	+	-
W	-	-	-	-	-

Transacções no .Net

Transaction Models

- **For a .NET Framework object to participate in an automatic transaction, the .NET Framework class must be registered with Windows 2000 Component Services:**
  - However, not all transactions are automatic.
  - The activities you perform when programming transactions depend on the transaction model you choose.
  - The common language runtime supports both manual and automatic transaction models.

Manual Transactions

- **The SQL Client and OLE DB .NET providers support manual transactions in the common language runtime:**
  - In ADO.NET, you can use either of these .NET providers to control transactions.
  - Both providers include a set of managed objects that create a database connection, begin a transaction, and commit or roll back the transaction.
  - The major difference between the two is their connection mechanisms:
    - The SQL Client .NET Provider provides a set of objects that call SQL Server directly.
    - By contrast, the OLE DB .NET Provider uses native OLE DB to enable data access.
- **ADO.NET transactions:**
  - are handled entirely inside the database and
  - are unsupported by the Microsoft Distributed Transaction Coordinator (DTC) or any other transactional mechanism.

Example of Manual Transactions

- **The following code example demonstrates transactional logic using ADO.NET with Microsoft® SQL Server™:**

```
SqlConnection myConnection = new SqlConnection("Data Source=localhost;Initial
Catalog=Northwind;Integrated Security=SSPI;");
myConnection.Open();

// Start a local transaction.
SqlTransaction myTrans = myConnection.BeginTransaction();

// Enlist the command in the current transaction.
SqlCommand myCommand = new SqlCommand();
myCommand.Transaction = myTrans;
```

Example of Manual Transactions (cont.)

```
try {
        myCommand.CommandText = "Insert into Region (RegionID,
                        RegionDescription) VALUES (100, 'Description')";
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "Insert into Region (RegionID,
                        RegionDescription) VALUES (101, 'Description')";
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
} catch(Exception e) {
        myTrans.Rollback();
        Console.WriteLine(e.ToString());
        Console.WriteLine("Neither record was written to database.");
} finally { myConnection.Close(); }
```


Manual Transactions and MSMQ

- **Components written in a managed language, such as Microsoft Visual Basic .NET:**
  - can send and receives messages from the Microsoft Message Queuing (MSMQ).
- **MSMQ is a technology that implements message queuing in an application:**
  - With MSMQ, you can create or delete message queues, send or receive messages, and manage message queues. T
  - ransactions are an important part of enterprise systems, which frequently require the asynchronous capabilities of MSMQ.
- **The common language runtime supports manual transactions through the MessageQueueTransaction Class:**
  - MSMQ transactions are handled entirely inside the MSMQ engine and
  - are unsupported by the Microsoft Distributed Transaction Coordinator (DTC) or any other transactional mechanism.

Automatic Transactions

- **Support the same automatic distributed transaction model:**
  - Microsoft Transaction Server (MTS),
  - COM+ 1.0, and
  - the common language runtime
- **Once an ASP.NET page, XML Web service method, or .NET Framework class is marked to participate in a transaction:**
  - it will automatically execute within the scope of a transaction.
  - You can control an object's transactional behavior by setting a transaction attribute value on the page, XML Web service method, or class.
  - The attribute value, in turn, determines the transactional behavior of the instantiated object.

Automatic Transactions (cont.)

- **Based on the declared attribute value, an object will automatically:**
  - participate in an existing or ongoing transaction,
  - be the root of a new transaction, or
  - never participate in a transaction at all.
- **The syntax to declare the transaction attribute varies slightly:**
  - in a .NET Framework class,
  - an ASP.NET page, and
  - a XML Web service method.

Automatic Transactions (cont.)

- **The declarative transaction attribute specifies:**
  - how an object participates in a transaction, and
  - is configured programmatically.
- **Although this declarative level represents the logic of a transaction, it is one step removed from the physical transaction.**
- **A physical transaction occurs when a transactional object accesses a data resource, such as a database or message queue:**
  - The transaction associated with the object automatically flows to the appropriate resource manager.
- **An associated driver, such as OLE DB, Open Database Connectivity (ODBC), or ActiveX Data Objects (ADO):**
  - looks up the transaction in the object's context, and
  - enlists in the transaction through the Distributed Transaction Coordinator (DTC).
  - The entire physical transaction occurs automatically.

Automatic Transactions (cont.)

- **ASP.NET supports automatic transactions**
- **By inserting a transaction directive in your ASP.NET page, you can instruct the page to:**
  - participate in an existing transaction,
  - begin a new transaction, or
  - never participate in a transaction.
- **You can indicate the level of transaction support on a page by placing the directive in your code:**
  - For example, you can ensure that the page activities always execute in the scope of a transaction by inserting the following directive:

```
<%@ Page Transaction="Required" %>
```

  - If you omit the transaction directive, transactions are disabled for the page.

Automatic Transactions (cont.)

Directive	Description
Disabled	Indicates that the transaction context will be ignored by ASP.NET. This is the default transaction state.
NotSupported	Indicates that the page does not run within the scope of transactions. When a request is processed, its object context is created without a transaction, regardless of whether there is a transaction active.
Supported	Indicates that the page runs in the context of an existing transaction. If no transaction exists, the page runs without a transaction.
Required	The page runs in the context of an existing transaction. If no transaction exists, the page starts one.
RequiresNew	Indicates that the page requires a transaction and a new transaction is started for each request.

Automatic Transactions and XML Web Services

- **ASP.NET provides built-in support for creating and exposing XML Web services:**
  - using a programming abstraction that is consistent and familiar to Web Forms.
  - The resulting model is scalable, extensible, and embraces HTTP, XML, SOAP, and WSDL open Internet standards, among others.
  - XML Web services can be accessed and consumed by any client or Internet-enabled device.
- **XML Web services provide to you the option of running your code within the scope of an automatic transaction:**
  - all interactions with resource managers such as SQL Servers, MSMQ Servers, Oracle Servers, and SNA Servers are done within transactions

Automatic Transactions and XML Web Services (cont.)

- You can declare an automatic transaction by using the TransactionOption property of the WebMethod attribute.
- Setting the TransactionOption property to TransactionOption.RequiresNew:
  - begins a new transaction each time a XML Web service client calls the XML Web service method.

Automatic Transactions and XML Web Services (cont.)

```
<%@ WebService Language="C#" Class="Orders" %>
<%@ assembly name="System.EnterpriseServices" %>
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.Services;
using System.Web.Util;
using System.EnterpriseServices;
```

Automatic Transactions and XML Web Services (cont.)

```
public class Orders : WebService {
[ WebMethod(TransactionOption=TransactionOption.RequiresNew)]
public int DeleteAuthor(string lastName) {
    String deleteCmd = "DELETE FROM authors2 where au_lname='" + lastName + "'" ;
    SqlConnection sqlConn = new SqlConnection("user
    id=sa;database=pubs;server=myserver");
    SqlCommand myCommand = new SqlCommand(deleteCmd,sqlConn);
    // If a XML Web service method is participating in a transaction and an
    // exception occurs, ASP.NET automatically aborts the transaction.
    // Likewise, if no exception occurs, then the transaction is
    // automatically committed.
    myCommand.Connection.Open();
    return myCommand.ExecuteNonQuery();
}
```


Automatic Transactions and .NET Framework Classes

- Instances of a .NET Framework class can participate in an automatic transaction
- Each resource accessed by a class instance, or object, enlists in the transaction:
  - For example, if an object uses ADO.NET, the resource manager for the database determines whether the object should execute in a transaction
  - If so, it automatically enlists the database in the transaction.

Automatic Transactions and .NET Framework Classes (cont.)

- **Use the following process to prepare a class to participate in an automatic transaction:**
  1. Apply the TransactionAttribute to your class.
  2. Derive your class from the ServicedComponent Class.
  3. Sign the assembly with a strong name. To sign the assembly using attributes create a key pair using the Sn.exe utility.

     sn -k TestApp.snk Add the AssemblyKeyFileAttribute or AssemblyKeyNameAttribute assembly attribute specifying the name of the file containing the key pair to sign the assembly with a strong name.

     [assembly: AssemblyKeyFileAttribute("TestApp.snk")]
  4. Register the assembly that contains your class with the COM+ catalog.
     1. If the client calling instances of your class is managed by the common language runtime, the registration is performed for you.
     2. However, if you anticipate that an unmanaged caller may create and call instances of your class, use the .NET Services Installation Tool (Regsvcs.exe) to perform the registration manually.

Automatic Transactions and .NET Framework Classes (cont.)

Attribute value	Description
Disabled	Eliminates the control of automatic transactions on the object. An object with this attribute value applied can engage the Distributed Transaction Coordinator (DTC) directly for transactional support. `[Transaction(TransactionOption.Disabled)]`
NotSupported	Indicates that the object does not run within the scope of transactions. When a request is processed, its object context is created without a transaction, regardless of whether there is a transaction active. `[Transaction(TransactionOption.NotSupported)]`
Supported	Indicates that the object runs in the context of an existing transaction, if one exists. If no transaction exists, the object runs without a transaction. `[Transaction(TransactionOption.Supported)]`
Required (default)	Indicates that the object requires a transaction. It runs in the scope of an existing transaction, if one exists. If no transaction exists, the object starts one. `[Transaction(TransactionOption.Required)]`
RequiresNew	Indicates that the object requires a transaction and a new transaction is started for each request. `[Transaction(TransactionOption.RequiresNew)]`

Automatic Transactions and .NET Framework Classes (cont.)

```
// ---------------------------------------------------------------
// TestApp.cs
// Generate a Strong name:
//          sn -k TestApp.snk
// Compile the code:
//          csc /target:exe /r:System.EnterpriseServices.dll TestApp.cs
// Run TestApp:
//          start TestApp.exe
// ---------------------------------------------------------------
using System; using System.Runtime.CompilerServices;
using System.EnterpriseServices; using System.Reflection;

//Registration details.
//COM+ application name as it appears in the COM+ catalog.
[assembly: ApplicationName("TestApp")]
//Strong name for assembly.
[assembly: AssemblyKeyFileAttribute("TestApp.snk")]
```


Automatic Transactions and .NET Framework Classes (cont.)

```
[Transaction(TransactionOption.Required)]
public class Account : ServicedComponent {
    //Provides SetComplete behavior in the absence of exceptions.
    [AutoComplete]
    public void Debit(int amount) {
        // Do some database work. Any exception thrown here aborts the
        // transaction; otherwise, transaction commits.
    }
}
public class client {
    public static int Main() {
        Account accountX = new Account();
        accountX.Debit(100);
        return 0;
}
```

Voting in an Automatic Transaction

- .NET Framework classes and ASP.NET pages can vote to commit or abort their current transaction:
  - The absence of an explicit vote in your code casts a commit vote by default.
  - The default commit, however, may decrease the performance of your application by lengthening the time it takes for each transaction to release expensive resources.
- Explicit voting also allows your class or page to abort a transaction if it encounters a significant error:
  - you can improve your application's performance by catching a fatal error early in the transaction, ending the transaction, and releasing resources.

AutoComplete

- System.EnterpriseServices.AutoCompleteAttribute causes:
  - an object participating in a transaction to vote in favor of completing the transaction if the method returns normally.
  - If the method call throws an exception, the transaction is aborted.
  - You can apply this attribute only to classes deriving from the ServicedComponent class.

```
[Transaction(TransactionOption.Supported)]
public class Account : ServicedComponent {
        [AutoComplete] public void Debit(int amount) {
                // Do some database work. Any exception thrown here
                // aborts the transaction; otherwise, transaction commits.
        }
}
```

SetAbort e SetComplete

- The System.EnterpriseServices.ContextUtil class:
  - exposes the SetComplete or SetAbort methods,
  - to explicitly commit or abort a transaction
  - SetComplete indicates that your object votes to commit its work;
  - SetAbort indicates that your object encountered a problem and votes to abort the ongoing transaction.
- A transaction is neither committed nor aborted until the root object of the transaction deactivates.
- Further, a single abort vote from any object participating in the transaction causes the entire transaction to fail.

SetAbort e SetComplete (cont.)

```
//Try to do something crucial to the transaction in progress.
if( !DoSomeWork() ) {
   //Something goes wrong.
   ContextUtil.SetAbort();
}
else {
   //All goes well.
   ContextUtil.SetComplete();
}
```

Transacções no Java

Transacções na Plataforma J2EE

- O suporte transaccional é um elemento essencial da arquitectura
- O fardo da implementação da gestão das transacções encontra-se na plataforma.
- O servidor implementa os protocolos de baixo nível tais como:
  - a interacção entre o Transaction Manager e os sistemas de base de dados,
  - propagação do contexto da transacção,
  - two-phase commit
- Os beans só devem decidir se a transacção faz commit ou abort

Transacções na Plataforma J2EE (cont.)

- A plataforma J2EE suporta 2 tipos de demarcação de transacções:
  - Programática (ou manual)
    - Lógica transaccional no código da aplicação
    - Utilização da Java Transaction API
  - Declarativa (ou automática)
    - Os beans nunca emitem instruções explicitamente
    - As transacções são iniciadas e completadas automaticamente pelo container

Bean-Managed Transaction Demarcation

- Apenas os session beans podem escolher este tipo de demarcação
- A UserTransaction interface permite demarcar os limites da transacção:

```
UserTransaction ut = ejbContext.getUserTransaction();
ut.begin();
// perform transactional work here
ut.commit();
```

Container-Managed Transaction

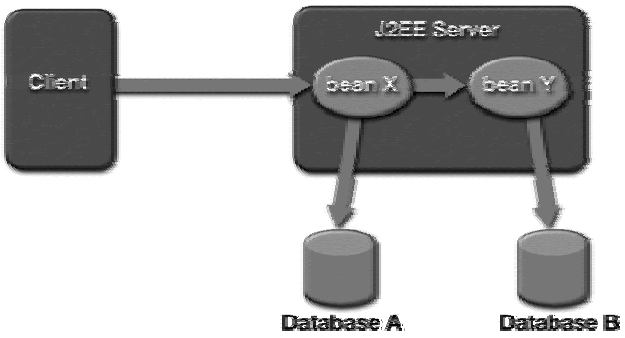
- **Os entity beans têm que escolher este tipo de demarcação**
  - Os atributos transaccionais permitem controlar o envolvimento de um bean numa transacção.
  - São especificados no deployment descriptor:
    - Required,
    - RequiresNew,
    - NotSupported,
    - Supports,
    - Mandatory,
    - Never
  - O mesmo atributo pode ser especificado para todos os métodos ou um diferente para cada um.
  - O Bean continua a ter controlo sobre a transacção:
    - Por exemplo pode fazer rollback de uma transacção iniciada no container (context.setRollbackOnly())

Atributos das Transacções

Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	none	T2
	T1	T1
RequiresNew	none	T2
	T1	T2
Mandatory	none	error
	T1	T1
NotSupported	none	none
	T1	none
Supports	none	none
	T1	T1
Never	none	none
	T1	error

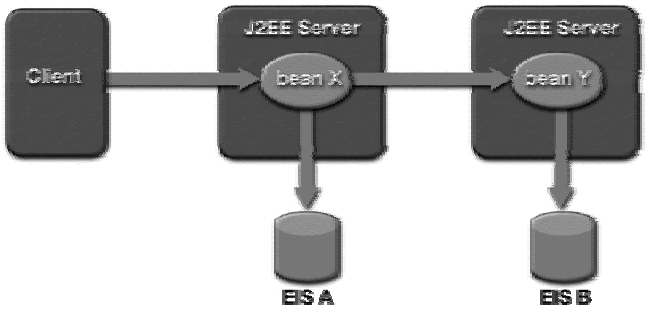
Transacções Distribuídas

- **Acesso a múltiplas Bases de Dados**
  - O servidor J2EE e os adaptadores para as duas bases de dados asseguram que os updates são todos committed ou todos rolled back.



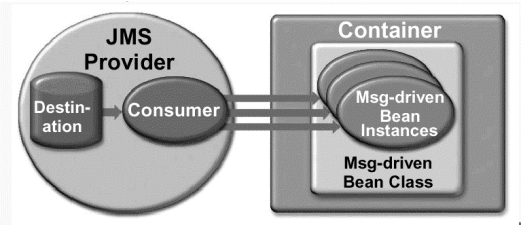
Transacções Distribuídas (cont.)

- **Acesso a partir de múltiplos EJB Servers**
  - Os 2 servidores cooperam para propagar o contexto de X para Y e usam um protocolo two-phase commit para assegurar que os 2 sistemas são actuallizados na mesma transacção



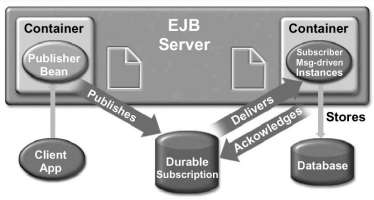
Message Driven Beans

- Novo tipo: assíncronos e sem estado
- Invocados na recepção de uma mensagem JMS
- Não possuem interface Home ou Remote
- Suportam processamento de mensagens concorrentes.



Transacções Distribuídas

```
public class SubscriberMsgBean implements MessageDrivenBean {
private transient MessageDrivenContext mdc = null;
/* Message acknowledge and database update participate
* in distributed txn.
*/
    public void onMessage( Message inMessage) {
        TextMessage msg = (TextMessage) inMessage;
        try {
            < look up JDBC database >
            < store info from message in database >
        } catch( Exception e) {
            mdc. setRollbackOnly();
        }
    }
}
```



Resumo

- **Fundamentos:**
  - modelo
  - falta de objecto
  - Transparência
  - *Persistent State Service* (arquitectura, PDL)
  - Suporte de persistência (ficheiros, bases de dados)
  - Serialização
  - Gestão de versões dos dados serializados
  - Acesso a bases de dados
  - Mapeamento objecto-relacional
  - Transacções
- **Exemplos:**
  - Serialização em CORBA, Java, .Net
  - Enterprise Java Beans, ADO, JDO
  - Transacções no Java e no .Net