Doctoral course: object technologies

IMO Group
Department of computer science
University of Vigo

J. Baltasar García Perez-Schofield
http://webs.uvigo.es/jbgarcia/

Persistence

Persistence

- Persistence is the term employed for desig-ning the storing and retrieval of data. In ob-ject-oriented programming languages, this data are objects.
- Many applications follow a very simple way of working: they restore data from a previous session, process them, and store them.

Terminology

- *Serialization*: save the state of the object directly, as a sequence of bytes.
- *Swizzling*: converting pointers, from their format in memory to their format on disk, and viceversa.
- *Activation*: Retrieve an object stored on disk.
- *Pasivation*: Storing on disk of an object in memory.

Persistence

- The recovery process is known as *unflattening*, while the storing process is called *flattening.*
- What is really happening in both processes is a *decoding* and a *coding* process, respectively.
- Why not directly save the needed objects, and re-cover them when necessary?

Object storing

- Depending on the programming language:
  - JAVA: it has a serializing process to disk, more or less automatic.
  - C++: it does not have any serialization mechanism, you just can use the average record serialization:
    - `fwrite( &object, sizeof( object ), 1, file );`

Object retrieval

- In programming languages such as C++, it is only possible to recover the state of an object stored prevously; however, it is not known to what class the object pertains, or even whether it is an object.
- In programming languages such as Java, the reco-very is slightly better, since we can obtain the .class archive and the stored state archive; however, the recovery and management of these files is only simple if the same application that sto-red them is the one that is going to use them.

Persistence support: difficulties in current object-oriented program-ming languages

- The most of them allow you to serialize the state of an object to a file on disk.
- However, simple serialization of the state is not enough in order to work with a stored object. Seria-lization is needed in order to obtain persistence, but it is not persistence by itself.
- Most of these serialization mechanisms, including the one present in Java, are in practice limited to simple objects (i.e., objects that do not store refe-rences to other objects), and not complex ones.

The true nature of persistence

- Research in persistence tries to go one step further from the idea of just storing objects in files (in the same fashion that other data is stored).
- The objective is to build an storing and retrieval mechanism, as automatic and transparent as possible, making obsoletes the concepts of:
  - File: is not needed any more.
  - Distinction between primary (RAM) and secondary (disk) memory.

Brief historic survey

- Object-oriented databases begun to be developed and researched at the end of the 70's.
- Since then, the concept of persistence went from the database field to the operating systems field, and finally, to the programming languages field.
- Research in persistence lost its strength in mid 90's, and it was absorbed by research in Aspect Oriented Programming.

Orthogonal persistence

- An object can be persistent, regardless its type (i.e., the class it pertains to).
- Objects should be managed homogene-ously, regardless whether they are persis-tent or not.
- The decision of whether an object is persis-tent or not is made by the system. It just cheks whether it is reachable (by following references) from a persistent root or not.

Persistence: why did not it triumph?

- Backwards compatibility: a lot of running applications are based on a file system.
- Change of programming style: the trustworthy file concept disappears, becoming a different programming fashion.
- Performance: persistent systems are not so efficient as traditional ones.

Mechanisms needed for persist-ence

- Swizzling (pointer conversion)
- Clustering (object grouping in the persistent store)
  - How is it possible to group objects in the persist-ent store, so when a cluster is read, all (or most of them) the related objects are in memory for the next processes?
- Schema evolution
  - How to react when a class changes?
- Object caching.
- Integrity of the persistent store.

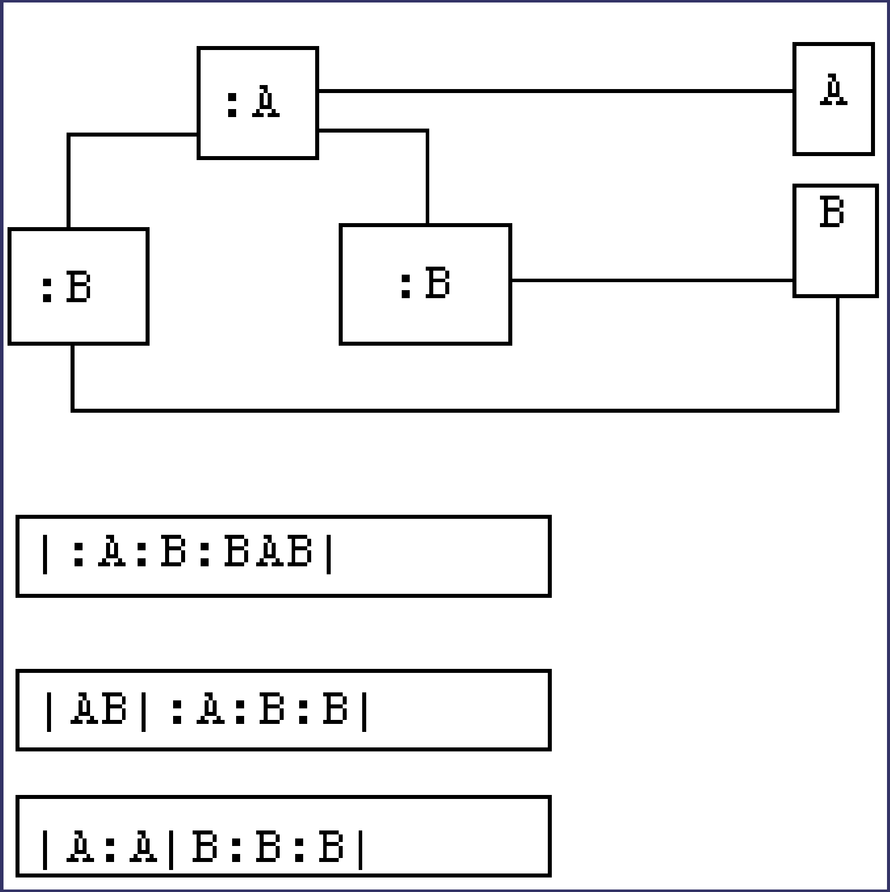
Clustering

- *Clustering* or grouping, is about how to store together objects in the persistent store.
- In relational databases, a record is never read in read operation, but a cluster (group) of them. This is also appliable to average operating system files.
- This way, it is expected that the read objects are going to be the ones used in later operations, without the need to read from disk.

Clustering

- Thus, objects are stored in clusters, so when an object of this cluster is read, the entire cluster is brought to primary memory, and when an object is modified, the entire cluster is written to disk.
- The key point would be to find a grouping policy minimizing the number of reads needed in order to work with that objects.

Clustering



- We should find an ideal grouping implying the minimal number of disk accesses.
- With the first policy, there is only one cluster involved.
- In the second one, two clusters.
- ... as well as two clusters in the third one.

Clustering

- The first policy (putting all classes and objects altogether) is the best one, however it is not possible to group all objects in one cluster in a real world application.
- The second policy stores all classes in a cluster and all objects in another one. It is not very feasible as well.
- The third one is very common, and it has to do with the moment of the creation of objects: each class is saved with its objects.

Clustering

- There are some adaptative technics, that group objects based on statistical calculus accounting which objects are used together.
- While these techniques always give the best possible clustering, it has been demon-strated that they are very slow.

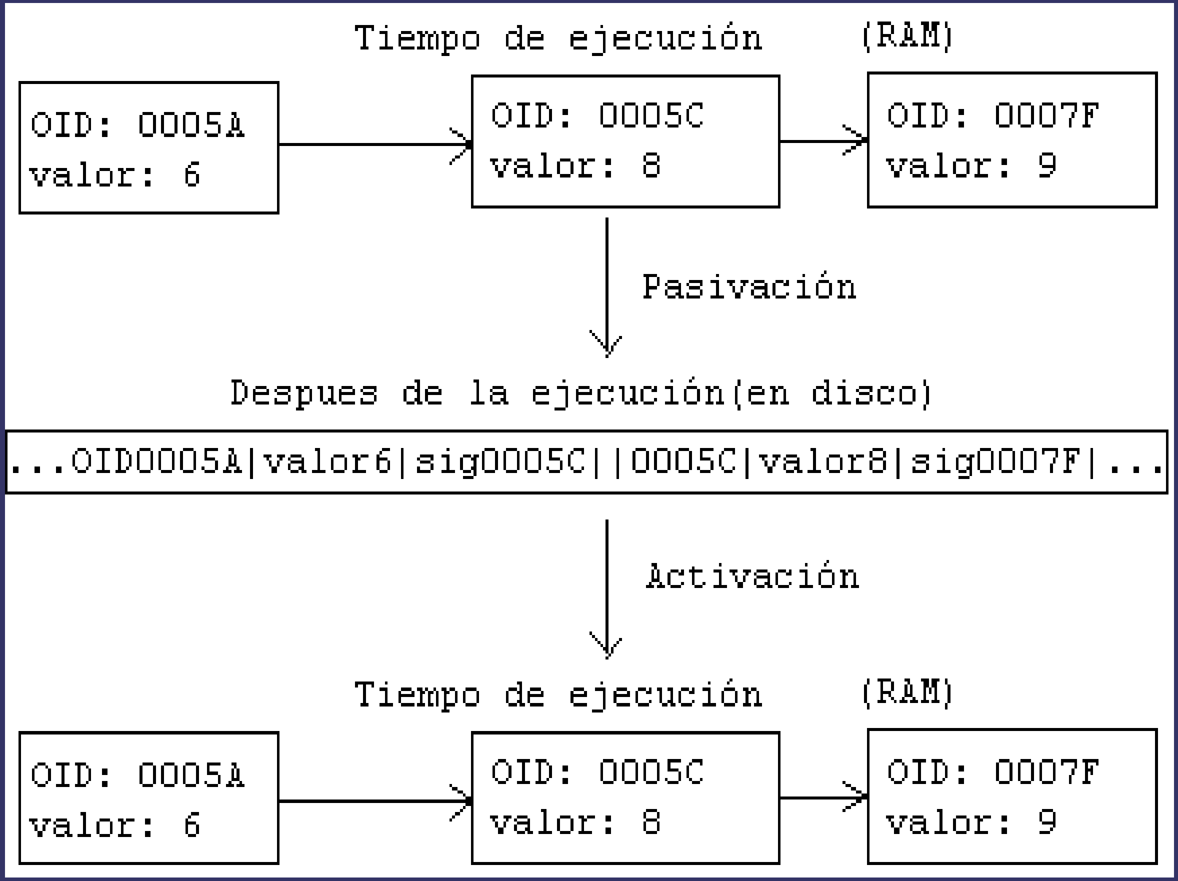
Clustering

- An intermediate solution is to ask the user which objects should be grouped together.
  - The ODMG 3.0 standard includes syntax to let the user manage the grouping policies. This is not transparent at all.
  - Other systems just group those objects that were created in the same session.
  - Barbados uses a metaphor of directories, in which each container is a directory. Users organize their objects in directories, and, this way, they are transparently managing the clustering policy.
  - Zero uses containers that can be, as Barbados, identified with folders, but that behave as collections of objects.

Swizzling

- *Swizzling,* means to translate pointers from its natural form, in memory, to a codified form in disk, and viceversa.
- A pointer (in C++, or a reference in Java) is normally substituted by an OID (Object Identifier), so the object structure in memory can be rebuilt.
- There are two basic strategies for converting pointers in the recovery stage:
  - Eager
  - Lazy

Swizzling


Tiempo de ejecución (RAM)
OID: 0005A valor: 6
OID: 0005C valor: 8
OID: 0007F valor: 9
Pasivación
Despues de la ejecución(en disco)
...OID0005A|valor6|sig0005C||0005C|valor8|sig0007F|...
Activación
Tiempo de ejecución (RAM)
OID: 0005A valor: 6
OID: 0005C valor: 8
OID: 0007F valor: 9

- When objects are stored, their pointers are substituted by OID's.
- The reverse process happens when those objects are reloaded in memory.

Swizzling eager/lazy

- Eager
  - When an object is loaded, all its pointers are swizzled.
    - Barbados employs a mixed system: when a container is loaded, all references inside it are converted. However, the container is just a part of the persistent store.
- Lazy
  - Objects are loaded and only the minimal number of references are swizzled. Only when those references are going to be used, are swizzled.
    - Oberon-D uses lazy swizzling. When a reference raises an error of “object not found”, Oberon-D examines whether it is unswizzled. In the later case, swizzles it, unmarks the error, and resumes execution.

Schema Evolution

- It is the same problem that happens when in a relational database a table is changed: all its records must be adapted.
- In an object-oriented database (or a persist-ent store), this problem happens when a class changes, i.e., it is modified. All its ob-jects must be adapted as well.

Schema Evolution

- There are also two possible policies for schema evolution, very similars to the swizz-ling mechanism.
  - **Eager**: All objects are adapted at the time the modification of the class is detected.
  - **Lazy**: Objects are adapted as long as they are used.

Schema Evolution

- Eager:
  - PJama has a command line tool that accepts a *.class* file and a text file describing the adaptation for its objects, and runs all over the persistent store making the necessary changes.
    - This tool allows programmers to apply very complex schema evolutions, but it implies that the persistent store cannot be in use while the tool is running.
    - The main benefit is that, once the tool is finished, the conversion has also been made and all objects are synchronised.

Schema Evolution

- Lazy:
  - O_2 takes note about every conversion in any class, and converts objects when they are used, but not before.
    - This means that it supports a versioning system, which keeps trace of every single modification in a class, and applies all adaptations for each modification in order, when an object of that class is used.
    - This is an extremely complex system, which makes various objects of different versions of the same class coexist in the same persistent store.
    - The main advantage is that the system is never offline, and that objects are only converted when used, so the conversion does not have to stop the system.

Schema Evolution

- A mixture between eager and lazy:
  - In Barbados, when a class is changed, all objects of that class pertaining to the container currently in use are changed. The remaining objects in other containers are pending of change. When a container is loaded, the objects of that class are converted.
  - It tries to get the advantages of both systems: it does not have to put the system offline, but the objects in the same container are synchronised.

Applications of persistence

- OOOS (Object-Oriented Operating Systems):
  - EROS (http://www.eros-os.org/)
  - GRASSHOPPER (http://www.gh.cs.su.oz.au/Grasshopper/)
- OOPPS (Object-Oriented Persistent Programming Systems)
  - Barbados (http://www.lsi.uvigo.es/lsi/erosello/imo/Imospain/pers.html)
  - Pjama (http://www.dcs.gla.ac.uk/pjava/)
  - Oberon-D (http://www.ssw.uni-linz.ac.at/Research/Projects/OberonD.html)
- OODBMS (Object-Oriented Database Management Systems):
  - O_2 (http://www.dbis.informatik.uni-frankfurt.de/REPORTS/GOODSTEP/goodstep.html)

References

- Orthogonal persistence
  - Atkinson M.P., Morrison R. (1995). “Orthogonality Persistent Object System”, *VLDB Journal* v4 n3, pp319-401, ISSN: 1066-8888
  - The first publication trying to state three basic prin-ciples for *standard* persistence.
  - Orthogonality (independence) of:
    - type
    - management
    - designation of persistent objects

References

- ODMG 3.0
  - http://www.odmg.org
  - Cattel R., Barry D., (eds.), (2003). "*The Object Data Standard: ODMG 3.0*". Morgan Kaufmann Publishers. ISBN 1-55860-647-4
  - ODMG is the stadnard followed by many firms that sell relational databases with abstraction layers based in objects, such as *ORACLE*.
  - There are also JDBC implementations of this standard for Java.

References

- PJama
  - Sun research:
    - http://www.sunlabs.com/forest/index.html
  - University of Glasgow:
    - http://www.dcs.gla.ac.uk/pjava/
  - PJama, or persistent Java, was a project finisedh in September 2000, trying to give persistence support for Java.
  - It was directed by historical experts in persist-ence.
  - It is not completely orthogonal (it violates the second and third rule), as it tries to keep back-wards compatibility with existing Java programs.

Doctoral course: object technologies

IMO Group
Department of computer science
University of Vigo

J. Baltasar García Perez-Schofield
http://webs.uvigo.es/jbgarcia/