Curso de Doctorado: Tecnologías de Objetos

Grupo IMO
Área de Lenguajes y Sistemas Informáticos
Departamento de Informática

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

Persistencia

Persistencia

- Persistencia es el término utilizado para la salvaguarda y recuperación de datos, concretamente, de objetos.
- La mayoría de las aplicaciones siguen una estructura sencilla de funcionamiento: recuperación de datos de una anterior ejecución, procesado, y salvaguarda de los nuevos datos.

Terminología

- *Serialización*: guardar el estado de un objeto directamente como una secuencia de bytes en un archivo en disco.
- *Swizzling*: conversión de punteros, de su formato en disco a su formato en memoria, y viceversa.
- Activación: Carga en memoria de un objeto que reside en disco.
- Pasivación: Salvaguarda en disco de un objeto que reside en memoria.

Persistencia

- El proceso de recuperación se denomina *unflattening*, desaplanar, y el de salvaguarda *flattening* (aplanar).
- Lo que está sucediendo en realidad en ambos procesos es una codificación de los datos contenidos en los objetos de la aplicación a un fichero, y viceversa.
- ¿Por qué no guardar directamente los objetos, y recuperarlos cuando sea necesario?

Salvaguarda de objetos

- Depende del lenguaje en el que se esté trabajando:
  - JAVA: posee un sistema de serialización de objetos a disco, más o menos automático.
  - C++: no posee ningún sistema de serialización, utilizándose mecanismos que tienen como base la grabación directa de objetos a disco:
    - `fwrite(&objeto, sizeof(objeto), 1,Fichero)`

Recuperación de objetos

- En los lenguajes como C++, sólo puede recuperarse el estado de un objeto guardado previamente; sin embargo no puede saberse a qué clase pertenece el objeto, y ni siquiera si es un objeto.
- En lenguajes como Java, la recuperación es un poco mejor, puesto que podemos obtener el archivo .class y el de datos; sin embargo, la recuperación y el tratamiento de estos archivos sólo es realmente sencillo si es la misma aplicación que los grabó la que los va a utilizar.

Persistencia: dificultades en los lenguajes orientados a objetos actuales

- La mayor parte de los lenguajes permiten, hoy en día, **serializar** el estado de un objeto a un archivo en disco.
- Sin embargo, la simple serialización del estado no es suficiente para poder trabajar con un objeto guardado. La serialización es necesaria para obtener persistencia, si bien la serialización por sí misma no es persistencia.
- La mayoría de estos mecanismos, incluyendo el de Java, están en la práctica limitados a objetos que contienen datos “directos”, siendo incapaces de manejar objetos compuestos (complejos).

La verdadera naturaleza de la persistencia

- La investigación en persistencia trata de ir un paso más allá que la idea inicial de guardar objetos en archivos de la misma forma que es posible guardar todo tipo de datos en ellos.
- Se trata de proporcionar un mecanismo tan automático como sea posible para la recuperación y salvaguarda de objetos, resultando por tanto obsoletos los conceptos de:
  - Archivo: ya no es necesario.
  - Distinción entre memoria primaria y secundaria: ya no es necesaria.

Breve revisión histórica

- Las bases de datos orientadas a objetos empiezan a ser investigadas y desarrolladas a finales de los años 70.
- Desde entonces, empieza a trasladarse el concepto de persistencia a los SSOO y a los lenguajes de programación.
- La persistencia pierde buena parte de su fuerza investigadora mediados los 90.

Persistencia Ortogonal

- Un objeto puede ser persistente, independientemente de la clase a la que pertenece.
- Los objetos deben ser tratados homogéneamente, sean persistentes o no persistentes.
- La decisión de si un objeto es persistente la toma el sistema, en base a un mecanismo de alcanzabilidad desde una raíz persistente.

Persistencia: ¿por qué no triunfó?

- Compatibilidad hacia atrás: un excesivo número de sistemas se basan en un sistema de ficheros.
- Desconfianza: desaparece el fichero para ser sustituido por una parte del sistema que lo gestiona sin poder intervenir.
- Rendimiento: los sistemas persistentes no son tan eficientes como los sistemas no persistentes.

Mecanismos y técnicas necesarias para persistencia

- Swizzling (conversión de punteros)
- Clustering (agrupación de objetos en el almacenamiento persistente)
  - ¿Cómo agrupar los objetos para que al leer cada grupo (*cluster*), leamos los necesarios para que los siguientes procesos no supongan acceso al disco?
- Schema evolution (evolución del esquema)
  - ¿cómo reaccionar cuando una clase cambia?
- Técnicas de caché de objetos
- Seguridad

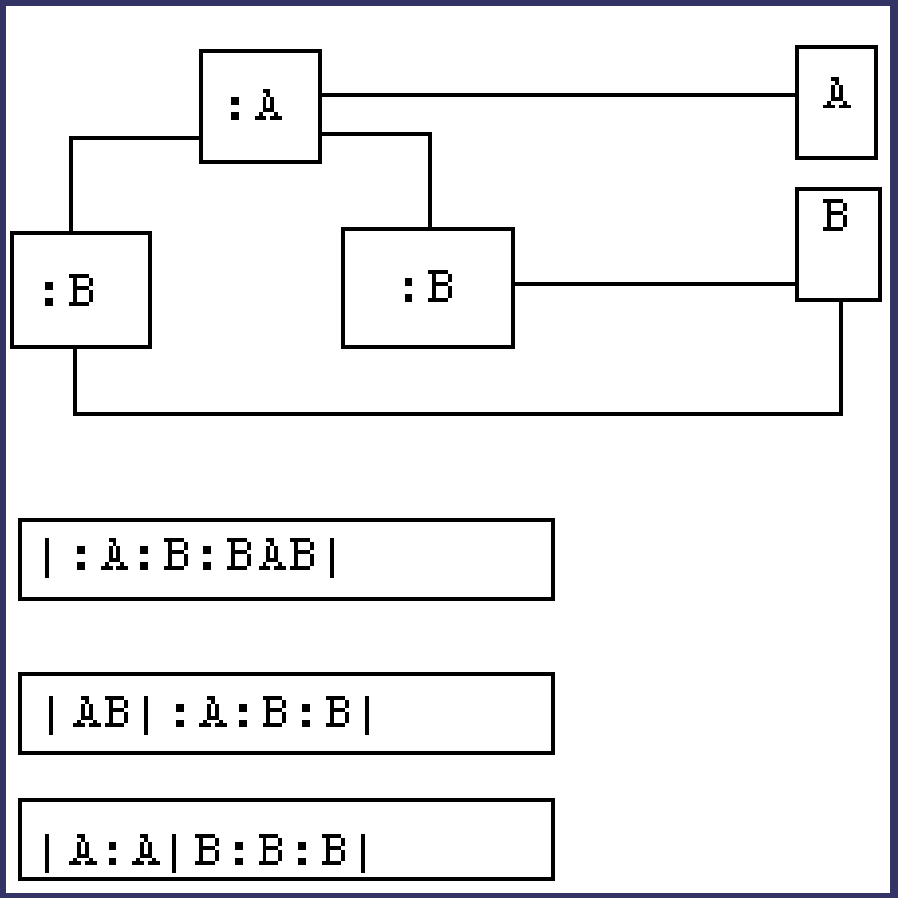
Clustering

- El *clustering* o agrupamiento, trata de cómo agrupar los objetos en el almacenamiento persistente.
- En las bases de datos relacionales nunca se lee realmente un registro en una operación de lectura, e incluso en ficheros normales del sistema operativo, nunca se leen un byte o cuatro bytes, sino que se lee en bloques de un tamaño predeterminado.
- De esta forma, se espera que los datos ya en memoria satisfagan posteriores operaciones de lectura **sin** recurrir al disco.

Clustering

- Por tanto, se agrupan los objetos en bloques, de forma que cuando se guardan se guarda ese bloque completo, y cuando se recuperan, se recupera también ese bloque completo.
- El punto clave es establecer una política de agrupamiento que minimice el número de lecturas necesarias para trabajar con esos objetos.

Clustering



- Deben ser necesarias las menos operaciones de lectura posibles para trabajar con esos objetos.
- En la primera, sólo es necesario cargar 1 cluster.
- En la segunda, dos clusters.
- En la segunda, también dos clusters.

Clustering

- Con la primera política de clustering, sólo es necesario leer un *cluster*. Sin embargo, no es posible colocar todos los objetos del almacenamiento persistente en un solo *cluster*.
- Con la segunda política, se guardan las clases en un *cluster* y los objetos en otro.
- La tercera es bastante habitual, y tiene que ver muchas veces con el momento de creación de los objetos: se trata de colocar en el mismo cluster a una clase y a sus objetos.

Clustering

- Existen algunas técnicas adaptativas, que mediante cálculos de tipo estadístico, colocan a los objetos en sus *clusters* correspondientes según cuántas veces se utilicen juntos.
- Estas técnicas estadísticas han demostrado ser muy lentas.

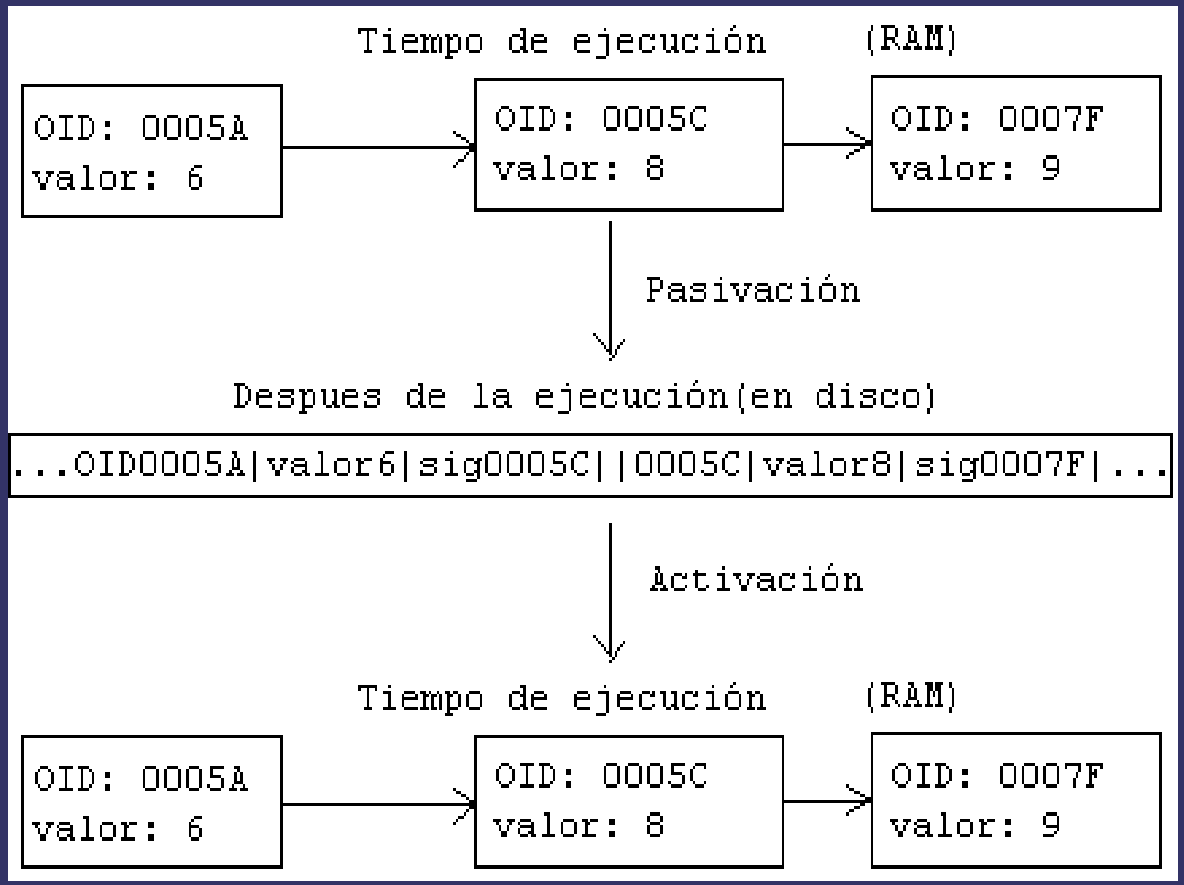
Clustering

- Una solución de compromiso es dejar que le usuario indique cuándo dos o más objetos deben guardarse en el mismo *cluster*.
  - El estándar ODMG 3.0 añade sintaxis que permite especificar esta circunstancia. Es muy poco transparente.
  - Barbados utiliza una metáfora de directorios, de forma que cada contenedor es un directorio. Así, el mismo usuario escoge el *directorio* donde va a colocar los objetos, y por tanto el *clustering* de los mismos, igual que en directorios en disco, de archivos.

Swizzling

- El *swizzling,* (término intraducible), trata de la conversión de punteros entre su formato en disco y su formato (natural) como dirección de memoria, y viceversa.
- El puntero (como la referencia de Java o el puntero, propiamente, de C++) suele sustituirse por un OID (OBject Identifier), para de esa forma restaurarlo cuando se cargue de nuevo el objeto.
- Existen dos estrategias básicas para convertir los punteros en la fase de carga:
  - Eager (o temprana, o incluso deseosa).
  - Lazy (o tardía, o incluso, perezosa).

Swizzling



- Cuando se guardan objetos, se sustituyen sus punteros por OID's.
- El proceso contrario sucede cuando se cargan esos objetos en memoria.

Swizzling eager/lazy

- Eager (temprano, deseoso)
  - En cuanto se cargan uno o más objetos, todas sus referencias son resueltas.
    - Barbados utiliza un sistema mixto: cuando un contenedor es cargado, todas las referencias dentro de él son convertidas. Sin embargo, el contenedor es sólo una parte del almacenamiento persistente.
- Lazy (tardío, perezoso)
  - Los objetos son cargados y tan sólo las referencias mínimas son resueltas (quizás, ninguna). Sólo cuando las referencias son empleadas, éstas son convertidas.
    - Oberon-D utiliza un sistema de swizzling lazy. Cuando una referencia provoca un error de “objeto no encontrado”, Oberon-D examina si está sin convertir (unswizzled). En tal caso, la convierte y continúa la ejecución.

Evolución del esquema
(Schema Evolution)

- Se trata del mismo problema que sucede cuando en una base de datos relacional se cambia una tabla: todos los registros de esa tabla deben ser adaptados.
- En una base de datos orientada a objetos, o en un almacenamiento persistente, éste problema sucede cuando una clase cambia, es decir, es modificada. Todos sus objetos deben cambiarse también.

Evolución del esquema

- También existen dos políticas de evolución del esquema, con las mismas premisas que en el mecanismo de swizzling.
  - Eager: Todos los objetos son convertidos en el mismo momento en el que se detecta la modificación de la clase.
  - Lazy: Los objetos de la clase modificada son convertidos a medida que van siendo utilizados.

Evolución del esquema

- Eager:
  - PJama proporciona una solución *eager* a este problema: existe una herramienta de línea de comando que se encarga de, pasándole el archivo *.class* de la nueva clase, y una función de conversión, sustituir la clase en cuestión y aplicar la función de conversión a todos los objetos afectados. La función de conversión tiene la ventaja de permitir transformaciones complejas entre objetos anteriores y objetos nuevos. Lo mejor de este método es que todos los objetos son convertidos una vez finalizado el trabajo de la herramienta. Lo peor es que, mientras se ejecute está conversión de objetos, el sistema no podrá ser usado.

Evolución del esquema

- Lazy:
  - O_2 proporciona una solución *lazy* a este problema: cada transformación de la case es anotada, y a medida que se carguen los objetos de esa clase, los objetos son convertidos. La mayor ventaja es que la conversión de objetos no supone que el sistema precise de estar fuera de uso. La mayor desventaja es que conviven objetos de diferentes *versiones* de las clases en el almacenamiento persistente, y que es posible que una clase sea modificada varias veces antes de que todos sus objetos sean convertidos, por lo que será necesario guardar todas las transformaciones que haya sufrido la clase.

Evolución del esquema

- Intermedio eager/lazy:
  - Barbados proporciona una solución intermedia a este problema: cuando una clase cambia, todos los objetos de esa clase que existan en su mismo contenedor son convertidos. A medida que los contenedores con objetos de esa clase son cargados, los objetos van siendo convertidos, pudiendo definirse una función de conversión. Trata de aunar las ventajas de ambos sistemas: no todos los objetos del almacenamiento persistente son convertidos, sino sólo una parte cada vez. Por otro lado, todos los objetos están listos para ser usados una vez que el contenedor ha sido cargado.

Aplicaciones de persistencia

- OOOS (Object-Oriented Operating Systems): Sistemas operativos orientados a objetos.
  - EROS (http://www.eros-os.org/)
  - GRASSHOPPER (http://www.gh.cs.su.oz.au/Grasshopper/)
- OOPPS (Object-Oriented Persistent Programming Systems): Sistemas de programación orientada a objetos persistentes.
  - 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): Sistemas gestores de bases de datos orientadas a objetos.
  - O_2 (http://www.dbis.informatik.uni-frankfurt.de/REPORTS/GOODSTEP/goodstep.html)

Bibliografía

- Persistencia ortogonal
  - Atkinson M.P., Morrison R. (1995). “Orthogonality Persistent Object System”, *VLDB Journal* v4 n3, pp319-401, ISSN: 1066-8888
  - Se trata de la primera publicación que asienta unas bases (tres, en concreto), para cualquier tipo de implementación de persistencia en cualquier sistema.
  - Ortogonalidad (independencia) de:
    - tipo
    - trato
    - designación de objetos persistentes

Bibliografía

- 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 es el estándar que siguen muchas empresas que venden bases de datos relacionales con capas de abstracción de objetos, como *ORACLE*.
  - También hay implementaciones tipo JDBC para Java que siguen este estándar.

Bibliografía

- PJama
  - Sun research:
    - http://www.sunlabs.com/forest/index.html
  - Universidad de Glasgow:
    - http://www.dcs.gla.ac.uk/pjava/
  - PJama, o Java persistente, fue un proyecto terminado en Septiembre del 2000 para soportar persistencia ortogonal en Java.
  - Fue dirigido por históricos investigadores de la persistencia
  - No es completamente ortogonal (viola la segunda y la tercera regla), ya que se buscaba compatibilidad hacia atrás con los programas existentes en Java.

Curso de Doctorado: Tecnologías de Objetos

Grupo IMO
Área de Lenguajes y Sistemas Informáticos
Departamento de Informática

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