El Sistema de Persistencia en Oviedo3

Ortín Soler Francisco, Álvarez Gutiérrez Darío, Izquierdo Castanedo Raúl, Martínez Prieto Ana Belén, Cueva Lovelle Juan Manuel

Área de Lenguajes y Sistemas Informáticos. Departamento de Informática.

Universidad de Oviedo.

C/Calvo Sotelo, 33007.

Teléfono: (98)5103286. Fax: (98)5103354

E-Mail: (ortin, darioa, ric,belen, cueva)@pinon.ccu.uniovi.es

Resumen

Este artículo describe el sistema de persistencia desarrollado en un proyecto integral orientado a objetos. Todo el proyecto Oviedo3 se centra en el concepto de una máquina abstracta orientada a objetos a la cual se le añade la condición de poder crear instancias persistentes. Se identificará pues el proyecto global y la máquina abstracta y se describirá el análisis y diseño efectuados para implementar el sistema de persistencia.

Palabras Clave

Persistencia, Oviedo3, máquina abstracta, memoria virtual, Carbayonia.

1. Introducción

En este artículo se describe el sistema de persistencia para el proyecto de investigación Oviedo3. Este proyecto se basa en la planificación y desarrollo de un sistema integral orientado a objetos [CUEVA96]. Se identifican distintas capas (ver Figura 1) siendo el núcleo de todas ellas una máquina abstracta orientada a objetos[ALVAREZ97]. Todo el código de las distintas capas de Oviedo3 será el código binario de esta máquina, centrando así la portabilidad del sistema en el desarrollo del simulador de la máquina abstracta orientada a objetos bajo distintas plataformas.

El contenido de este artículo se centra en el estudio del sistema de persistencia para Oviedo3[ALVAREZ96]. Este sistema ha de enmarcarse dentro de una de las capas del proyecto global. Se considera pues la persistencia como una característica propia de la máquina abstracta orientada a objetos (denominada Carbayonia). No se tratará pues de un nivel más en las capas del proyecto global, ni de un módulo aparte de la máquina abstracta, sino que se deberá ampliar la funcionalidad de ésta para que sea capaz de ofrecer objetos persistentes.

2. Niveles de persistencia

Debemos especificar que se entiende por persistencia enmarcarla dentro de nuestro entorno de Oviedo3 y más concretamente de la máquina abstracta orientada a objetos Carbayonia.

El primer nivel de persistencia es el asegurar la supervivencia de los datos de un programa a la propia ejecución del programa [BOOCH94]. Si aplicamos esto a un lenguaje orientado a objetos, podremos entender que posee esta propiedad si es posible que los objetos creados en este programa puedan persistir una vez finalizado el mismo. A continuación se definen los siguientes términos [TUNES96]:

- Persistencia: Propiedad por la que la información del sistema puede persistir durante el tiempo que sea requerida.

- Ortogonal: Toda la información puede ser persistente y debe ser manipulada de la misma manera, independientemente del tiempo que deba persistir.

- Completa: Cuando la persistencia ortogonal es aplicada a *toda* la información del sistema.

Estos tres términos engloban la definición de la persistencia desde el punto de vista de los datos, en un programa de un lenguaje procedural y desde el punto de vista de los objetos en una metodología orientada a objetos. Nos referiremos a este tipo de persistencia como *Persistencia de Objetos*.

La idea del siguiente nivel de persistencia nace cuando entendemos como información persistente de la máquina la propia computación y no sólo los objetos y clases que utiliza. Por computación no queremos decir los métodos de los objetos o los mensajes que se puedan pasar distintos objetos entre sí, ya que esta forma de operar con los distintos datos de cada objeto se contempla en el primer tipo de persistencia cuando hacemos persistentes las clases. La computación de la máquina virtual es el *estado* en el que se encuentra en cada momento de ejecución.

El almacenamiento de los sucesivos estados de la máquina virtual, podría dar al sistema la capacidad de recuperar la ejecución de forma exacta [TUNES96]. Un sistema que cumpla este requisito es un sistema computacionalmente persistente y seguro, pero se enfrentaría a una falta de eficiencia por la actualización de sus distintos estados a un almacenamiento secundario. A este tipo de persistencia la denominaremos *Persistencia de Computación* (algunas bibliografías definen este término como *estabilidad* [MERLIN96]), y aunque no va a ser abordada en este estudio se darán las herramientas necesarias para un futuro desarrollo.

El último nivel de persistencia es el que se abre a la existencia de una sola máquina y que identifica la persistencia de un objeto no solo en el tiempo sino también en el espacio. Esto es lo que en alguna bibliografía se ha definido como *Persistencia en el Espacio* [BOOCH94]. Este concepto nace desde el punto de vista genérico de lo que se entiende como persistencia, pero en la máquina abstracta de Oviedo3 es más coherente abordar esta característica como la unión entre el sistema de persistencia y el de distribución. De esta forma se identifica la máquina abstracta como persistente y se posibilita la llamada al sistema de persistencia remoto de otra máquina, mediante los servicios de distribución desarrollados.

3. Persistencia de datos vs. persistencia de objetos

Todos los programas en cualquier lenguaje de programación actúan mediante un conjunto de instrucciones sobre una serie de datos. Si se necesita que unos datos persistan más allá de la duración de la ejecución de un programa, entonces se necesitará un almacén

permanente de datos. Hay varias razones para necesitar unos datos persistentes [RUMBA96]:

- Los datos persistentes pueden ser el mecanismo más sencillo para pasar datos de un programa a otro.

- Los datos persistentes permiten que el mismo programa reanude el procesamiento en un momento posterior.

Mediante la persistencia de datos, conseguimos que la información quede almacenada y pueda ser posteriormente recuperada. Para acceder a los datos guardados es necesario conocer el formato exacto de su almacenamiento, por lo que no se puede afirmar que dicha información sobreviva a la aplicación que lo creó.

Se puede plantear la persistencia de objetos como la persistencia de los datos, siendo éstos los atributos de cada objeto. Esto hace que surja el problema de ser necesario conocer la estructura del objeto para reconocer dichos datos, además de que no se recuperarían los métodos o los posibles modos de operar con los atributos de cada objeto (puesto que no fueron almacenados). Se puede afirmar pues que el simple hecho de hacer persistente los atributos de un objeto no hace ese objeto persistente puesto que no sobrevive a la aplicación que los creó (es necesario el programa en cuestión para recuperar el objeto de forma íntegra).

La persistencia abarca pues algo más que la mera duración de los datos. En la persistencia de objetos no sólo persiste el estado de un objeto, sino que su clase debe trascender también a cualquier programa individual, de forma que todos los programas interpreten de la misma manera el estado almacenado [BOOCH94]. El sistema de persistencia en Oviedo3, almacenará pues no solo los objetos (el estado de sus atributos) sino también sus clases (sus métodos) de forma que se consiga así que realmente el objeto sobreviva a la aplicación pudiendo ser recuperado por otro programa de forma íntegra.

4. Persistencia ideal

Uno de los puntos de vista innovadores de este artículo es el de dar persistencia a una máquina abstracta orientada a objetos frente a los sistemas de persistencia existentes en lenguajes de programación de alto nivel que utilizan herramientas como las bases de datos en almacenamiento [JAVASP96] y los RPCs en distribución [INFER96]. La máquina abstracta se basa en un lenguaje ensamblador orientado a objetos y no posee este tipo de herramientas. Deberá plantear el sistema de persistencia desde otro punto de vista: una persistencia que definimos como *ideal* [ORTIN97].

Entendemos por ideal, el hecho de que la propia máquina abstracta se encargue de almacenar en disco y recuperar a memoria las instancias persistentes de forma eficiente y totalmente transparente al usuario. No se tendrá que especificar de forma explícita cuando un objeto ha de ser almacenado y mucho menos dónde; sólo se declara persistente para que siga existiendo en el resto de las ejecuciones. El usuario considerará pues la máquina como un único recurso, sin jerarquía de memorias y podrá declarar tantas instancias persistentes como le permitan los recursos de su sistema.

Con la persistencia ideal se termina con las operaciones típicas de salvar, recuperar, actualizar o modificar información a disco, y muere el concepto de archivo: sólo existen objetos persistentes. Este concepto de persistencia es identificado por algunas fuentes bibliográficas como *orientado a los humanos* puesto que el usuario informático no necesita

tener conocimientos sobre la máquina física [MERLIN96].

Con lo comentado, se busca que el sistema de persistencia de Oviedo3 cumpla una serie de requisitos u objetivos que lo caracterizan como ideal:

- Soporte a instancias persistentes. Cualquier objeto podrá ser declarado como persistente.

- Gestión de clases persistentes. Los objetos persistentes han de ser instancias de clases persistentes para poder ser accesibles mediante otras aplicaciones (se pueda llamar a sus métodos).

- La declaración de una instancia como persistente es suficiente para asegurar el almacenamiento y actualización del objeto de forma automática. El usuario sólo deberá pues distinguir entre los objetos que desee volver a utilizar y los temporales.

- Para el usuario no existe memoria principal ni memoria secundaria. El sistema tiene una serie de recursos que han de ser utilizados de forma transparente.

- El sistema de persistencia trabajará de forma eficiente, utilizando siempre los recursos más rápidos cuando estén disponibles e intercambiándolos por otros lentos de forma eficiente.

- Muere el concepto de archivo: sólo hay objetos persistentes. Frente a los datos de un archivo, con un objeto persistente se obtienen además las operaciones a realizar con dichos datos (métodos).

5. Máquina abstracta sin persistencia

Bajo el concepto de persistencia ideal, se ha especificado el conjunto de requerimientos a exigir en el diseño del sistema de persistencia. Antes de entrar en análisis y diseño del sistema de persistencia, analizaremos la máquina abstracta para lograr una especificación coherente y lógica [IZQUIERDO96].

Carbayonia está compuesta fundamentalmente por cuatro áreas. Un área se podría considerar como una zona o bloque de memoria de un micro tradicional. Pero en Carbayonia no se trabaja nunca con direcciones físicas de memoria, si no que cada área se puede considerar a su vez como un objeto el cual se encarga de la gestión de sus datos, y al que se le envían mensajes para que los cree o los libere. A continuación se expone una breve descripción de las áreas que componen Carbayonia:

Área de Clases: En este área se guarda la descripción de cada clase. Esta descripción incluye los métodos que tiene, las variables miembro que la componen, de quien deriva, etc. Este área es fundamental para conseguir temas con la relación entre instancias y la persistencia de instancias.

Aquí ya puede observarse una primera diferencia con los micros tradicionales, y es que en Carbayonia realmente se guarda la descripción de los datos. Por ejemplo, en un programa en ensamblador del 80x86 hay una clara separación entre instrucciones y directivas de declaración de datos: las primeras serán ejecutadas por el micro mientras que las segundas no. En cambio Carbayonia ejecuta (por así decirlo) las declaraciones de las clases y va guardando esa descripción en éste área.

Área de Instancias: Aquí es dónde realmente se almacenan los objetos. Cuando se crea un objeto se deposita en éste área, y cuando éste se destruye se elimina de aquí. Se relaciona con el área de clases de manera que cada objeto puede acceder a la información de la clase a la que pertenece.

Las instancias son identificadas de forma única con un número que secuencialmente se irá incrementando a medida que se van creando nuevas instancias. La única posibilidad para acceder a una instancia es mediante una referencia que posee como identificador el mismo que la instancia. La única forma de acceder a una instancia mediante una referencia es accediendo a los métodos del área de instancias.

Referencias del Sistema: Son una serie de referencias que están de manera permanente en Carbayonia y que tienen funciones específicas dentro el sistema:

- this: apunta al objeto con el que se invocó el método en ejecución.
- exc: apunta al objeto que se lanza en una excepción.
- rr (return reference): referencia donde los métodos dejan el valor de retorno.

Área de Theads: En la ejecución de una aplicación se crea un thread principal con un método inicial en la ejecución. Mediante las herramientas de concurrencia se podrán ir creando más threads, coordinarlos y finalizarlos. Cada thread posee una pila de contextos y el conjunto de threads existentes en una ejecución se almacenan en este área.

La simulación realizada se basa en la creación de objetos dentro del simulador que identifiquen a las distintas partes de la arquitectura de la máquina virtual. Sus métodos son los servicios que estas partes ofrecen al resto de la máquina. Conforme se va interpretando el código fuente, se van modificando los valores de esta estructura.

6. Análisis funcional

La arquitectura anteriormente comentada deja entrever que lo que hace el simulador es imitar la funcionalidad de Carbayonia. Para cada clase definida creará un objeto clase en el área de clases, para cada instancia de estas clases utilizada en los programas, se creará una instancia en el área de instancias y, finalmente, se simularán cada proceso en ejecución con una instancia de la clase thread dentro del área de threads. Los distintos elementos se relacionarán mediante referencias y las tres áreas existentes se encargarán de dar los distintos servicios de cada uno de sus elementos.

La existencia de las tres áreas definidas permitirá (de la misma forma que hasta ahora) encapsular las distintas propiedades de sus elementos, tratándolos de forma homogénea. De la misma forma, se implementarán estas funciones para los objetos susceptibles de ser persistentes y obtendremos, gracias a esta encapsulación, un funcionamiento similar al no persistente.

Lo primero que haremos será descartar elementos que nunca serán persistentes. La única parte será el área de threads y por lo tanto cada thread y sus contextos. El hecho de que pudiesen ser susceptibles de convertirse en persistentes, significaría que se podría guardar el estado exacto de computación de la máquina virtual (habría que almacenar también las tres referencias del sistema). Este tipo de sistema de persistencia se ha definido como

Persistencia Computacional, de forma que es capaz de hacer persistente el estado de ejecución de la máquina virtual. Sería tolerante a fallos frente a caídas de tensión pero a precio de pagar una gran ineficiencia.

El área de clases y la de instancias han de poder albergar objetos persistentes y tratarlos además de igual forma que a los objetos temporales, así obtendremos los métodos públicos de estas clases invariables. El hecho de que sean persistentes las instancias y las clases, implica que lo sean también los métodos, las referencias y las instrucciones.

Como se observa no sólo es necesario almacenar las instancias para hacer que los objetos puedan ser persistentes; es necesario almacenar también sus clases [BOOCH94] ya que en éstas se encuentran los métodos que actúan sobre el objeto y las relaciones entre sus objetos (agregados, asociados y heredados). Operando de esta manera además conseguimos que la persistencia pueda ser utilizada por otros simuladores, en otros sistemas operativos y lo que es más importante, por otras aplicaciones Carbayonia.

Dentro de cada área se diferencia el conjunto de objetos que tienen carácter persistente de los que no. Se deben facilitar funciones de paso de un objeto persistente a temporal y en sentido contrario. Con estos objetos interactuará el sistema de persistencia de la forma más transparente al usuario. Para ello se deberá idear un sistema que trate a los objetos temporales y a los persistentes todos de la misma forma (aunque dentro de los persistentes haya de varios tipos: instancias, clases, referencias, métodos e instrucciones).

Por tanto el sistema de persistencia ha de basarse en la idea de construir un almacenamiento persistente, transparente al usuario y que trate a todos los objetos de igual manera. Y es importante resaltar el hecho de que no se deberá modificar la implementación de las clases de la máquina abstracta y mucho menos la forma de simular su funcionamiento.

En resumen, el hecho de que una instancia o clase Carbayonia sea persistente, se reduce a que sea persistente su objeto u objetos que lo representan en el simulador. Si conseguimos que sean persistentes estos objetos, podremos reconstruir el elemento persistente Carbayonia puesto que queda totalmente definido su estado por los objetos del simulador. Además debe ser lo suficientemente transparente como para no modificar la funcionalidad de las clases implicadas. El problema de la persistencia queda reducido pues, a resolver la persistencia en el lenguaje de programación utilizado en el simulador.

7. Diseño

Con la fijación de los objetivos a cumplir por el sistema de persistencia ideal requerido y el análisis funcional realizado sobre la especificación del simulador de la máquina abstracta orientada a objetos, presentamos el diseño realizado para el sistema de persistencia en Oviedo3.

7.1. Una memoria virtual

La idea principal en el desarrollo del sistema de persistencia de Oviedo3 se basa en un módulo cuya funcionalidad sea muy parecida a la de una memoria virtual, manejando toda la información relativa a los objetos persistentes [MERLIN96].

Se trata la memoria volátil como si fuese mucho mayor de lo que es, utilizando así la memoria secundaria siempre que sea necesario y sin que el usuario lo indique explícitamente. El sistema tendrá pues un conjunto de recursos pero el usuario sólo deberá de preocuparse de que el conjunto global de recursos se agote (no quede memoria

secundaria libre) y nunca del intercambio de información entre éstos.

7.2. Tipo de memoria virtual

Para que el intercambio de información entre el almacenamiento volátil y el no volátil de una memoria virtual se efectúe eficientemente, se agrupan los elementos en tamaños de información. Esto se realiza para que se trate de leer y escribir lo menos posible en la memoria secundaria, puesto que es mucho más lenta que la principal. Así tendremos parte de la memoria virtual en disco y parte en RAM.

Se suelen especificar agrupaciones de longitud fija, llamadas páginas o agrupaciones de tamaño variable denominadas segmentos. Otra variación de esto es dividir la información en segmentos de tamaño variable y agruparlos en páginas de tamaño fijo. A esta técnica se le conoce como segmentación más paginación. Las memorias virtuales basadas en la paginación más segmentación se suelen usar en los sistemas operativos.

La información necesaria para que un objeto pueda recuperar su estado, almacenada de forma secuencial, lo definimos como segmento dentro del diseño de la memoria virtual. Agrupando varios segmentos hasta que llegue a ser menor o igual que el tamaño de una página, obtenemos una página. De esta forma podremos implementar un sistema de segmentación más paginación para objetos persistentes.

Uno de los problemas a tener en cuenta es elegir el tamaño de la página y el tipo de segmento de tamaño máximo y mínimo. Esto es un tema abordado generalmente en bibliografías de sistemas operativos [DEITEL93]. Debemos tener en cuenta que:

- El tamaño de página ha de ser lo suficientemente grande como para que pueda albergar al menos, el segmento (objeto persistente) de mayor tamaño posible.

- Los segmentos no han de ser excesivamente pequeños, de forma que no pueda ocupar más la referencia al objeto (segmento) que el propio objeto.

- Se han de tener en cuenta todos los aspectos positivos y negativos de elegir páginas muy grandes o muy pequeñas y llegar a un compromiso [DEITEL93].

Por lo indicado anteriormente y teniendo en cuenta el carácter de análisis e investigación del proyecto Oviedo3, el sistema de persistencia trabaja con un tamaño de página configurable. Cada sistema podrá utilizar un tamaño propio y será interpretado correctamente puesto que es el primer elemento del archivo de intercambio. De forma paralela, se podrá especificar el tamaño de la página que se toma por defecto.

7.3. Diagrama de clases

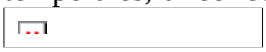
Identificamos una clase susceptible de ser persistente dentro del simulador de la máquina abstracta orientada a objetos. El diagrama de clases que establece la conexión con estas clases y el sistema de persistencia es el mostrado en la Figura 4. La metodología utilizada es OMT [RUMBA96]

7.3.1. Identificación de objetos persistentes

El área de instancias trabaja con numeraciones de las instancias existentes en la ejecución de un programa. Conforme se van creando instancias, esta área va asignándoles identificadores naturales secuenciales de un tamaño de 4 bytes. La forma de identificar a las

instancias persistentes y en general a todo elemento persistente (método, instrucción, referencia, clase o instancia), será con el mismo identificador pero con otra semántica. Este identificador es el atributo *ID* de la clase *TVMItemTemplate* que se asocia al objeto persistente mediante un template C++.

En una memoria virtual se ha de trabajar con objetos virtuales.Éstos deberán estar en la memoria principal cuando se tengan que utilizar pero podrán pasar a la secundaria cuando no sean requeridos. De esta forma es necesario trabajar con lo que se conoce como punteros a objetos virtuales. Estos punteros han de ser capaces de identificar dónde se encuentra el objeto dentro de la RAM o en el disco, cuando estén allí ubicados. La función de puntero virtual es ofrecida por las instancias de *TVMItemTemplate* y su atributo *ID* tiene la información adecuada para localizarlo.

Estudiaremos pues las características de los identificadores de instancias. Para los objetos temporales, al ser secuencial la generación de sus identificadores, podremos utilizar hasta  instancias. La utilización de estos cuatro bytes para los objetos persistentes hace coherente la manipulación de las instancias pero su función se amplía a:

- Conocer la página en la que se encuentra.
- Conocer el segmento dentro de la página.
- Localizar el objeto temporal en memoria asociado, en el caso de que exista.

El bit más significativo nos da su condición de persistencia. En función del tamaño de página preestablecido (2 elevado a n bytes) tenemos la identificación de la página a la que pertenece (los 32-1-n bits más significativos después del primero) y el desplazamiento del segmento (los 32-n bits menos significativos). Un ejemplo con una página de 64Kbytes se muestra en la Figura 5. Sea como fuere, cualquier tamaño de página que queramos elegir, nos permitirá un almacenamiento máximo de 2.147.483.648 bytes o lo que es lo mismo de 2 Gbytes. Todo esto será para el usuario el tamaño del "recurso memoria" que dispone para instancias persistentes.

Una vez conocido el segmento, accedemos a él. Su cabecera tiene información relativa al segmento y por lo tanto al objeto almacenado. Se identifica el tipo de segmento (tipo de objeto), su longitud (puesto que es variable) y la dirección del objeto temporal asociado. Con este último campo, la memoria virtual conoce la situación del objeto temporal a partir de su *ID*.

7.3.2. Funciones de TVirtualMemory

Las distintas funciones que ha de llevar a cabo, así como las implicaciones que se deducen del diagrama de clases mostrado son las siguientes:

- Poder acceder a cada segmento dentro de cada página en memoria de forma automática. Se accede mediante el ID de cada objeto persistente.
- Implementar políticas de emplazamiento y reemplazamiento genéricas, pudiendo utilizar cualquiera. Para ello se utiliza la clase abstracta Tpolicy y en concreto sus derivados.
- Llevar una estadística fiable de la ubicación de las páginas en cada memoria para

poder acelerar en lo posible los accesos y facilitar la implementación de las distintas políticas. Para esto la memoria virtual esta asociada a una lista de objetos instanciados de TStatistics. Cada uno de ellos guarda estadísticas de utilización de todas las páginas en disco y memoria.

- Utilizar de forma eficiente la memoria principal y realizar el menor número de trasferencias entre las distintas capas de la jerarquía de memoria. Para ello se ha creado una tabla hash de páginas en memoria identificada por la clase *TDictionaryAsHashTable*.

- Poder utilizar cualquier tamaño de página para poder realizar pruebas antes de llegar a un compromiso. Se configura mediante el atributo *BytesBlock* de la clase *TvirtualMemory.*

- Se deberá encargar de establecer y mantener un formato de archivo de intercambio independiente del entorno utilizado. Así, el sistema de persistencia de una máquina abstracta podrá ser portado a otra máquina que se ejecute en otro entorno.

- Facilitará un puntero real a partir de un puntero a objeto virtual. Para esto, se sobrecarga el operador -> en TVMItemTemplate.

7.3.3. Emparejamiento objeto temporal - objeto persistente

El desarrollo del sistema de persistencia ha de ser lo suficientemente genérico como para poder ser aplicado a cualquier clase sin que se modifique la funcionalidad de sus objetos. Para cumplir este objetivo, se establece una relación entre una pareja de objetos temporal y persistente denominada protocolo [CORBA97].

Cada vez que se declare un objeto persistente, se crea uno temporal. El usuario trabajará con el objeto temporal y el sistema de persistencia se encargará de efectuar una actualización entre los dos objetos mediante un protocolo, de esta forma, el usuario no será consciente de que el objeto es persistente. El objeto temporal es pues la instancia de *TVMItemTemplate* y el objeto persistente es el segmento de la memoria virtual asociado. Hay que decir que los segmentos que hay dentro de cada página no son los objetos persistentes propiamente dicho sino que es la información necesaria para que el objeto pueda reconstruir su estado de forma completa.

La memoria tiene dos métodos por los cuales se crean y destruyen objetos temporales asociados a persistentes. Cuando por necesidades de memoria y la política de reemplazo lo crea conveniente, una página sea volcada a disco, los objetos temporales asociados a los segmentos de la página deberán ser destruidos (para liberar la memoria asociada). De la misma forma cuando se solicite un objeto persistente y no exista, se cargará su página y se creará el objeto temporal con su estado persistente almacenado.

Como observamos, es necesaria una actualización continua entre la pareja de objetos identificadas. El momento en el que se debe transferir la información lo indica la memoria virtual, pero no el modo de hacerlo. De esta forma, toda clase susceptible de ser persistente, ha de implementar los métodos de identificación (*Type*), tamaño (*Size*), actualización temporal - persistente (*Write*) y actualización persistente - temporal (*Read*) [ORTIN97].

7.4. Bloqueo de páginas

El hecho de implementar una memoria virtual para poder utilizar grandes cantidades de

datos es sólo una visión parcial de lo que es un sistema de persistencia. Hay que tener en cuenta que lo que deseamos hacer persistente son objetos y no meros datos. Los objetos poseen métodos y por lo tanto computación y esto hace que el problema se haga bastante más complejo [BOOCH94].

Para solucionar uno de los problemas anteriores, se diseña un protocolo entre objetos y una representación uniforme de objetos persistentes como veíamos en el apartado anterior pero surge un problema adicional. El hecho de que queramos hacer que un objeto sea persistente y por lo tanto que se libere su objeto temporal cuando sea necesario, dará problemas si se libera en la ejecución de uno de sus métodos. Esto puede ocurrir si en el método se usa un objeto persistente y éste hace que la memoria virtual nos destruya a nosotros.

Cuando el objeto temporal asociado a un objeto persistente ejecuta uno de sus métodos ha de bloquear la página a la que pertenece su objeto persistente porque si ésta es liberada, el objeto temporal asociado también será destruido con el consiguiente error en ejecución.

Los mecanismos de bloqueos por página son similares al funcionamiento de un semáforo. Cada vez que se realiza un bloqueo se incrementa en una unidad el contador de bloqueo y cada vez que se desbloquea se decrementa. Una página podrá ser liberada cuando esté en memoria y además su contador de bloqueo sea exactamente igual a cero.

La clase *TVirtualMemory* posee dos métodos (*Wait y Signal*) que, identificando un objeto persistente, nos bloquea su página asociada. El contador de bloqueos es un campo de los objetos *TStatistics* de cada página puesto que es otro criterio para llevar a cabo los reemplazamientos.

También es necesario, por motivos de eficiencia, bloquear en un momento de ejecución todas las páginas de memoria para que no haya actualizaciones a disco. Esto se lleva a cabo con la llamada a los métodos ya identificados, sin darle el identificador de ningún objeto.

8. Interfaz con el usuario

Una vez especificado el funcionamiento del sistema de persistencia en Oviedo3, describiremos cómo el usuario puede utilizarlo mediante el lenguaje Carbayonia. La interfaz de usuario varía en muy poco del sistema base de la máquina abstracta, significando tan sólo puntuales añadiduras, conservando el modo de funcionamiento ya establecido [IZQUIERDO96].

Se ha seguido con la sintaxis del lenguaje orientado a objetos de la máquina abstracta. No se identifican apenas instrucciones, son consecuciones de llamadas a métodos. Se identifican unas clases primitivas y el usuario crea las suyas en función de éstas. De esta forma, se identifica una nueva clase *Persistence* cuyas instancias nos permiten llamar al sistema de persistencia. Para facilitar la labor al usuario, se ha creado una instancia siempre presente en tiempo de ejecución identificada por la nueva referencia del sistema *Persistence.*

La declaración de clases persistentes se hace de igual modo que una clase normal pero anteponiendo la palabra *Persistent* a la de *Class* [POET97]. El hecho de que una clase sea persistente no quiere decir que sus agregados lo sean y por lo tanto la condición de persistencia de cada agregado ha de ser declarada explícitamente de la misma manera que la clase.

Los objetos ancestros de un objeto persistente son también persistentes y la condición de

persistencia de los agregados de los primeros viene dada por la definición de la clase. Un ejemplo puede ser: el buffer es un objeto agregado que es temporal y que se irá actualizando paulatinamente al objeto persistente File. El objeto ancestro de tipo Texto que se crea será también persistente y su persistencia será definida por su clase.

Las instancias en el sistema de persistencia, se identifican de forma única con una cadena de caracteres (con un objeto *String*). La funcionalidad de este objeto viene dada por los siguientes métodos:

- *Exists (string): bool*. Nos devuelve un objeto booleano que indica la existencia de un objeto persistente con el identificador igual que el parámetro pasado.

- *Add (string, object)*. Da el carácter de persistencia a un objeto temporal asignándole además un nombre para posteriores identificaciones.

- *GetObject (string): object*. Dado el nombre del objeto persistente, obtenemos una referencia al objeto. Toda operación realizada sobre este objeto persistirá en el tiempo.

- *Delete (string)*. Elimina del directorio de persistencia el objeto cuyo identificador coincida con el del primer parámetro.

9. Conclusiones

El sistema de persistencia de Oviedo3 se puede definir como *ideal.* No se entiende pues como una serie de librerias que dan la facilidad de almacenar una serie de datos, sino como la capacidad de poder utilizar objetos persistentes, sin preocuparse de dónde, cuándo o cómo han sido almacenados. Desaparece por tanto el concepto de archivo.

La idea principal en el desarrollo de este sistema de persistencia se basa en un módulo, cuya funcionalidad es muy parecida a la de una memoria virtual, que gestione toda la información relativa a los objetos persistentes. Para la interfaz con el usuario se identifica una nueva clase, *Persistence,* cuyas instancias permiten invocar al sistema de persistencia, y una nueva referencia del sistema.

El disponer de una máquina abstracta con persistencia va a ser especialmente beneficioso para el SGBDOO. El motor para el SGBDOO de Oviedo3 aprovechará la persistencia proporcionada por la máquina abstracta y basándose en ella realizará, por ejemplo, la gestión de índices.

10. Referencias

[ALVAREZ96] **Persistencia en sistemas operativos orientados a objetos bases de datos.** Álvarez D., et al. Actas de las Primeras Jorn Datos. La Coruña, 1996.

[ALVAREZ97] **An object-oriented abstract machine as the substrate for a** D., et al. 11th European Conference on Object-Oriented Progra

[BOOCH94] **Análisis y diseño orientado a objetos con aplicaciones.** Gra Santos, 1994.

[CORBA97] **CORBA and OMG Information Resources.** The Object M Broker Architecture, 1997.

http://www.omg.org

[CUE VA96] **Acercando las tecnologías orientadas a objetos al hard** Primeras Jornadas de Trabajo de Ingeniería del Software. Ed. M

[DEITEL93] **Introducción a los sistemas operativos.** Harvey M. Deitel. 1993.

[INFERN96] **Inferno – Lucent Technologies**. 1996.http://inferno.lucent.com/in

[IZQUIERDO96] **Máquina abstracta orientada a objetos**. Raúl Izquierdo Oviedo. Escuela Técnica Superior de I.I. e Ingenieros Informát

[JAVASP96] **JavaSpaces™ .** Sun Microsystems Computer Corporation. Jav

[MERLIN96] **The Merlin Project.** 1996. http://www.lsi.usp.br/~jecel/merlin.html

[ORTIN97] **Diseño y construcción del sistema de persistencia en Ovie** Universidad de Oviedo. Escuela Técnica Superior de I.I. e Inge

[POET97] **Poet Software.** Poet 4.0 C++ Programmer's Guide. http://www.

[RUMBA96] **Modelado y diseño orientado a objetos. (Metodología OM**

[TUNES96] **The Tunes Proyect.** 1996. http://www.eleves.ens.fr:8080/home/ridea