Concurrencia en un sistema operativo orientado a objetos basado en una máquina abstracta reflectiva orientada a objetos

L. Tajes Martínez, F. Álvarez García, M.A. Díaz Fondón, J.M. Cueva Lovelle, D. Álvarez Gutiérrez, R. Izquierdo Castanedo

Departamento de Informática, Universidad de Oviedo

c/ Calvo Sotelo, s/n, 33007 OVIEDO - ESPAÑA

{ tajes, fag, fondon, cueva, darioa, ric}@pinon.ccu.uniovi.es

Resumen

Este trabajo describe los principales aspectos del sistema operativo para el sistema Oviedo3 dedicándose una atención muy especial al soporte para la concurrencia. Oviedo3 se ha concebido como un sistema operativo basado totalmente en el paradigma de orientación a objetos y para dar soporte al mismo. Este sistema operativo se construye sobre una máquina abstracta orientada a objetos, independiente de la plataforma subyacente, que proporciona el modelo básico de objetos que se utiliza en todo el sistema. Los cuatro elementos básicos que caracterizan el sistema operativo son la seguridad, la persistencia, la distribución y la concurrencia. La concurrencia permite explotar el paralelismo tanto a nivel interno como entre objetos diferentes. Se basa en el concepto de objeto autocontenido soportado por la máquina. Los objetos encapsulan sus datos y su computación estando dotados, conceptualmente, de un planificador interno que gestiona las ejecuciones concurrentes y controla el acceso correcto a los datos del mismo.

Palabras Clave: Concurrencia, Sistemas operativos Orientados a Objetos, Máquina abstracta, Reflectividad

1 INTRODUCCIÓN

En los últimos años se han llevado a cabo varios intentos de introducir concurrencia en el sistema con el objetivo de mejorar el rendimiento por encima de los límites de la máquina subyacente. Estos intentos se han llevado a cabo de distintas formas con distintos enfoques y han tenido diferente éxito.

La mayor parte de los enfoques que se han dado para introducir concurrencia en un sistema han sido distribuyendo el modelo de concurrencia elegido entre el sistema operativo y el lenguaje tratando de implementar construcciones para la concurrencia y la sincronización.

En los lenguajes tradicionales sobre sistemas operativos tradicionales, los diseñadores dotaban al sistema operativo con los mecanismos básicos para introducir la concurrencia como podían ser: clonación y destrucción de procesos dotándolos de vida independiente, mecanismos de planificación de los distintos procesos existentes, etc.

El sistema operativo ofrecía así la abstracción de actividad o proceso que era capaz de ejecutar parte del código de un programa. Además, era necesario ofrecer un mecanismo que permitiese la sincronización de los mismos en puntos determinados. Así, el sistema operativo era capaz de parar procesos y ofrecía abstracciones de sincronización de muy bajo nivel como

los semáforos. Además de gestionar *personalmente* el acceso correcto a sus estructuras de datos internas.

Sobre estos sistemas operativos, los lenguajes no concurrentes, es decir, aquellos que no disponían de un soporte en tiempo de ejecución que soportase construcciones de concurrencia, podían utilizar los mecanismos ofrecidos por el sistema operativo.

Por otro lado, también es posible encontrar lenguajes, incluso construir uno propio, que apoyándose en las funciones ofrecidas por los sistemas operativos puedan ofrecer un modelo de concurrencia más sofisticado. Por ejemplo, es posible que el sistema operativo ofrezca una abstracción de semáforo y el lenguaje, basándose en los semáforos gestionados por el sistema operativo construya monitores. Esta extensión nacía con la principal finalidad de evitar los peligros derivados de dejar en manos del programador herramientas de muy bajo nivel.

Se han desarrollado multitud de combinaciones al introducir modelos de concurrencia, iguales o diferentes, a nivel de sistema operativo y lenguaje.

En cualquier caso, por debajo de todo esto, el soporte lo daba una máquina hardware que ofrecía un modelo de computación muy diferente al ofrecido por lenguaje + sistema operativo. Mientras estos ofrecen una visión de procesos en ejecución sincronizándose con algún mecanismo de más o menos alto nivel, la máquina ofrece una visión basada en regiones de memoria, registros y pila y un conjunto de instrucciones de muy bajo nivel para cargar, almacenar y/o transferir datos entre memoria y registros y ejecutar la siguiente instrucción señalada por un determinado registro. Además, con el fin de facilitar la ejecución de actividades concurrentes, la máquina hardware subyacente puede ofrecer instrucciones que permitan habilitar o deshabilitar interrupciones para evitar cualquier interferencia en la ejecución de determinadas operaciones críticas.

2 APARICIÓN DEL PARADIGMA OO

Recientemente, aparecen y se popularizan los lenguajes de programación OO como C++, Eiffel, etc. que introducen un paradigma diferente basado en objetos. Estos objetos, encapsulan datos y métodos que constituyen el único punto de entrada al objeto y el único mecanismo válido para acceder y/o modificar los datos.

Sin embargo, la incorporación del paradigma de OO no se está llevando a cabo de una manera integral en todos los componentes de un sistema. Existen lenguajes, bases de datos, etc., que utilizan el paradigma de la orientación a objetos que deben realizar un cambio de paradigma para interactuar con otros elementos del sistema como el sistema operativo. Esto provoca un grave problema de desadaptación de impedancias y de interoperabilidad, al tener que cambiar de paradigmas y/o adaptando interfaces en función del elemento del sistema con que se esté tratando. Todo ello redunda en una proliferación de capas de software adicional, que intentan paliar estos problemas, produciendo una complejidad adicional en el sistema.

En cuanto a la concurrencia, suele lograrse de manera similar a la anterior, planteando los mismos problemas. Así, los sistemas operativos ofrecen mecanismos básicos de creación de procesos y, por encima, los lenguajes ofrecen una plataforma en tiempo de ejecución para soporte de la concurrencia y sincronización.

Sin embargo, dependiendo del lenguaje los procesos pueden ser entidades independientes de los objetos o pueden estar relacionados con ellos. Así, podemos encontrarnos con **objetos activos** si la entidad objeto está relacionada con la entidad que soporte la ejecución (actividad, job, hilo, etc.), o bien **objetos pasivos**, si los objetos son meros contenedores de estado y

métodos y existe una entidad o jerarquía de objetos especial llamados objetos hilo, actividad, job, etc. que se encargan de llevar a cabo la ejecución.

Así, por ejemplo, en el lenguaje GUIDE, los hilos, que reciben el nombre de actividades, se comunican llamando a operaciones en objetos pasivos compartidos. El mecanismo de sincronización consiste en asociar condiciones de activación con las operaciones de los objetos. Estas condiciones de activación deben ser ciertas antes de que se permita la ejecución. Sin embargo, en ABCL/1, los objetos son entidades activas que encapsulan un hilo de control y que se comunican a través del paso de mensajes.

Por debajo de esto se sigue trabajando con máquinas tradicionales que ofrecen poca ayuda en la gestión y control de la concurrencia.

3 SISTEMA INTEGRAL OO

Para solucionar los problemas de interoperabilidad derivados de soportar entornos OO con sistemas operativos tradicionales, se propone desarrollar un entorno integral OO que recibe el nombre de Oviedo3 donde, desde las interfaces de usuario hasta las aplicaciones, pasando por lenguajes, compiladores, etc. utilizan los conceptos de orientación a objetos. (Figura 1).

El sistema operativo es la base de este sistema integral y ofrecerá una única abstracción que es el objeto. Este puede crear nuevos objetos o enviar mensajes a otros objetos. Entre los aspectos fundamentales que debe incorporar un sistema operativo para dar soporte al sistema integral anterior están los mecanismos de protección, persistencia, distribución y concurrencia de manera totalmente transparente para los objetos del sistema.

Una técnica para estructurar un sistema operativo orientado a objetos como soporte de un sistema integral que aporta varias ventajas es utilizar como base una máquina abstracta orientada a objetos que proporcionará el modelo básico y soportará los objetos del sistema. La funcionalidad del sistema operativo se proporciona mediante un conjunto de objetos de igual categoría que cualquier otro objeto que pueda crear el usuario.

En los siguientes apartados se discuten las características de una máquina abstracta sobre la que basar el sistema operativo, qué mecanismos puede utilizar para relacionarse con el sistema operativo (como reflectividad), así como las ventajas que proporciona. A continuación, se describen las líneas maestras del diseño del sistema operativo que da soporte al sistema integral. Por último se detalla cómo el uso de esta máquina facilita la consecución de una de las características fundamentales del sistema operativo como es la concurrencia.

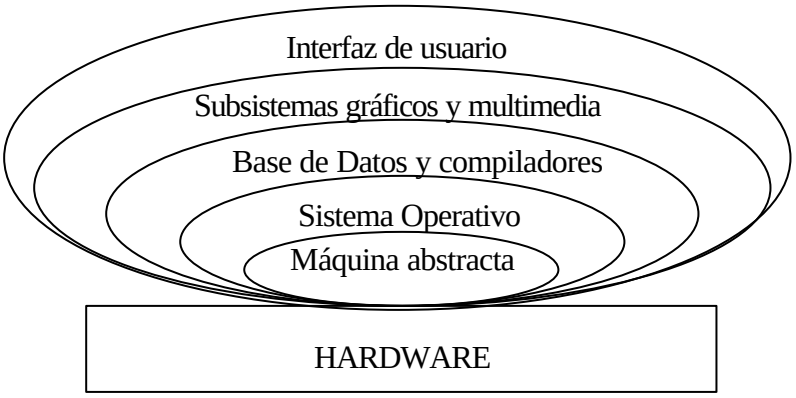

Figura 1. Componentes del sistema Oviedo3.

4 La máquina abstracta

La aproximación del proyecto Oviedo3 para construir un sistema integral OO es utilizar una máquina abstracta reflectiva que proporcione el soporte básico de objetos para el resto del sistema, y un SOOO en la forma de un conjunto de objetos que proporcionan funcionalidades para los objetos de usuario. (Véase figura 2)

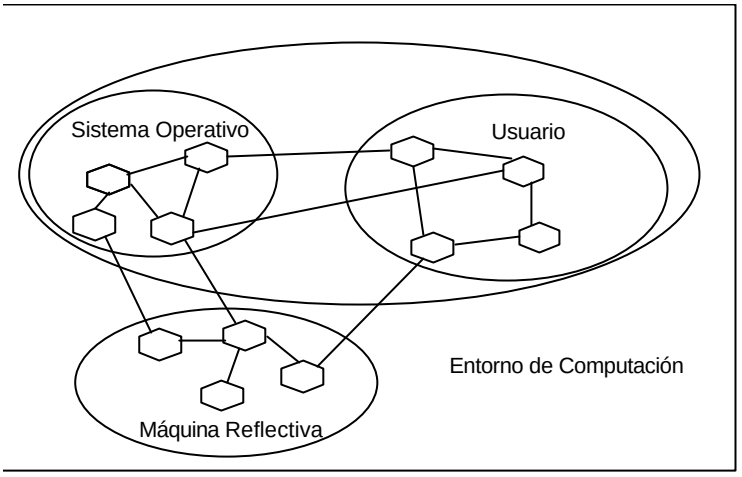

Figura 2. Entorno de computación compuesto de un conjunto de objetos homogéneos

La máquina abstracta debe proporcionar la funcionalidad y la arquitectura básica para dar soporte al sistema operativo. Entre las obligaciones de la máquina abstracta destacan:

- Proporciona el modelo de objetos para todo el sistema. Este modelo debe proporcionar objetos simples, homogéneos y autocontenidos.
- Da soporte a los demás objetos tanto de usuario como los que implementan funcionalidad del sistema operativo.
- Permite gran portabilidad de los objetos de usuario o del sistema operativo, que trabajarán sin modificaciones en cualquier plataforma donde esté la máquina abstracta.
- Facilita la implementación de características del sistema operativo.

4.1 Reflectividad

Para lograr la máxima homogeneidad y poder aplicar las ventajas de la OO incluso a la propia máquina de soporte, es necesario dotar a la misma de una arquitectura reflectiva. Así, la propia máquina se describe como un conjunto de objetos que pueden ser utilizados como cualquier otro objeto del sistema. La flexibilidad y adaptabilidad del entorno se maximizan.

4.2 Arquitectura de la máquina abstracta

A continuación se describe la estructura de la máquina abstracta Carbayonia, que se ha diseñado en los inicios del proyecto específicamente para dar soporte al sistema operativo de la forma descrita anteriormente. Esta estructura puede considerarse como una referencia de las características que debería tener una máquina abstracta para dar soporte a este sistema operativo orientado a objetos. También podrían adaptarse para este fin otras máquinas abstractas.

La máquina da soporte un modelo de objetos cuyas características fundamentales son las siguientes:

- Abstracción e identidad de los objetos
- Encapsulación
- Herencia y, además, relaciones genéricas y de agregación entre objetos.
- Polimorfismo o paso de mensajes

En la figura 3 se muestra la arquitectura de la máquina abstracta Carbayonia [1], que está compuesta por cuatro áreas: Área de clases, de referencias, de instancias y referencias del sistema.

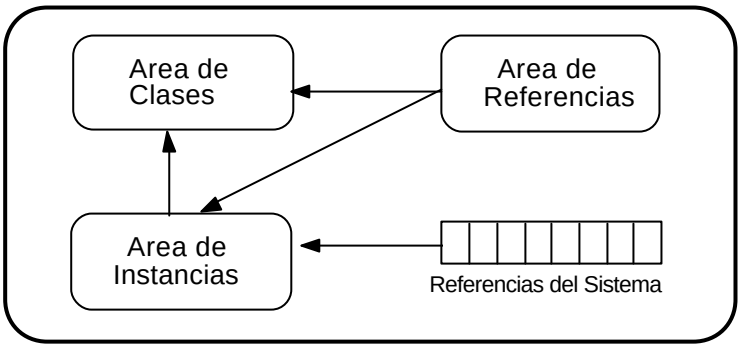

Figura 3. Arquitectura de la máquina Carbayonia.

Cada área se puede considerar como un objeto que se encarga de la gestión de sus datos.

Su principal característica es que toda acción sobre un objeto se realiza a través de una referencia al mismo.

- **Área de clases**. Guarda la descripción de cada clase.
- **Área de referencias**. Almacena referencias a objetos. Estas tienen tipo (relación con el área de clases) y apuntan a objetos de ese tipo (relación con el área de instancias).
- **Área de instancias**. Almacena objetos creados que se eliminan cuando se destruyen. Cada objeto tiene acceso a información de su clase(relación con el área de clases).
- **Área de referencias del sistema**. Que tienen funciones específicas y existen de manera permanente en el sistema.

5 SO4

El núcleo central del sistema Oviedo3 es el sistema operativo, denominado SO4. Ofrece la abstracción de un espacio de objetos único en el que existen los objetos indefinidamente, virtualmente infinito y en el que cooperan transparentemente los objetos situados en diferentes máquinas a través del intercambio de mensajes. Además, el SO se encarga de lograr de manera transparente una serie de características importantes, entre las que destacan como la seguridad, la persistencia, la distribución y la concurrencia. La seguridad garantiza que sólo pueden enviar mensajes a otros objetos aquellos que estén autorizados. La persistencia permite que los objetos permanezcan en el sistema de manera indefinida hasta que ya no son necesitados. La distribución hace indiferente la máquina en la que está el objeto al que se

envía el mensaje. La concurrencia proporciona al modelo de objetos la capacidad de ejecutar tareas en paralelo.

SO4 se estructura sobre la máquina abstracta como un conjunto de objetos que proporcionan de manera transparente determinadas funcionalidades al resto de los objetos del sistema. Hay varios sistemas operativos OO, como Choices, Apertos, Spring y otros. Cada uno de ellos se enfoca con una filosofía OO diferente, con diferentes estructuras. Choices [2] está basado en un framework OO C++ que puede especializarse para personalizar el sistema operativo. Apertos [3] usa una separación objeto/metaobjeto separación para estructurar el sistema operativo. Spring [4] usa un lenguaje de definición de interfaces para definir las interfaces del sistema como objetos y un microkernel con las abstracciones de distribución embebidas. Inferno es un sistema operativo basado también en una máquina virtual acoplada con un kernel, diseñada para el desarrollo de aplicaciones de red.

Las líneas maestras que guían el diseño del sistema operativo son:

- **Objeto como única abstracción**. La única entidad existente en el sistema es el objeto, sea cual sea su granularidad.
- **Modelo de objetos intencionadamente estándar**, con las características más comunes de lenguajes de programación más extendidos: encapsulación, herencia y polimorfismo [5].
- **Modo de trabajo exclusivamente OO.** Los objetos pueden crear clases que hereden de otras, crear objetos de una clase y enviar mensajes a otros objetos.
- **Simplicidad**: Fidelidad a las líneas anteriores, buscando la máxima sencillez.

Algunas características importantes que se desean los objetos en consonancia con las líneas anteriores son:

- **Objetos homogéneos**. Todos los objetos tienen la misma consideración, no hay objetos especiales que reciban un trato distinto. Los objetos propios del SO no son diferentes del resto de objetos de usuario.
- **Transparencia**. Los objetos deben ser ajenos a la existencia de mecanismos del SO que permiten conseguir características como persistencia, envío de mensajes, distribución, etc.
- **Objetos autocontenidos**. Los objetos deben tener un alto grado de autonomía, y su comportamiento no debe depender de otros objetos. Toda la información acerca del objeto está encapsulada en el mismo. Esto facilita la consecución con transparencia de objetivos como la persistencia, la distribución, etc. y decanta el modelo de objetos a un modelo de objetos activo, en el que el objeto encapsula su computación.
- **Semántica completa**. Los objetos tienen toda la semántica que da el modelo de objetos, no son considerados como una mera zona de memoria contigua.
- **Identidad de los objetos**. Cada objeto tiene un identificador único, que se usará como referencia para acceder al mismo.

6 Concurrencia

En este sistema, donde las acciones son llevadas a cabo por objetos que pueden comportarse como servidores ante la petición de otros que asumen el papel de clientes, se intenta alcanzar la máxima concurrencia [8] garantizando siempre la corrección de las operaciones que se efectúen y con ello la consistencia en el estado del objeto.

El sistema operativo debe establecer un control de la misma entre objetos, entre distintos métodos de un objeto y entre distintas instancias del mismo método de un objeto.

En el primer nivel, se debe permitir la coexistencia del máximo número de servidores y clientes posible según las necesidades de trabajo del momento.

No se establecerán diferencias en los dos niveles posteriores y se tratarán de la misma forma las peticiones de activación de métodos dentro de un objeto, ya sean el mismo o distintos métodos.

Se trata de dotar al sistema con un modelo de concurrencia [6] que consiga los siguientes objetivos:

- **Maximizar el grado de paralelismo de forma segura.** Se debe permitir la concurrencia entre objetos y dentro de un objeto evitando ejecutar simultáneamente métodos que puedan dar lugar a inconsistencias en el estado de una instancia.
- **Simplicidad**. El modelo de concurrencia debe ser tan simple como sea posible para facilitar su uso.

Los principales aspectos del modelo de concurrencia propuesto para el sistema son:

- **Objetos Activos Multihilo**. [7, 8]. Debido al modelo de objetos autocontenidos proporcionado por la máquina abstracta, los objetos deben ser activos porque incluyen la computación. Conceptualmente, cada objeto dispone de un multiprocesador virtual para múltiples ejecuciones, además de los necesarios mecanismos de control para la correcta ejecución y la preservación de la integridad. Otros sistemas operativos existentes, como Clouds [9] y Guide [10] prefieren un modelo de objetos pasivo y suministran una abstracción como proceso o hilo para proporcionar computación. Una tercera posibilidad es dotar a los objetos con un número limitado de hilos. Sin embargo, la primera propuesta define un modelo más flexible ya que dentro de cada objeto la activación de métodos es totalmente dinámica y se lleva a cabo ante la invocación de un trabajo al objeto servidor por parte de un cliente (figura 4).

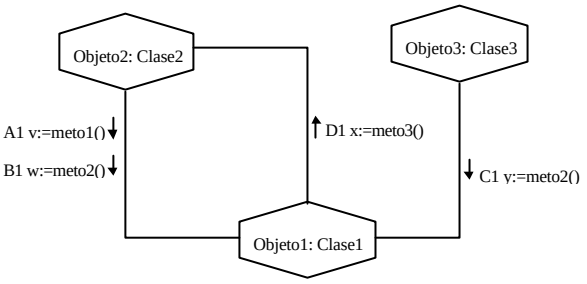

Figura 4. Llamadas concurrentes a un objeto

- **Invocación Síncrona** [8]. Dada la composición de la máquina y la jerarquía básica de clases que ofrece, es evidente que la ejecución de un método en un objeto dará lugar a múltiples invocaciones a otros métodos en otros objetos. Estas invocaciones pueden ser síncronas o asíncronas. Se ha optado por una invocación síncrona, siendo el criterio de la simplicidad el que más ha pesado a la hora de decidir. Con este tipo de invocación, la ejecución de un método en un objeto, se bloquea durante la invocación de un método en otro objeto.
- **Métodos exclusivos y concurrentes**. Todos los métodos definidos para un objeto deben estar catalogados como exclusivos o concurrentes. Un método exclusivo modifica el estado del objeto y no puede coexistir con ningún otro hilo en el objeto. Los objetos se comportan como monitores cuando tratan con este tipo de métodos. Los

métodos concurrentes no modifican su estado, por tanto, su invocación es compatible con otras de forma concurrente. Esta aproximación se ha hecho muy popular y, recientemente, lenguajes como Java tienen un modelo de concurrencia similar. [11].

Este modelo intenta conseguir un balance entre la maximización del paralelismo y la simplicidad conceptual. La invocación síncrona junto con métodos exclusivos y concurrentes es más sencilla de entender para el programador y el usuario del objeto. En cualquier caso, se mantiene un alto grado de paralelismo.

Un conjunto de objetos proporciona la funcionalidad del mecanismo de concurrencia. Por tanto, se incluirá parte de esta funcionalidad en la máquina y el resto en objetos que proporcionan la funcionalidad del sistema operativo.

6.1 Definición de métodos

La máquina debe ofrecer la posibilidad de caracterizar los métodos como exclusivos y concurrentes permitiendo de esta forma definir puntos de sincronización en las ejecuciones.

- Los exclusivos modifican el estado del objeto y deben ser ejecutados sin interferencias
- Los concurrentes pueden convivir sin problemas con otros métodos activos en el objeto ya que únicamente acceden al estado del objeto sin modificarlo.

6.2 Activación de métodos en un objeto

Las exigencias que se le han hecho a la máquina se fundamentan en que es la única que posee la habilidad de ejecutar métodos. Por tanto, entre las instrucciones de la máquina abstracta debe ofrecerse una instrucción `invocamétodo` que lleve a cabo el paso de mensaje desde el objeto origen al objeto destino con la petición de que ejecute un determinado método de los ofrecidos en el punto de entrada.

Por otro lado, es necesario determinar quién debe ofrecer el soporte a los mecanismos de planificación de las ejecuciones. Dado que cada objeto es autocontenido y encapsula su computación, será cada uno de los objetos el que implemente un planificador interno de las ejecuciones que se están llevando en sus métodos.

La activación de métodos dentro de un objeto vendrá así claramente condicionada por el tipo de método al que nos referimos.

- Si el método es exclusivo, la activación de un método se demorará hasta que no haya ningún método activo dentro del objeto. Es decir, funcionan como monitores.
- En el caso de un método concurrente, la restricción se refiere únicamente a la existencia de hilos activos ejecutando métodos exclusivos.

Cada objeto debe registrar qué métodos están pendientes de finalizar y reúnen las condiciones necesarias para ello.

Así se dotará al objeto de un mecanismo similar al propuesto como extensión al Eiffel [12]. Todo objeto encapsulará su propio objeto planificador y de esta forma, todo mensaje enviado al objeto como petición de ejecución de un método (o invocación de método ya que estos son conceptos sinónimos) llegará al planificador del objeto.

Se puede ver un objeto compuesto de dos partes diferenciadas, los datos y los metadatos. Los datos se corresponden con los valores de sus variables de estado mientras los metadatos se corresponden con los valores del entorno en que se ejecuta el objeto. Parte de estos

metadatos serían los correspondientes al entorno de ejecución de los métodos del objeto. Por ejemplo, el estado del objeto, si está ocioso o si está ocupado, el estado de las ejecuciones de este objeto, como pueden ser los métodos invocados en este objeto, el estado de cada una de las invocaciones, etc.

Antes de permitir que un método comience su ejecución, el planificador interno del objeto consultará los metadatos del objeto para determinar, por ejemplo, si existe algún tipo de restricción respecto a la concurrencia. Es decir, si el método que estamos intentando ejecutar está catalogado como exclusivo, sólo se comenzará su ejecución si el estado del objeto es ocioso, o bien si es él mismo quien está *ocupando* el objeto. En caso contrario, se esperaría a un momento más oportuno.

Este planificador también podría implementar políticas de prioridades respecto a métodos que serían particulares y modificables para cada objeto. Por ejemplo, es posible determinar que un determinado método pase a ejecutarse en el mismo momento en que se invoque y, si es necesario, se paren todos los métodos que ya estén en ejecución.

6.3 Desactivación de métodos en un objeto

Hay dos posibilidades para que un método en ejecución deje de estarlo:

- Finalización por causas naturales, en cuyo caso debe enviarse respuesta a quien haya realizado la invocación al método. Para resolver esto, la máquina abstracta debe ofrecer una instrucción `exitmétodo` que permita devolver la referencia al objeto obtenido como resultado al que haya efectuado la llamada.
- Bloqueo en una llamada a un método de otro o del mismo objeto, lo que lo llevaría al estado inactivo y registra el bloqueo.

En cualquier caso, se estudia la situación de todos aquellos métodos que están inactivos.

En la figura 5 se muestra, con la utilización de un diagrama de transición de estados, el comportamiento de un objeto ante la invocación de un método del mismo.

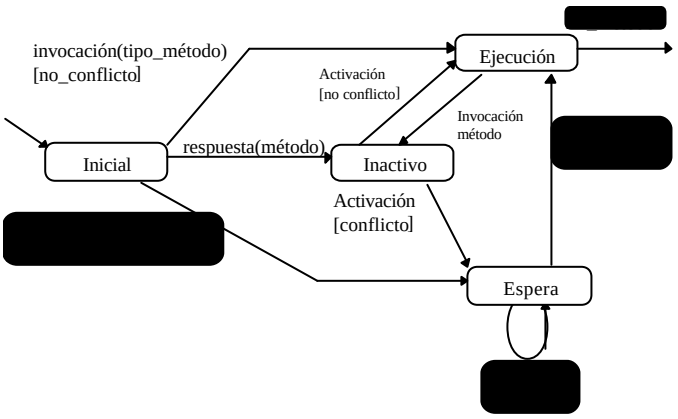

Figura 5. Comportamiento de un objeto ante la llegada de un mensaje

Referencias

[1] Cueva Lovelle, J.M., y otros. Sesión "Sistemas Operativos Orientados a Objetos: Seguridad, Persistencia, Concurrencia y Distribución", II Jornadas sobre Tecnologías Orientadas a Objetos, Oviedo, Marzo 1996.

[2] Campbell, A., Coulson, G., Garcia, F., Hutchinson, y D., H. Leopold, "Designing and Implementing Choices: an Object-Oriented System in C++", Communications of the ACM, Septiembre 1993.

[3] Y. Yokote, "Kernel Structuring for Object-Oriented Operating Systems: The Apertos Approach", Proc. of the International Symposium on Object Technologies for Advanced Software (ISOTAS), 1993.

[4] Mitchell, J.G., Gibbons, J.J., Hamilton, G., Kessler, P.B., Khalidi, Y.A., Kougiouris, P., Madany, P.M, Nelson, M.N., Powell, M.L., y S.R. Radia, "An Overview of the Spring System", Proc. of Compcon Spring 1994, Febrero 1994.

[5] Booch, G. "Object-Oriented Analysis and Design with Applications", 2ª edición, Benjamin Cummings, 1993.

[6] Tajes Martínez, L., "Introducción de la concurrencia en sistemas operativos Orientados a Objetos", II Jornadas sobre Tecnologías Orientadas a Objetos, Oviedo, Marzo 1996.

[7] Chin, R.S., y S.T. Chanson, "Distributed Object-Based Programming Systems", ACM Computing Surveys, Vol. 23, N.1, Marzo 1991.

[8] Papathomas, M., "Concurrency Issues in Object-Oriented Programming Languages", in Object-Oriented Development TR, Centro Universitario de Informática, Universidad de Geneva, ed. Tsichritzis, D., 1989.

[9] Dasgupta, P., LeBlanc, R.J., y W.F. Appelbe, "The Clouds Distributed Operating System", ICDCS, 1988.

[10] D. Hagimont, "Protection in the Guide object-oriented distributed system", ECOOP 1994.

[11] Sun Microsystems Computer Corporation, "The Java Language Specification", URL ftp://ftp.javasoft.com/docs/langspec-1.0.ps.Z, Noviembre 1996.

[12] D. Caromel, "A General Model for Concurrent and Distributed Object-Oriented Programming", SIGPLAN Notices, 24(4), Abril 1989.