UNIVERSIDAD CATÓLICA DE SANTA MARÍA

ESCUELA DE POSTGRADO

MAESTRIA EN INGENIERÍA DE SISTEMAS

TESIS

INFORME DE INVESTIGACIÓN

USO DE MODELO DE COMPONENTES EN PROGRAMACIÓN PARALELA PARA EL DESARROLLO DE APLICACIONES DISTRIBUIDAS, AREQUIPA 2003

Presentado por:

SULLA TORRES, JOSÉ ALFREDO

para optar el Grado Académico de Magister en Ingeniería de Sistemas

Arequipa - Perú
- 2003 -

A mi esposa

A mi hija

A mis padres

A mis hermanos

A mis maestros

con gratitud

ÍNDICE GENERAL

Índice de Figuras

RESUMEN

El desarrollo de software ha pasado de estar en una etapa artesanal, a otra que podríamos denominar de industrialización del software. La mayor influencia en esta evolución la han tenido las técnicas de orientación a objetos y los sistemas distribuidos. Además de los problemas propios que ya plantea el desarrollo de aplicaciones software a partir de objetos (como la reutilización), el uso de sistemas distribuidos introduce nuevas dificultades debidas a la heterogeneidad de sus partes, la falta de una visión global del sistema, los cambios dinámicos en su configuración, o la evolución de los componentes.

Nuestro trabajo propone el uso de un modelo de componentes diseñado para la construcción modular de aplicaciones en este tipo de sistemas, separando los diferentes aspectos que componen los requisitos específicos de las aplicaciones, y en su adición modular e independiente una aceleración en su procesamiento (mediante el uso de *hilos*).

El modelo está soportado por un marco formal de trabajo que permite especificar los conceptos y mecanismos que introduce, y razonar sobre el comportamiento de los componentes y los controladores que se construyen en él. Finalmente, se ha construido con el modelo una aplicación, permitiéndoles beneficiarse de las facilidades que éste ofrece para el desarrollo de aplicaciones en sistemas distribuidos.

ABSTRACT

The software development has passed of being in a handmade stage, to another that we could denominate of industrialization of the software. The biggest influence in this evolution has had it the orientation techniques to objects and the distributed systems. Besides the own problems that it already outlines the development it gives applications software starting from objects (as the reutilization), the use of distributed systems it introduces new due difficulties to the heterogeneity of its parts, the lack gives a global vision it gives the system, the dynamic changes in its configuration, or the evolution gives the components.

Our work proposes the use gives a model gives component designed for the construction to modulate gives applications in this type gives systems, separating the different aspects that compose the specific requirements gives the applications, and in their addition to modulate and independent an acceleration in their prosecution (by means of the use of threads).

The pattern is supported by a formal mark gives work that allows to specify the concepts and mechanisms that it introduces, and to reason on the behavior the components and the controllers that are built in him gives. Finally, it has been built with the pattern an application, allowing them to benefit gives the facilities that this offers for the development it gives applications in distributed systems.

INTRODUCCIÓN

Las tendencias de la tecnología de computación actualmente se encuentran dirigidas principalmente por dos que involucran tanto el *hardware* como el *software*: las arquitecturas paralelas y distribuidas así como la programación con uso de componentes. Hasta fechas recientes, ambas se habían tratado por separado en sus etapas de desarrollo; en ese sentido, el presente trabajo busca combinar todos estos aspectos para acelerar la eficiencia en la ejecución de aplicaciones que cumplan con esos criterios.

El presente informe concluye con la obtención de un modelo de componentes paralelo para aplicaciones distribuidas, ya que como resultado se ha podido mejorar la eficiencia en la ejecución de aplicaciones que tenían un comportamiento paralelo.

El documento posee un capítulo único donde se analiza e interpreta los resultados de la aplicación del modelo, basados en pruebas con estructuras de datos y algoritmos que permiten medir la eficiencia de los mismos. Acompañando a este capítulo se encuentran los modelamiento de casos y de pruebas, en los anexos se consignan el Plan del Proyecto de investigación.

El modelo elaborado ha tenido múltiples limitantes en su desarrollo y culminación, ya que se requirió desarrollar múltiples componentes con distintas variantes en el manejo de hilos y de acceso por parte de los usuarios que los manejan, así que fue preciso utilizar laboratorios de redes.

Es necesario indicar que dicho trabajo de investigación se pudo realizar gracias a la formación impartida y apoyo de la Escuela de Postgrado de la Universidad Católica de Santa María.

CAPÍTULO 1: RESULTADOS

A continuación presentamos los resultados por variables e indicadores.
Las evaluaciones realizadas sobre el uso de componentes en programación paralela para aplicaciones distribuidas son:

- En cuando al uso de componentes en programación paralela

  - Baja granularidad en programación paralela
  - Alta granularidad en programación paralela
  - Speedup en programación paralela
  - Componente de servidor interno
  - Componente de servidor externo

- En cuando aplicaciones distribuidas.

  - Componente Cliente - Servidor Local (Demora)
  - Componente Cliente - Servidor Distribuido (Latencia)

1.1. EN CUANTO AL USO DE COMPONENTES EN PROGRAMACIÓN PARALELA

A continuación presentamos los resultados, variables e indicadores.

En cuanto a la primera variable, modelo de componentes en programación paralela, exhibimos cuadros y gráficas por indicadores.

CUADRO N° 1

Baja granularidad en programación paralela[1]

Aplicación	Tiempo de Ejecución	Speedup
Secuencial	14	0.0101
Paralelo	1379	

Fuente: COMPROPARDIS-2003

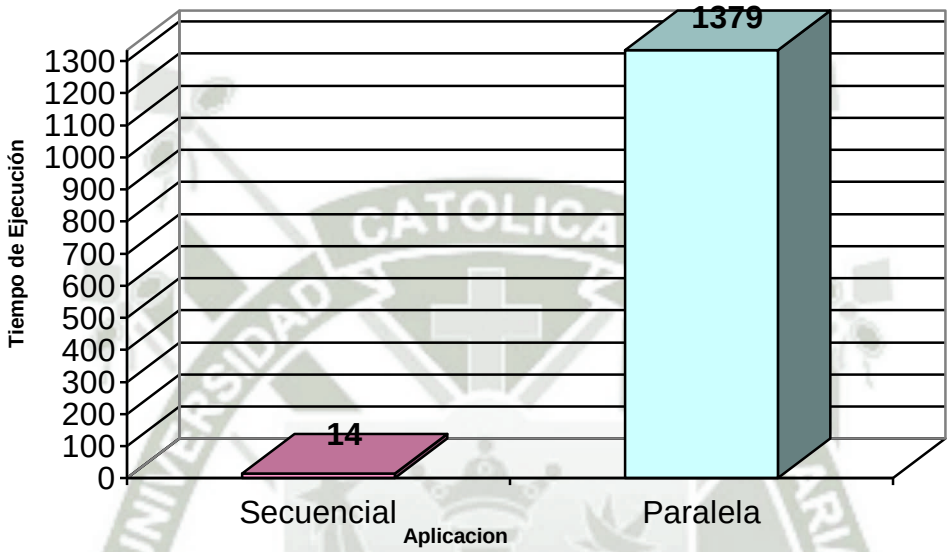
En el cuadro se observa que en una aplicación secuencial donde no existe el tiempo de creación de hilos, con una baja granularidad, la aplicación requiere en promedio alrededor de 14 microsegundos que es muy diferenciado frente a la ejecución paralela con baja granularidad.

Así, el speedup nos da como valor de 0.0101, debido al costo de creación de los hilos.

[1] Anexo 4 - Pruebas de Evaluación: a) En cuanto al uso de componentes - Granularidad baja

GRÁFICO N° 1

Baja granularidad en programación paralela



Fuente: COMPROPARDIS-2003

En la gráfica, los programas secuenciales con baja granularidad experimentan un tiempo de ejecución menor en comparación con los programas paralelos donde se observa que se obtienen tiempos de ejecución mayores.

A lo cual se deduce que ante bajas granularidades (problemas simples) los programas paralelos no son adecuados.

CUADRO N° 2

Alta granularidad en programación paralela[2]

Aplicación	Tiempo de Ejecución	Speedup
Secuencial	34346	25.17
Paralelo	1368	

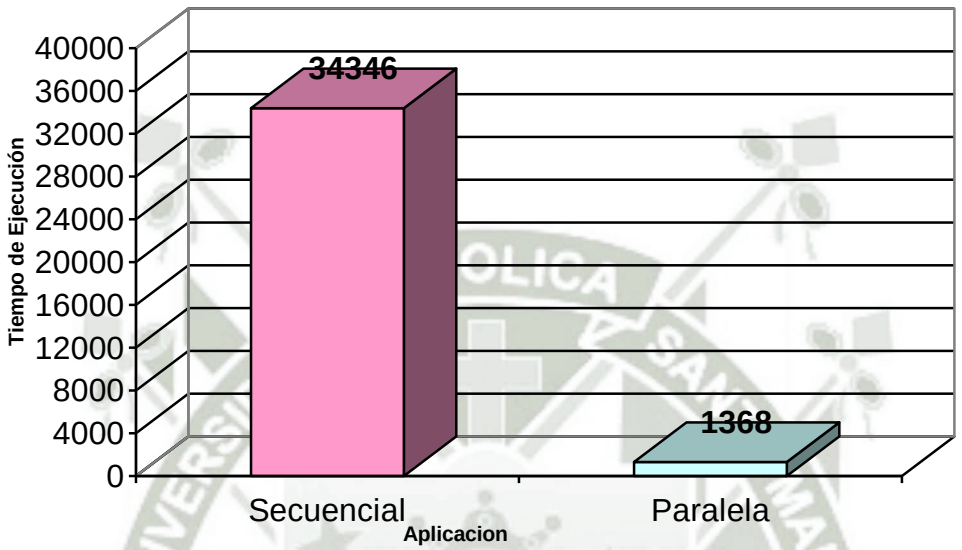
Fuente: COMPROPARDIS-2003

En el cuadro se observa que el tiempo de ejecución de una aplicación secuencial con una alta granularidad tiene en promedio un tiempo de 34346 microsegundos, sin embargo, la aplicación con el uso de hilos mejora significativamente.
De esta manera el indicador speedup refleja que se tiene un valor de 25.17.

[2] Anexo 4 - Pruebas de Evaluación: a) En cuanto al uso de componentes - Granularidad alta

GRÁFICO N° 2

Alta granularidad en programación paralela



Fuente: COMPROPARDIS-2003

En la gráfica los programas secuenciales con alta granularidad experimentan un tiempo de ejecución muy elevados en comparación con los programas paralelos donde se observa que se obtienen tiempos de ejecución óptimos.

CUADRO N° 3

Speedup en programación paralela[3]

Granularidad	Speedup
Baja	0.0101
Alta	25.17

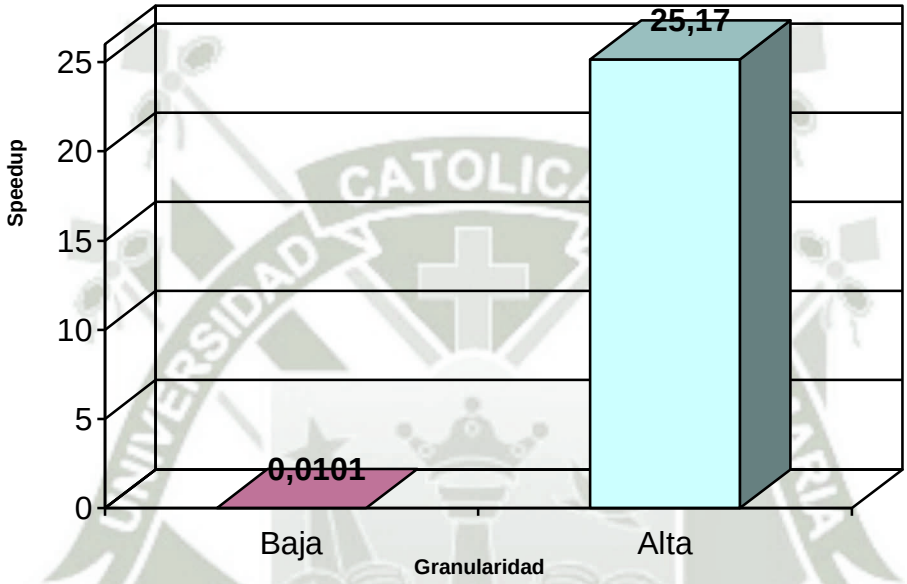
Fuente: COMPROPARDIS-2003

En el Cuadro se observa que una aplicación ante una baja granularidad el speedup es casi nula, mientras que ante una alta granularidad el speedup es muy óptimo, lo que significa que ante aplicaciones de alto procesamiento es adecuado el uso de hilos.

[3] Anexo 4 - Pruebas de Evaluación: a) En cuanto al uso de componentes - Speedup

GRÁFICO N° 3

Speedup en programación paralela



Fuente: COMPROPARDIS-2003

En la gráfica se aprecia que el speedup para aplicaciones con alta granularidad son mejores frente a las aplicaciones que cuentan con bajas granularidades.

CUADRO N° 4

Componente de servidor interno[4]

Argumentos	Tiempo de Ejecución
Sin	189
Con	192

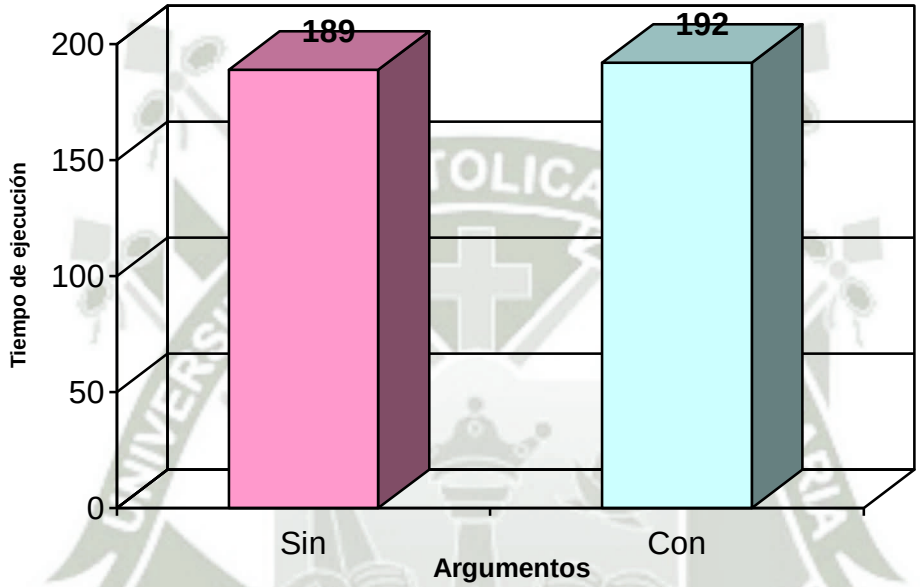
Fuente: COMPROPARDIS-2003

Se observa en el cuadro que la llamada a los métodos de un servidor de proceso interno sin argumentos no difieres demasiado de la llamada a los métodos del mismo servidor con argumento.

[4] Anexo 4 - Pruebas de Evaluación: a) En cuanto al uso de componentes – Servidor de proceso interno

GRÁFICO N° 4

Componente de servidor interno



Fuente: COMPROPARDIS-2003

En la gráfica se aprecia que el tiempo de ejecución de la llamada a métodos con o sin argumentos en un servidor de proceso interno son muy similares, con un ligero exceso para la llamada a métodos con argumentos.

CUADRO N° 5

Componente de servidor externo[5]

Argumentos	Tiempo de Ejecución
Sin	130827
Con	223644

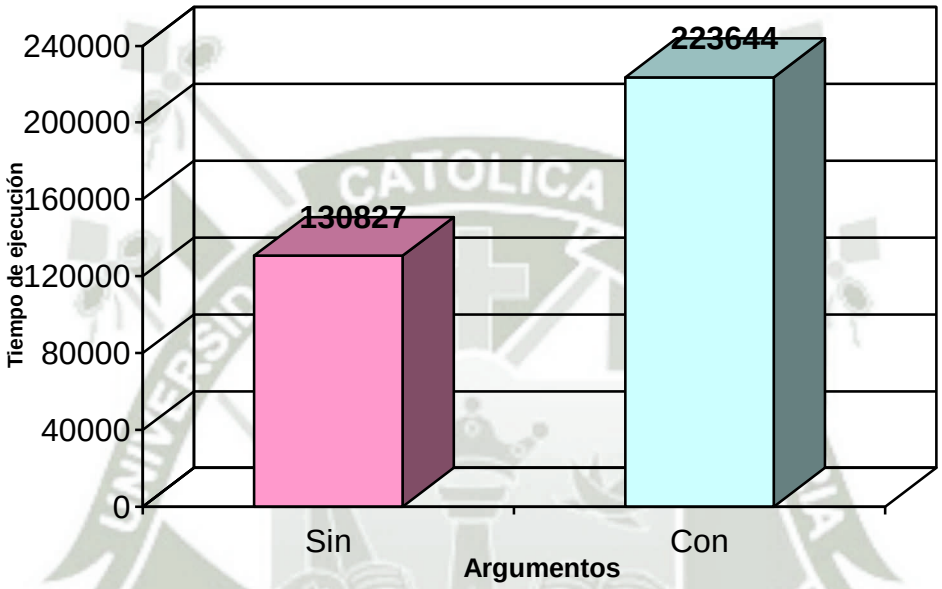
Fuente: COMPROPARDIS-2003

El cuadro muestra los resultados encontrados cuando un cliente accede a un servidor de proceso externo hace que la llamada a métodos con argumentos sea alrededor de 70% mas lenta, la mayor parte de este tiempo se gasta comunicando la llamada al método a través de los procesos.

[5] Anexo 4 - Pruebas de Evaluación: a) En cuanto al uso de componentes – Servidor de proceso externo

GRÁFICO N° 5

Componente de servidor externo



Fuente: COMPROPARDIS-2003

En la gráfica se aprecia que la llamada a métodos con argumentos en un servidor externo es mas lenta que uno sin argumentos.

CUADRO N° 6

Componente de servidor interno y externo[6]

Servidores	Tiempo de Ejecución	
	Sin argumentos	Con argumentos
Servidor Externo	130827	223644
Servidor Interno	189	192

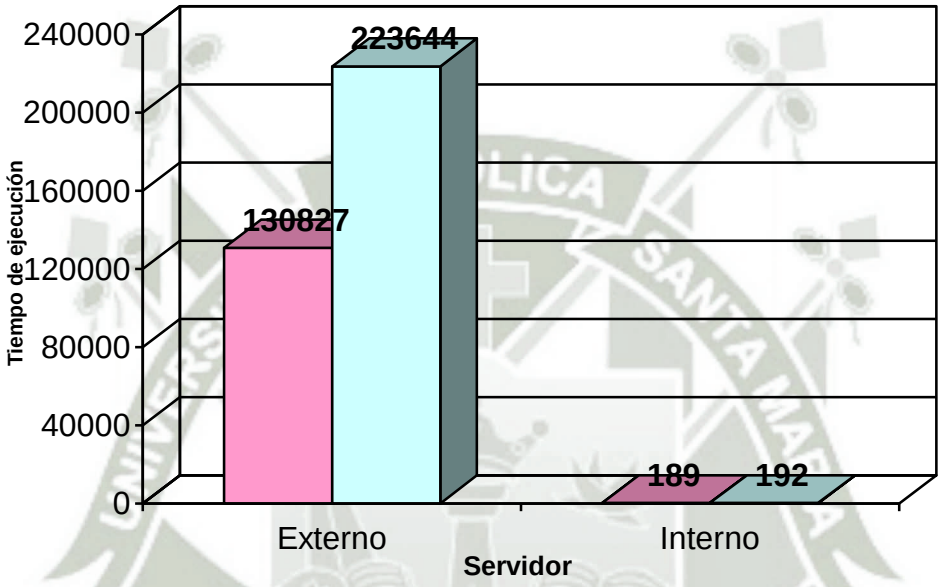
Fuente: COMPROPARDIS-2003

El cuadro muestra los resultados encontrados donde el servidor de proceso externo tiempo con argumentos tiene un tiempo de ejecución más elevado frente a la ejecución sin argumento y sobre todo frente al servidor de proceso interno, pero necesario para la comunicación distribuida, mientras que el servidor de proceso interno la demora es muy baja por lo mismo que es local.

[6] Anexo 4 - Pruebas de Evaluación: a) En cuanto al uso de componentes – Servidor de proceso

GRÁFICO N° 6

Componente de servidor interno y externo



Fuente: COMPROPARDIS-2003

En la gráfica se aprecia que las llamadas al método en un servidor de proceso externo tiene una demora elevada frente al servidor de proceso interno.

1.2. EN CUANTO A APLICACIONES DISTRIBUIDAS.

CUADRO N° 7

Componente Cliente – Servidor Local[7]

Hilos	Tiempo de Ejecución
Sin	72226
Con	58783

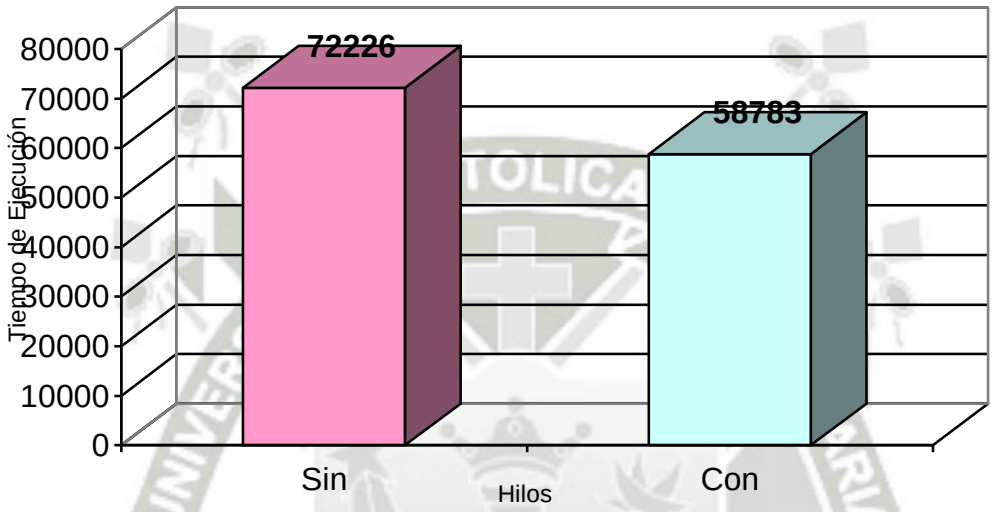
Fuente: COMPROPARDIS-2003

En el cuadro se observa que el uso de componentes con hilos en la comunicación cliente - servidor a nivel local, mejora el tiempo de ejecución en un 25% frente al tiempo de ejecución del componente sin hilos.

[7] Anexo 4 - Pruebas de Evaluación: a) En cuanto a aplicaciones distribuidas – Cliente/Servidor Local

GRÁFICO N° 7

Componente Cliente – Servidor Local



Fuente: COMPROPARDIS-2003

En la gráfica se aprecia que tiempo de ejecución de un componente con un servidor externo con hilos es más económica que el tiempo de ejecución de un servidor externo sin hilos.

CUADRO N° 8

Componente Cliente – Servidor Distribuido[8]

Hilos	Tiempo de Ejecución
Sin	130163
Con	106476

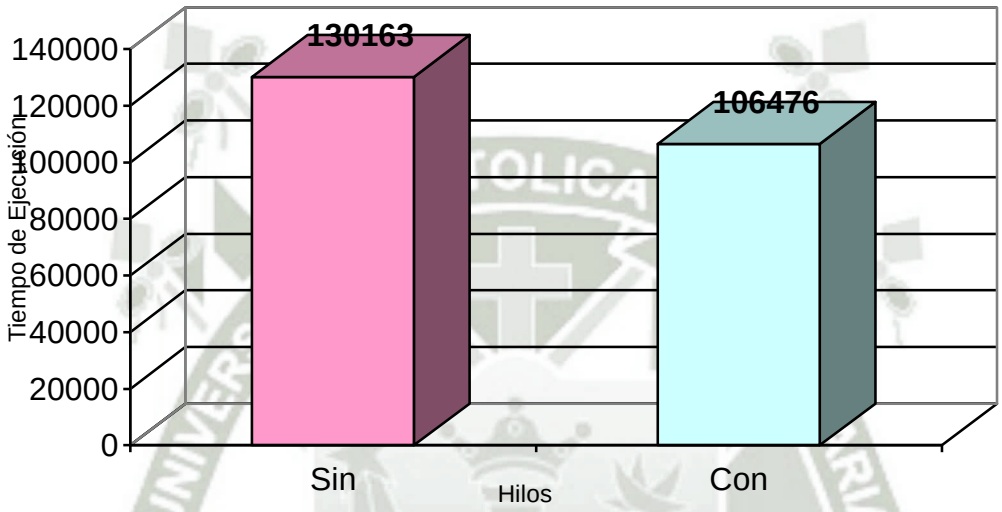
Fuente: COMPROPARDIS-2003

En el cuadro se observa que la demora de comunicación (latencia) cliente - servidor de componentes en una red distribuida, con el manejo de hilos tiene una mejora del 20% aproximadamente en comparación con el manejo sin hilos.

[8] Anexo 4 - Pruebas de Evaluación: a) En cuanto a aplicaciones distribuidas – Cliente/Servidor Distribuido

GRÁFICO N° 8

Componente Cliente – Servidor Distribuido



Fuente: COMPROPARDIS-2003

En la gráfica se aprecia que tiempo de ejecución de un servidor distribuido sin hilos es más costosa que el tiempo de ejecución de un servidor con hilos.

CONCLUSIONES

1. El uso de modelo de componentes son servidores de proceso externo facilita la comunicación de aplicaciones distribuidas por sus características de independientemente del lenguaje de programación y transparencia de localización.
2. El uso de la programación paralela mediante hilos depende de la granularidad, siempre que el tiempo creación se justifique frente al tiempo de ejecución ya que experimenta una mejor utilización de los procesadores.
3. Los cambios entre un servidor de proceso interno y uno de proceso externo son la robustez y tolerancia de errores de uno contra la velocidad y el rendimiento de otro.
4. La programación paralela permite optimizar la demora de comunicación y el tiempo de ejecución de aplicaciones distribuidas combinando el uso de modelo de componentes con el uso de hilos.
5. Queda comprobada la hipótesis, ya que las aplicaciones con el uso de componentes en programación paralela optimizan el manejo de información y operaciones en procesamientos distribuidos.

SUGERENCIAS

1. Orientar a los desarrolladores el uso de la programación paralela para las aplicaciones que así lo permitan, dado que se cuenta con herramientas de programación y hardware que lo soporten, de este modo sus aplicaciones estarán preparadas para soportar distintas arquitecturas de trabajo y abarcarán una amplia gama de aplicaciones orientadas para ello.
2. El dictado de temas de modelo de componentes como parte de cursos de Lenguaje de Programación y Sistemas Distribuidos, debido a su beneficio en su aplicabilidad en el mundo informático actual, de este modo el programador de sistemas conocerá de una mejor forma de solucionar problemas, a la vez que aumentara sus conocimientos en programación.
3. Incluir la nueva versión de componentes en las herramientas de programación de los usuarios, convirtiendo, en un conjunto de servicios fundamental para el desarrollo de aplicaciones de alta disponibilidad, escalables, con buen rendimiento y funcionales en la ejecución de programas.
4. Profundizar la investigación con mejoras a los algoritmos paralelos distribuidos sobre modelo de componentes de tal forma que permita medir los tiempos de ejecución frente casos más profundos de sincronización, o a los distintos protocolos de encaminamiento.

BIBLIOGRAFÍA

[1] Booch Grady, Jacobson Ivan y Rumbaugh Jim, *The Unified Software Development Process*, Addison Wesley, España, 1999.

[2] Ceballos Francisco, *Visual C++ 6.0 Programación Avanzada en Win32*, Ra-Ma, México, 1999.

[3] Coulouris George, Dollimore Jean y Kindberg Tim, *Sistemas Distribuidos Conceptos y Diseño*, 3ra. Edición, Addison Wesley, España, 2001.

[4] Ersek Richard, *Administración de COM+*, Microsoft Corporation, USA, 2000.

[5] Foster Ian, *Diseño y construcción de programas Paralelos*, McGraw Hill, USA, 1998.

[6] Freeze Wayne, *VB Developer's Guide to COM and COM+*, Sybex, USA, 2000.

[7] Gomaa Hassan, *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Addison-Wesley, USA, 2000.

[8] Gordon Alan, *Programación COM y COM+*, Anaya Multimedia, España, 2000.

[9] Jelly Innes, Gorton Ian y Croll Peter, *Software Engineering for Parallel and Distributed Systems*, Chapman & Hall, Great Britain, 1996.

[10] Kruglinski, David, *Programación con Visual C++*, 5ta. Edición, Microsoft Press, USA, 1998.

[11] Lea Doug, *Programación concurrente en Java - principios y patrones de diseño*, 2da. Edición Addison Wesley, España, 2001.

[12] Lester Bruce, *The art of Parallel Programming*, Prentice Hall, USA, 1998.

[13] Lou Baker and Bradley J. Smith, *Parallel Programing*, McGrawHill, USA, 1998.

[14] Pascual Jorge, Charte Francisco y Otros, *Programación avanzada en Windows 2000 con Visual C++ y MFC*, McGrawHill, España, 2000.

[15] Pattison Ted, *Programing Distributed Applications with COM+ and Microsoft Visual Basic 6.0*. Second Edition Microsoft Press, 2000.

[16] Platt David, *Understanding COM+*, Microsoft Press, EU, 1999.

[17] Pinnock Jonathan, *Professional DCOM Application Development*, Wrox Press, USA, 1998.

[18] Raya José Luis y Elena, *Windows 2000 Server, instalación, configuración y administración*, Alfaomega Ra-Ma, México, 2001.

[19] Rofail Ash y Shohoud Yasser, *Mastering COM and COM+*, Sybex, USA, 2000.

[20] Rubin William y Brain Marshall, *Undertanding DCOM*, Prentice Hall, USA, 1999.

[21] Szypersky y Pfister, *Programación Orientada a Componentes*, Wrox, USA, 1998.

[22] Tanenbaum Andrew, *Sistemas Operativos Distribuidos,* Prentice Hall, México, 1996.

[23] Vaughn William, *Programación de SQL Server 7.0 con Visual Basic*, Microsoft Press, España, 1999.

Direcciones de Internet Consultadas

[24] Aplicaciones DCOM. http://www.gsi.dit.upm.es/~jcg/is/...html_doc/oledcom6.htm

[25] Buscador de trabajos de investigación. http://www.researchindex.com

[26] CodeGuru. http://www.codeguru.com

[27] Conceptos y principios de TINA. http://www.tinac.com/95/file.list.html

[28] Confiabilidad de componentes. http://www.eiffel.com/doc/manual/.../trusted/

[29] El Component Object Model, Microsoft. http://www.microsoft.com/com

[30] JavaBeans. http://java.sun.com/beans/

[31] Modelos de componente CORBA. http://www.omg.org

[32] Rational Rose http://www.rational.com

[33] Servicios de componentes en Windows 2000 - COM+. http://www.fcharte.com

ANEXO 1

PROYECTO

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

ESCUELA DE POSTGRADO

MAESTRIA EN INGENIERÍA DE SISTEMAS

TESIS

PROYECTO DE INVESTIGACIÓN

USO DE MODELO DE COMPONENTES EN PROGRAMACIÓN PARALELA PARA EL DESARROLLO DE APLICACIONES DISTRIBUIDAS, AREQUIPA 2003

Presentado por:

SULLA TORRES, JOSÉ ALFREDO

para optar el Grado Académico de Magister en Ingeniería de Sistemas

Arequipa - Perú
- 2003-

PREÁMBULO

Los continuos avances en la Informática y las Telecomunicaciones están haciendo cambiar la forma en la que se desarrollan actualmente las aplicaciones software. En particular, el incesante aumento de la potencia de los ordenadores personales, el abaratamiento de los costos del hardware y las comunicaciones, y la aparición de redes de dato de cobertura global han disparado el uso de los sistemas abiertos y distribuidos. Esto ha provocado entre otras cosas, que los modelos de programación existentes se vean desbordados, siendo incapaces de manejar de forma natural la complejidad de los requisitos que se les exigen a las aplicaciones en dichos sistemas.

Comienzan a aparecer por tanto nuevos paradigmas de programación, como pueden ser la coordinación, la programación paralela, la programación orientada a componentes, o la movilidad, que persiguen una mejora en los procesos de construcción de aplicaciones software. Dentro de uno de ellos, la programación orientada a componentes, se trabaja tanto en nuevos modelos de componentes como en extensiones para los existentes, así como en la estandarización de sus interfaces y servicios, y en la pertinaz búsqueda de cada vez más necesario mercado global de componentes software.

En este proyecto se trata de demostrar la factibilidad de desarrollar aplicaciones distribuidas con un modelo de componentes distribuidos (COM+) y el uso de la programación paralela, para hacerlo más óptimo en el manejo de información.

Surgen muchas preguntas en este entorno: ¿Cómo mantener la compatibilidad entre diversos lenguajes y sistemas? ¿Se podría utilizar un estándar que facilite la reutilización de piezas de software? ¿Cómo optimizar las operaciones en el manejo de información?. El proyecto de investigación tiene la confianza de tener la posibilidad de seguir trabajando para responder a estas preguntas, de forma que satisfagan las necesidades de comunidad informática.

PLANTEAMIENTO TEÓRICO

1. PROBLEMA DE INVESTIGACIÓN

1.1. Enunciado del Problema

Uso de modelo de componentes en programación paralela para el desarrollo de aplicaciones distribuidas, Arequipa 2003.

1.2. Descripción del problema

El problema se ubica en el área de sistemas y en particular en el campo de los modelos y las plataformas distribuidas de componentes, siendo su principal contribución el uso de un modelo de componentes software específicamente diseñado para el desarrollo modular de aplicaciones en sistemas abiertos.

El Modelo surge para aliviar algunas de las limitaciones que presentan los modelos y plataformas de componentes existentes a la hora de tratar la complejidad y los requisitos específicos de este tipo de sistemas, como son la heterogeneidad y la evolución independiente de sus componentes, la falta de visión global de los sistemas de fallos y los retrasos en las comunicaciones, o la necesidad de poder garantizar la seguridad y confidencialidad de datos.

Mi variable independiente es: Uso de componentes en Programación Paralela.
Mis indicadores son: Granularidad y Speedup.
Mi variable dependiente es: Aplicaciones distribuidas.
Mis indicadores son: Demora y Latencia.

La investigación es del Tipo de Documental y de Laboratorio, el Nivel de Investigación es descriptiva-explicativo cuando nos referimos al uso de

modelo de componentes y es de Nivel experimental cuando nos referimos al diseño e implementación del modelo de componentes con programación paralela para aplicaciones distribuidas, además de coyuntural.

1.3. Justificación del problema

Es bastante ambicioso, puesto que estamos hablando de una comunidad empresarial-informática que emplean el procesamiento distribuido para el manejo de su información.

Es factible, primeramente por los conocimientos en programación orientada a componentes, programación paralela y aplicaciones distribuidas en educación superior.

Tiene mucha aplicación en procesamiento distribuido, puesto que el manejo de información a ese nivel es común entre las empresas, y con la Tecnología de Programación orientada a componentes, mejorará el manejo de información. Es inédito, ya que no se han realizado trabajos similares, por lo menos hasta donde hemos podido investigar.

Es completamente moderno puesto que utiliza la tecnología de Programación orientada a componentes, la cual está empezando a tener gran importancia, sobre todo en lo que se refiere a programación distribuida.

Es necesario en nuestra sociedad digital, puesto que la celeridad de procesos en trabajos compartidos es de vital importancia.

2. MARCO CONCEPTUAL

2.1. MODELO DE COMPONENTES

La solución a los problemas planteados viene de mano de los conocidos como *modelos de componentes*. Un modelo de componentes es una especificación en la que se definen, entre otros datos, la estructura de los componentes, la comunicación entre ellos y los medios disponibles para que el desarrollador los manipule desde sus programas o desde entornos de desarrollo creados con este fin.

2.1.1. Componente

Un componente es una unidad de composición de aplicaciones software que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio[9].

2.1.2. Interfaces

Las interfaces de un componente determinan tanto las operaciones que el componente implementa como las que precisa utilizar de otros componentes durante su ejecución. Y los requisitos determinan las necesidades del componente en cuanto a recursos, como por ejemplo las plataformas de ejecución que necesita para funcionar.

Centrándonos más en el aspecto técnico y en el lenguaje que utilizamos, podríamos decir que una interfaz es una tabla formada por punteros a métodos implementados por un cierto objeto. En cierta forma, una interfaz es como una clase abstracta, es decir, una clase en

[9] Szypersky y Pfister, *Programación Orientada a Componentes*, 1998.

la que todos los métodos son virtuales puros.

Al crear un componente COM hay que decidir qué interfaces son las que se implementarán, ya sean interfaces estándar predefinidas o bien interfaces creadas por nosotros mismos. Implementar una interfaz significa que el objeto que vamos a crear tendrá que codificar todos los métodos indicados en ella.

2.1.3. Component Object Model

Define la forma de sus interfaces y los mecanismos para interconectarlos destinada a la comunicación de objetos independientes de plataforma y de lenguajes de programación concretos (ejem. COM, JavaBeans o CORBA).

COM es una especificación que define un modelo de componentes binarios. En ella se establece la estructura que debe tener un objeto COM y la forma de acceder a los servicios que implementa. No se entra, por el contrario, en cómo deben definirse o implementarse esos objetos al nivel de código fuente.

Consecuentemente, un objeto COM puede crearse usando cualquier lenguaje, ya sea orientado a objetos o no, y desarrollado en cualquier plataforma, aunque lo habitual es que ésta sea Windows. Lo único importante es que la imagen binaria de ese componente cumpla con el modelo de COM, nada más y nada menos[10].

La ventaja de un modelo de componentes como COM respecto a objetos generados con un lenguaje como C++ es clara. Un objeto COM podrá ser usado desde cualquier lenguaje y en cualquier sistema que cumpla con las especificaciones COM, puesto que se trata de un estándar binario.

[10] Pascual Jorge, *Programación Avanzada con Visual C++*. P.712

2.1.4. Estructura de COM

El concepto fundamental es la separación entre la interfaz pública de un componente y su implementación.

Un componente COM es un objeto que implementa una o más interfaces, entendiendo como tales simples tablas de punteros a métodos. El objeto tiene que asociar a cada uno de esos métodos el código apropiado para conseguir la función que se espera de ellos, realizando lo que se conoce como implementación.

2.1.5. Plataforma de Componentes

Basada en un modelo de componentes concreto, una plataforma de componentes es un entorno de desarrollo y de ejecución de componentes que permite aislar la mayor parte de las dificultades conceptuales y técnicas que conlleva la construcción de aplicaciones basadas en los componentes de este modelo. En este sentido, podemos definir una plataforma como una implementación de los mecanismos del modelo, junto con una serie de herramientas asociadas.

2.1.6. Plataforma de Componentes Distribuidos

Es un Marco de Trabajo Distribuido diseñado para integrar componentes y aplicaciones software en ambientes distribuidos, permitiendo la modularidad y reutilización en el desarrollo de nuevas aplicaciones. Este tipo de marcos ofrecen un entorno de desarrollo y de ejecución de componentes software que permite aislar la mayor parte de las dificultades conceptuales y técnicas que conlleva la construcción de aplicaciones basadas en componentes. Cada plataforma se basa en los estándares y mecanismos definidos por su

modelo de componentes base, en este sentido, podemos definir una plataforma como una implementación de los mecanismos del modelo, junto con una serie de herramientas asociadas[11].

2.1.7. Contenedores

Los componentes suelen existir y cooperar dentro de *contenedores*, entidades software que permiten contener a otras entidades, proporcionando un entorno compartido de interacción. Se aplica sobre todo para objetos y componentes visuales, que contienen a su vez a otros objetos visuales. Por ejemplo, un control Actives puede ser un contenedor de otros controles ActiveX.

Normalmente la relación entre componentes y sus contenedores se establece mediante eventos. Como ejemplo, pensemos en un contenedor, que se registra al ser creado en un escritorio o ventana (en general, un *drop site*, DS)para ser informado de cuándo alguien deposita ahí un componente. Cuando un usuario hace una operación que arrastra y suelta (*drag and drop*) un componente y lo deposita en el DS, el mecanismo de eventos del DS se lo notifica al contenedor, invocando el procedimiento que previamente éste registró, y pasándole como parámetro una referencia (*handle*) al componente que ha sido dejado en el DS. Ante esta notificación, el contenedor suele cambiar el aspecto del icono del componente para indicar que la operación de arrastre (*drag and drog*) ha terminado con éxito y pasarle al componente una referencia a los servicios que ofrece el contenedor, para que el componente pueda utilizarlos.

[11] Ceballos Francisco, *Visual C++ 6.0 Programación Avanzada en Win32*, Ra-Ma, México, 1999

2.2. PROGRAMACIÓN PARALELA

Un programa paralelo debe proveer una secuencia de operaciones para cada procesador haciéndolo en paralelo, incluyendo operaciones que coordinen e integren los procesadores separados en una coherencia computacional. Esta necesidad de creación y coordinación de muchas actividades computacionales paralelas añade una nueva dimensional proceso de la programación computacional. Los algoritmos para problemas específicos deben ser formulados de una manera que produzcan muchas operaciones paralelas[12].

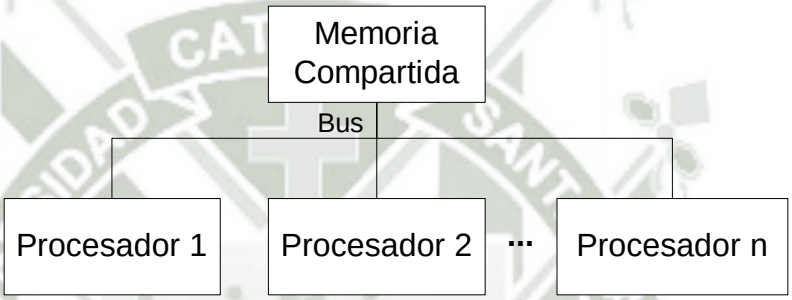


Figura 1: Esquema de Procesamiento Paralelo. Fuente Propia

2.2.1. Paralelismo de datos

El paralelismo de datos es la ejecución de la misma operación en cada componente de la estructura de datos en paralelo. Cada elemento es procesado en paralelo o la estructura de datos es particionado en grupos de elementos individuales, con cada grupo siendo procesado en paralelo. Los algoritmos de datos paralelos usualmente usan una sentencia FORALL para la creación de procesos, la cual es la forma paralela de un bucle, donde las iteraciones son ejecutadas todas en paralelo. Estas estructuras de datos usualmente son grandes arrays multidimensionales.

[12] Foster Ian, *Diseño y Construcción de Programas Paralelos*, 1998.

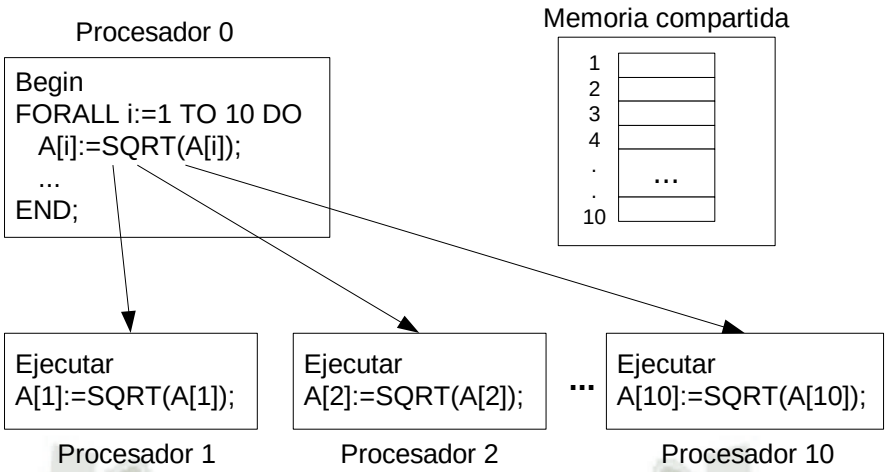


Figura 2: Creación de Procesos con Forall. Fuente[13]

2.2.2. Ley de AMDAHL´S

Uno de las cosas que puede limitar severamente la máxima velocidad (*speedup*) en programas paralelos, es la porción de código secuencial comparado con la porción paralela. Este hecho se expresa en una fórmula llamada *Ley de Amdahl´s*. Se asume que la computación tiene un total de *n* operaciones, cada uno requiere una unidad de tiempo para su ejecución. Asuma que una fracción *f* de operaciones debe ejecutarse secuencialmente. (donde $0<f<1$). Por tanto *fn* operaciones son hechas secuencialmente, y $(1-f)n$ operaciones son hechas paralelamente. Si este programa se ejecuta en cuna computadora paralela con *p* procesadores, entonces el tiempo de ejecución total mínima es $fn+(1-f)n/p$. El Tiempo total de ejecución de este programa en una computadora secuencial es simplemente *n*. Dividiendo el Tiempo de ejecución Secuencial entre el mínimo Tiempo de ejecución Paralela, dará el Máximo Speedup, lograda por una computadora paralela con *p* procesadores, como sigue:

$$Máximospedup = \frac{1}{f + \frac{1-f}{p}}$$

[13] Lester Bruce, *The art of Parallel Programming*, Prentice Hall, USA, 1998.

2.2.3. Comunicación de Procesos

Los algoritmos paralelos consideran la comunicación e iteración de los procesos, escriben un valor que es proyectado para el uso de otros procesos. Para ello existen las denominadas variables de canal que se usa para la comunicación y sincronización de los procesos. La variable de canal, colecciona valores de datos de procesos de escritura y los *canaliza* hacia las entradas de procesos lectura. Las variables de canal permiten a los procesos paralelos intercambiar y compartir valores de datos en una forma controlada.

En algunos casos cuando algunos procesos están en comunicación, un proceso estará enviando no sólo un único valor para otros procesos, sino toda una secuencia de valores. Este tipo de interacción es algunas veces es llamada Productor-Consumidor, donde en el proceso productor calcula datos y valores los cuales los envía al proceso consumidor para cálculos futuros. Conceptualmente, Un canal actúa como una cola FIFO de valores. A medida que los valores son escritos en el canal, estos son guardados en una cola hasta que sean leídos por algún otro proceso.

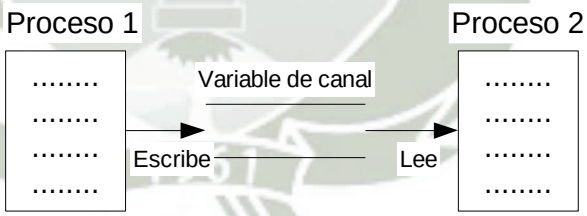


Figura 3: Canales de Comunicación. Fuente Propia

2.2.4. Datos compartidos

Cuando los procesos paralelos acceden a datos compartidos es importante para las operaciones ejecutadas sobre estos datos no sean incompatibles con otras operaciones como resultado del paralelismo, cada operación ejecutada por un proceso debe mantener su propia

integridad, respecto a la operación que esta siendo ejecutada por otro proceso. Una importante característica es la llamada *Spinlocks*, las cuales son usadas para ayudar a coordinar la distribución de valores de datos y estructuras de datos por procesos paralelos, mediante bloqueos (Lock).

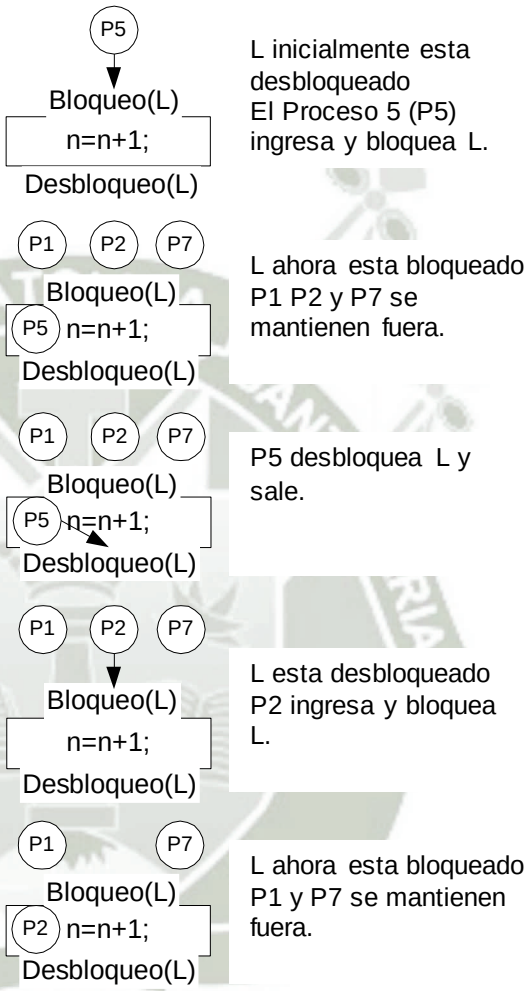


Figura 4: Efecto Spinlock. Fuente Propia

2.2.5. Sincronización Paralela

Muchos procesos paralelos pueden ser creados para trabajar en partes diferentes de un arreglo. Sin embargo, después de cada iteración, los procesos pueden ser sincronizados ya que los valores producidos por un proceso son usados por otros procesos en la siguiente iteración. Este estilo de programación paralela es la *sincronización paralela*.

Una *Barrera* es un punto en el programa donde los procesos paralelos esperan a cada otro. Los primeros procesos a ejecutar la sentencia Barrera simplemente esperarán hasta que todos los demás procesos lleguen. Después todos los demás procesos han alcanzado la sentencia Barrera, luego son liberados para continuar la ejecución paralela. El efecto de una Barrera se ilustra en la siguiente figura:

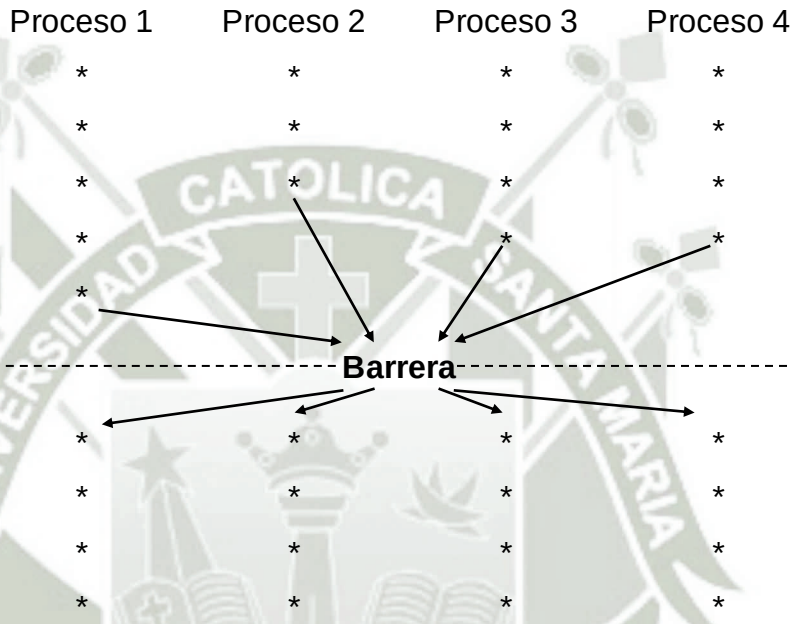


Figura 5: Sincronización de Barrera. Fuente[14]

2.2.6. Paso de Mensajes

En un sistema con múltiples computadores (*multicomputador*), el programador debe visualizar su programa como una colección de procesos, cada uno con sus propias variables locales privadas, más la habilidad de enviar y recibir información de otros procesos mediante el *paso de mensajes*. En este estilo de *paso de mensajes* de programación paralela, los procesos no tienen acceso a ninguna variable compartida común.

[14] Lou Baker and Bradley J. Smith, *Parallel Programing*, McGrawHill, USA, 1998

En un multicomputador los procesadores están conectados por una red, que puede transmitir bloques de información de un proceso a otro. Toda interacción entre estos procesadores es mediante el envío de mensajes a través de esta red de interconexión.

En una implementación multicomputador, la comunicación entre procesos se da mediante *puertos* asociados a cada proceso, que tienen la restricción de que solo un proceso puede recibir mensajes de un puerto dado.

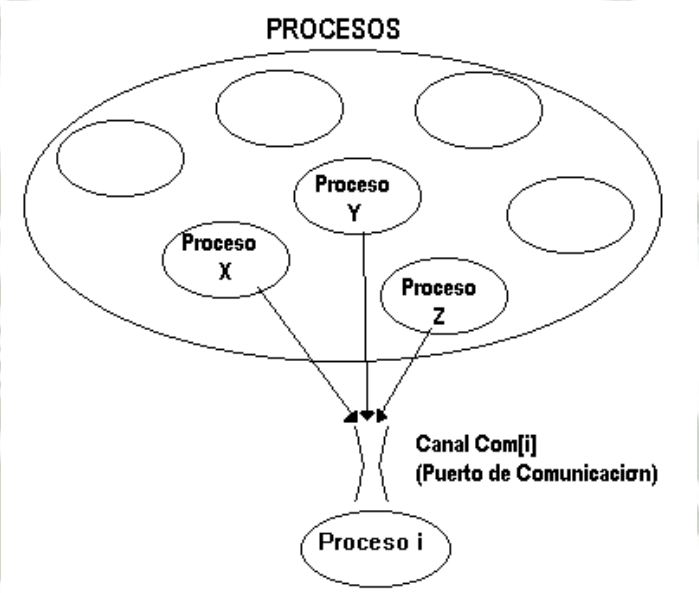


Figura 6: Canales de Comunicación. Fuente Propia

2.2.7. Particionamiento de Datos

Los programas de aplicación paralela siempre involucran grandes arreglos multidimensionales, los cuales forman la base para el paralelismo de datos. Sin embargo cuando estos principios son aplicados a multicomputadores la naturaleza distribuida de la memoria requiere que la información compartida se "particione" en porciones separadas los cuales son distribuidas entre las memorias del multicomputador. Un proceso paralelo es asignado para operar sobre cada porción individual de la información.

2.2.8. Trabajadores Replicados

En el paradigma de Trabajadores Replicados, se usa una estructura abstracta de datos llamada *Repositorio de Trabajo* (Work Pool). Un Repositorio de trabajo es una colección de descriptores de tareas, donde cada descriptor especifica una tarea computacional particular que puede ser habilitada por cualquiera de los trabajadores. Cuando un proceso trabajador esta ocioso, este recupera un nuevo descriptor de tarea del Repositorio de Trabajo.

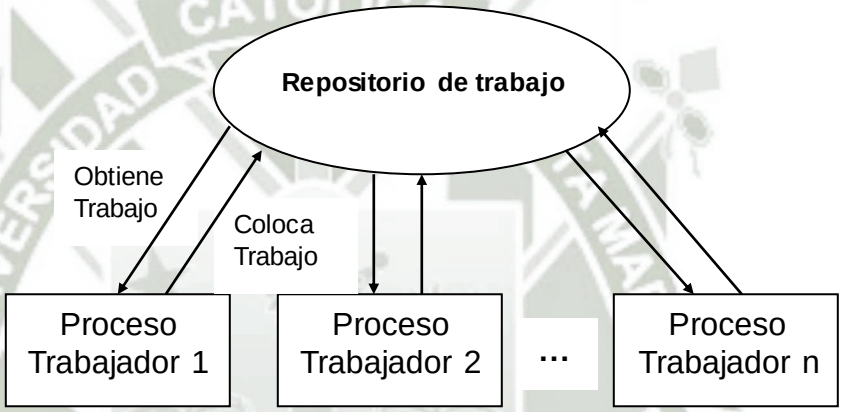


Figura 7: Repositorio de Trabajo. Fuente[15]

Para implementar a los Trabajadores Replicados se requiere de un *algoritmo de finalización distribuida*. El Repositorio de trabajo esta completamente particionado entre los trabajadores, y por eso cada elemento del trabajo debe ser enviado directamente a un trabajador específico. En los trabajadores replicados a menudo sus trabajadores no son indistinguibles, y esto es necesario para ciertos elementos de trabajo para ir a trabajadores específicos.

[15] Foster Ian, *Diseño y construcción de programas Paralelos*, McGraw Hill, USA, 1998

2.3.PROGRAMACIÓN ORIENTADA A COMPONENTES

2.3.1. Concepto

La Programación Orientada a Componentes (POC), aparece como una variante natural de la programación orientada a objetos (POO) para los sistemas abiertos, en donde la POO presenta algunas limitaciones; por ejemplo, no permite expresar claramente la distinción entre los aspectos computacionales y meramente composicionales de la aplicación, no define una unidad concreta de composición independiente de las aplicaciones (los objetos no lo son claramente), y define interfaces de demasiado bajo nivel como para que sirvan de contratos entre las distintas partes que deseen reutilizar objetos[16].

La POC nace con el objetivo de construir un mercado global de componentes software, cuyos usuarios son los propios desarrolladores de aplicaciones que necesitan reutilizar componentes ya hechos y probados para construir sus aplicaciones de forma más rápida y robusta.

Las entidades básicas de la POC son los componentes, en el mismo sentido como cajas negras que encapsulan cierta funcionalidad y que son diseñadas para formar parte de ese mercado global de componentes, sin saber ni quién los utilizará, ni cómo, ni cuando. Los usuarios conocen acerca de los servicios que ofrecen los componentes a través de sus interfaces y requisitos, pero normalmente ni quieren ni pueden modificar su implementación.

Entre los conceptos básicos que intervienen en la POC, y que permiten diferenciarla del resto de los paradigmas de programación, se encuentran los siguientes:

[16] Szypersky y Pfister, *Programación Orientada a Componentes*, Wrox, USA, 1998.

Composición Tardía: Dícese de aquella composición que se realiza en un tiempo posterior al de la compilación del componente, como puede ser durante su enlazado, carga o ejecución, y por alguien ajeno a su desarrollo, es decir, que sólo conoce al componente por su interfaz o contrato, pero no tiene por qué conocer ni sus detalles de implementación, ni la forma en que fue concebido para ser usado.

Entornos: Un entorno es el conjunto de recursos y componentes que rodean a un objeto o componente dado, y que definen las acciones que sobre él se solicitan, así como su comportamiento. Se pueden definir al menos dos clases de entornos para los componentes: el entorno de *ejecución* y el de *diseño*. El primero de ellos es el ambiente para el que se ha construido el componente, y en donde se ejecuta normalmente. El entorno de diseño es un ambiente restringido, que se utiliza para localizar, configurar, especializar y probar los componentes que van a formar parte de una aplicación, y en donde los componentes que van a formar parte de una aplicación, y en donde los componentes han de poder mostrar un comportamiento distinto a sus comportamiento normal durante su ejecución.

Eventos: Mecanismo de comunicación por el que se pueden propagar las situaciones que ocurren en un sistema de forma asíncrona. La comunicación entre emisores y receptores de los eventos se puede realizar tanto de forma directa como indirecta. Los eventos suelen ser emitidos por los componentes para avisar a los componentes de su entorno de cambios en su estado o de circunstancias especiales, como pueden ser las excepciones.

Reutilización: Habilidad de un componente software de ser utilizado en contextos distintos a aquellos para los que fue diseñado (reutilizar *no* significa usar más de una vez). Existen varias modalidades de reutilización, dependiendo de la cantidad de información y posibilidades de cambio que permita el componente a ser reutilizado.

Contratos: Especificación que se añade a la interfaz de un componente y que establece las condiciones de uso e implementación que ligan a los clientes y proveedores del componente. Los contratos cubren aspectos tanto funcionales (semántica de las operaciones de la interfaz) como no funcionales (calidad de servicio, prestaciones, fiabilidad o seguridad).

Polimorfismo: Habilidad de un mismo componente de mostrarse de diferentes formas, dependiendo del contexto; o bien la capacidad de distintos componentes de mostrar un mismo comportamiento en un contexto dado. Ambas acepciones representan los dos lados de una misma moneda. En POO el polimorfismo se relaciona con la sobreescritura de métodos y la sobrecarga de operadores. Sin embargo en POC muestra nuevas posibilidades:

- La *reemplazabilidad*, o capacidad de un componente de reemplazar a otro en una aplicación, sin romper los contratos con sus clientes (también conocido por polimorfismo de *subtipos* o de *inclusión*).
- El polimorfismo *paramétrico*, o implementación genérica de un componente. Similar en concepto a los *generics* de Ada o a los *templates* de C++, el polimorfismo paramétrico no se resuelve como ellos en tiempo de compilación sino en tiempo de ejecución.

Seguridad: Por seguridad en este contexto se entiende la garantía que debe ofrecer un componente de que respeta sus interfaces y contratos, y constituye el concepto básico sobre el que puede garantizarse la seguridad de un sistema. Puede hacerse una distinción entre seguridad a nivel de *tipos* y a nivel de *módulos*. La primera se refiere a que la invocación de los servicios de un componente se realice usando parámetros de entrada de los tipos adecuados y que los servicios devuelvan también valores de tipo adecuado (o subtipos suyos: *covarianza*). La seguridad a nivel módulo se refiere a que sólo se utilicen los servicios ajenos al componente que hayan sido declarados por él, y no otros[17].

[17] Szyperski y Pfister, *Programación en Componentes*, Wrox, USA, 1998

Reflexión: Es la habilidad de una entidad software de conocer o modificar su estado. A la primera forma se le denomina *reflexión estructural*, y a la segunda *reflexión de comportamiento*.

Dentro de la POC se está trabajando en varios fuentes: la adecuación de los lenguajes de programación y modelos de objetos existentes para que incorporen estos conceptos; el diseño de nuevos lenguajes y modelos de componentes; la construcción de herramientas de desarrollo y marcos de trabajo para componentes; y la aplicación de técnicas formales para razonar sobre las aplicaciones desarrolladas a base de componentes.

POC es una disciplina joven y por tanto en la que los resultados obtenidos hasta ahora se centran más en la identificación de los problemas que en la resolución de los mismos. En ese sentido, destacaremos los siguientes retos y problemas con los que se enfrenta actualmente:

a) *Clarividencia:* Este problema se refiere a la dificultad con la que se encuentra el diseñador de un componente al realizar su diseño, pues no conoce ni quién lo utilizará, ni cómo, ni en que entorno, ni para que aplicación; además, se encuentra también con la paradoja de "*maximizing reuse minimizes use*". Este problema está intrínsecamente ligado a la composición tardía y reusabilidad de los componentes.
b) *Evolución de los componentes:* La gestión de la evolución es un problema serio, pues en los sistemas grandes han de poder coexistir varias versiones de un mismo componente. Existen distintos enfoques para abordar este problema (desde la inmutabilidad de interfaces de COM a la integración de interfaces que propugna CORBA), aunque ninguno totalmente satisfactorio.
c) *Percepción del entorno:* Esta es la habilidad de un objeto o componente de descubrir tanto el tipo de entorno en donde se está ejecutando (de diseño o de ejecución), como los servicios y recursos disponibles en él. La inspección y la reflexión estructural son dos mecanismos comúnmente utilizados para implementar esta habilidad.

d) *Particularización:* Cómo particularizar los servicios que ofrece un componente para adaptarlo a las necesidades y requisitos concretos de nuestra aplicación, sin poder manipular su implementación. La reflexión de comportamiento es la técnica comúnmente utilizada, componiendo el componente con envolventes que le proporcionan la funcionalidad apropiada. La composición simultánea de envolventes y el estudio de las propiedades del nuevo componente a partir de las del antiguo son campos de investigación actualmente abiertos.

e) *Falta de soporte formal:* Por otro lado, la POC también se encuentra con otro reto añadido, como es la dificultad que encuentran los métodos formales para trabajar con sus peculiaridades, como puede ser la composición tardía, el polimorfismo o la evolución de los componentes.

2.3.2. Servidor COM

Los objetos C++ no existen individualmente de forma aislada sino que forman parte de aplicaciones, librerías de enlace dinámico, componentes o servicios. De igual manera, los componentes COM no existen aisladamente, sino que forman parte de lo que se llama servidores COM.

Básicamente, existen dos clases de servidores COM posibles: ejecutables y librerías de enlace dinámico.

El servidor no es simplemente un contenedor que aloja el código de los componentes que contiene, sino que además, debe ocuparse de otras tareas igualmente importantes. Una de ellas consiste en realizar el registro de los componentes, otra es crear la factoría o factorías que permitirán a los clientes crear dichos componentes[18].

Los componentes alojados en un ejecutable pueden ser utilizados por cualquier cliente igual que los contenidos en librerías de enlace dinámico, la diferencia es que en el primer caso el componente en sí se ejecuta en su

[18] Pascual Jorge. *Programación Avanzada Visual C++*. P.719

propio espacio de proceso, mientras que en el segundo lo hace en el espacio de proceso del cliente que lo está utilizando.

2.3.3. Estructura de un Servidor

El contenedor de primer nivel, que es el servidor COM, es como una caja que cuenta básicamente con tres elementos: clases de objetos, factorías y funciones adicionales que facilitan operaciones como el registro del servidor o la obtención de una factoría para un cierto objeto.

Los dos primeros elementos, clases de objetos y factorías, son comunes a todos los servidores sin importar su tipo. Los servidores adicionales se implementan de forma diferente según sea el servidor una librería de enlace dinámico o un ejecutable. En el primer caso, por ejemplo, existe una serie de funciones que han de ser exportadas, como DllRegisterServer() o DllGetClassObject(). Un ejecutable no puede exportar funciones como una DLL, por lo que esos mismos servicios se implementan de una forma alternativa.

2.4. PROGRAMACIÓN EN SISTEMAS DISTRIBUIDOS.

Las aplicaciones distribuidas introducen nuevos problemas de diseño y desarrollo. Algunas aplicaciones que son intrínsecamente distribuidas como es el caso de juegos multiusuario, teleconferencias y chats requieren una infraestructura fiable para su funcionamiento distribuido. Existen otras aplicaciones que aunque son distribuidas, dado que tienen al menos dos componentes ejecutándose en máquinas diferentes, no fueron diseñadas para funcionar como tales, lo que dificulta su ampliación y crecimiento. Cualquier aplicación para trabajo en grupo y la mayor parte de las aplicaciones cliente/servidor controlan la forma en que sus usuarios se comunican y cooperan entre sí. Un diseño distribuido de estas aplicaciones de forma que los

componentes se ejecuten en los lugares más adecuados beneficia al usuario y optimiza la utilización de los recursos. Este enfoque permite que la aplicación pueda adaptarse a diferentes clientes con diferentes capacidades ejecutando sus componentes en la parte del cliente cuando sea posible y en el servidor en caso contrario. La facilidad de crecimiento de las aplicaciones distribuidas es muy superior a la de las aplicaciones monolíticas, ya que con un diseño adecuado un solo servidor puede comenzar ejecutando todos los componentes y trasladar algunos de ellos a otras máquinas de menor coste conforme aumenta la carga.

2.4.1. DCOM

La arquitectura DCOM[19] *(Distributed Component Object Model)* es una extensión de COM que permite la comunicación entre objetos situados en diferentes máquinas a través de distintos tipos de redes (*LAN*, *WAN* e incluso *Internet*). Al ser DCOM una evolución natural de COM es posible utilizar todas aquellas aplicaciones, componentes, herramientas y conocimiento previos para trabajar ahora en un entorno distribuido. DCOM pretende solucionar los problemas más comunes que surgen en un entorno de trabajo distribuido:

- Fiabilidad ante posibles fallos en la red.
- Fiabilidad ante posibles fallos en el hardware.
- Eficiencia en términos de carga de la red.
- Posibilidad de trabajar con máquinas de diferentes prestaciones situadas en distintas áreas geográficas.
- Posibilidad de instalación tanto en grupos de trabajo pequeños como grandes.

2.4.2. Arquitectura DCOM

En los sistemas operativos de hoy en día los procesos están aislados unos

[19] Rubin William y Brain Marshall, *Undertanding DCOM*, Prentice Hall, USA, 1999

de otros. Un cliente que necesite comunicarse con un componente de otro proceso no puede realizar una llamada directa, sino que tiene que hacer uso de algún tipo de comunicación entre procesos proporcionada por el sistema operativo. COM/DCOM permite establecer esta comunicación de una manera completamente transparente interceptando la llamada del cliente y dirigiéndola a un componente en otro proceso.

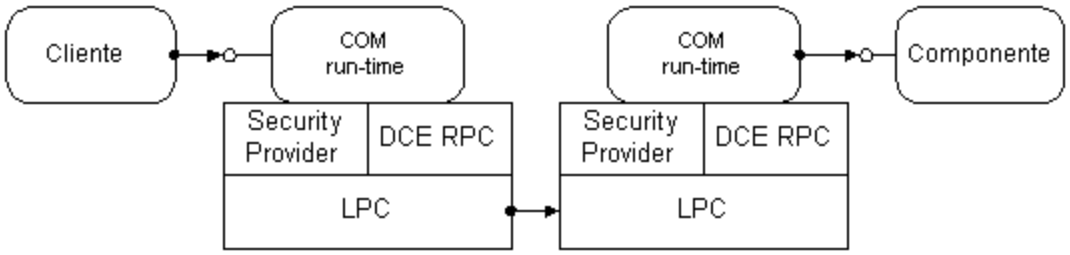


Figura 8: Componentes COM en diferentes procesos. Fuente Propia

Cuando cliente y componente residen en diferentes máquinas, DCOM simplemente sustituye la comunicación entre procesos por un protocolo de red, de modo que ninguno de los dos advierte este cambio. COM proporciona servicios orientados a objetos a clientes y componentes, y utiliza *DCE RPC* y varios sistemas de seguridad (*Security Providers*) para generar paquetes de red. *DCE RPC*, del que DCOM toma el concepto de *GUID* (*Globally Unique Identifier*), define un estándar para la conversión de parámetros y estructuras de datos almacenadas en memoria para formar paquetes de red. Su formato de representación de datos *NDR* (*Network Data Representation*) es independiente de la plataforma.

2.4.3. Independencia de Localización

Al implementar una aplicación distribuida real sobre una red pueden aparecer una serie de restricciones al diseño:

- Algunos componentes sólo pueden ser ejecutados en algunas máquinas o en determinadas localizaciones.

- Los componentes que interactúan más a menudo deberían estar situados "más cerca".
- Muchos componentes pequeños incrementan la flexibilidad del diseño, pero generan mayor tráfico en la red.
- Pocos componentes grandes reducen el tráfico en la red pero también la flexibilidad del diseño.

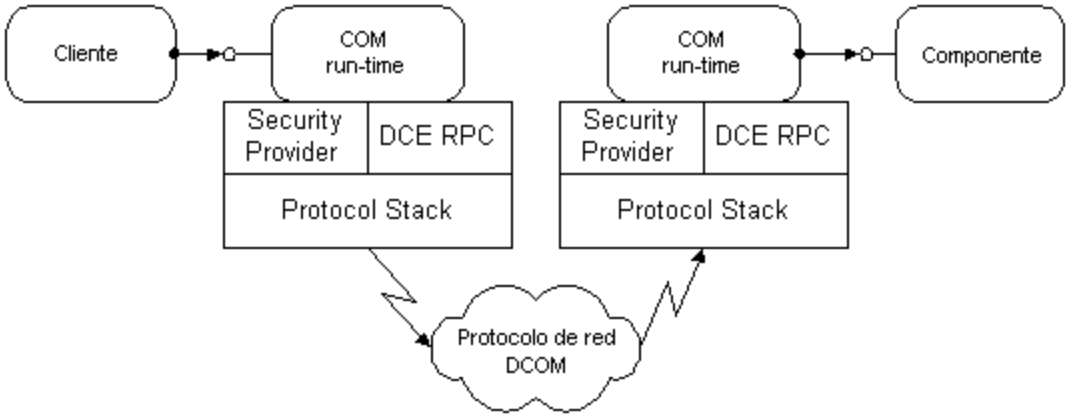


Figura 9: Componentes COM en diferentes máquinas. Fuente Propia

Con DCOM es fácil evitar estas restricciones, ya que oculta la localización de los componentes, y la forma en la que el cliente se conecta a ellos y realiza una llamada a sus funciones es siempre la misma. DCOM no sólo no requiere cambios en los programas fuente, sino que tampoco es necesario recompilarlos, porque la reconfiguración cambia la forma en que los componentes se conectan entre ellos. Esto simplifica en gran medida la tarea de distribuir los diferentes componentes de la aplicación con el fin de optimizar el rendimiento.

2.4.4. Facilidad de Crecimiento (Scalability)

Un factor crítico de las aplicaciones distribuidas es su capacidad de adaptarse al aumento del número de usuarios, volumen de datos o funcionalidades requeridas. La aplicación debería ser pequeña y rápida

cuando la demanda es baja, pero también ser capaz de atender una demanda creciente manteniendo un rendimiento y una fiabilidad aceptables. DCOM proporciona algunos elementos que facilitan la capacidad de crecimiento de las aplicaciones.

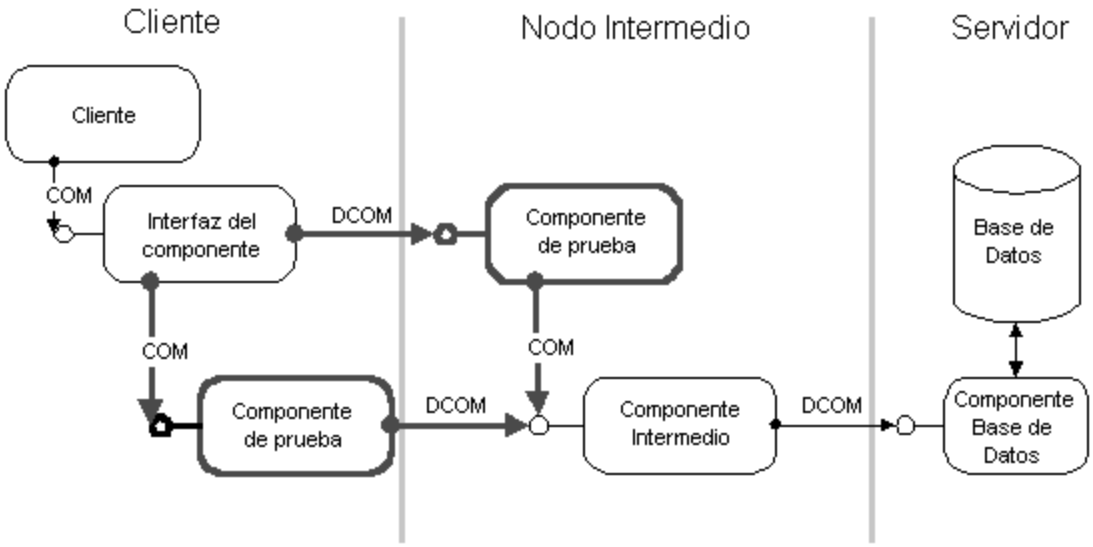


Figura 10: Independencia de la localización. Fuente Propia

· Desarrollo flexible.

En ocasiones no es posible atender a una demanda creciente ni siquiera aumentando la velocidad que proporciona una máquina multiprocesador. La independencia de la localización de DCOM facilita la labor de distribuir componentes entre diversas máquinas, ofreciendo un modo más sencillo y barato de aumentar la capacidad de las aplicaciones. La redistribución es más sencilla cuando los componentes son "sin estado" o no comparten su estado con otros componentes, ya que en este caso es posible ejecutar varias copias en diferentes máquinas. La demanda puede ser distribuida de forma uniforme entre todas las máquinas o de acuerdo a criterios como capacidad y carga. Cambiar la forma en la que los clientes se conectan a los componentes y éstos últimos entre sí es fácil con DCOM, porque permite, sin necesidad de añadir código o recompilar, la

redistribución dinámica de los componentes. Sólo es necesario actualizar el registro, archivo del sistema, o base de datos en la cual se almacena la situación de cada componente en un momento determinado.

Muchas aplicaciones reales tienen uno o más componentes críticos que son utilizados en la mayor parte de las operaciones. Pueden ser, por ejemplo, componentes de bases de datos a los que hay que acceder en serie de modo FCFS (first come-first served). No es posible duplicar este tipo de componentes, ya que su finalidad es precisamente establecer un punto único de sincronización para todos los usuarios de la aplicación. Con el objetivo de mejorar el rendimiento de una aplicación distribuida estos componentes críticos deberían ejecutarse en un servidor dedicado de gran potencia y altas prestaciones.

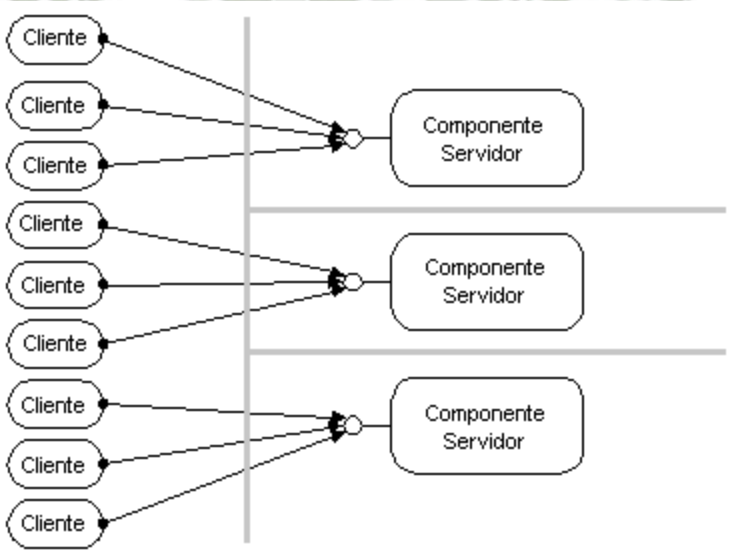


Figura 11: Distribución paralela de componentes. Fuente Propia

Los componentes críticos suelen formar parte de una cadena de procesos, por ejemplo, órdenes de compra o venta en un sistema de venta electrónico: los pedidos deben ser procesados conforme se van recibiendo y en ese orden (*FCFS*). Una posible solución sería dividir el trabajo en componentes más pequeños y ejecutar cada uno de ellos en una máquina

diferente. El efecto sería similar al que se consigue con la técnica del *pipelining* en los microprocesadores modernos: cuando llega la primera petición es procesada por el primer componente y en caso de que sea válida pasa al siguiente. Tan pronto como el primer componente mande al segundo la petición, quedará libre para procesar nuevas entradas y de esta forma tendremos dos máquinas trabajando en paralelo sobre dos peticiones diferentes, a la vez que garantizamos que éstas son procesadas en el orden adecuado. Con DCOM es posible aplicar esta misma técnica cuando trabajamos en una sóla máquina, ya que podemos ejecutar varios componentes en diferentes hebras o en diferentes procesos, facilitando además el crecimiento de nuestra aplicación mediante la distribución de las hebras en una máquina multiproceso o de los procesos en diferentes máquinas.

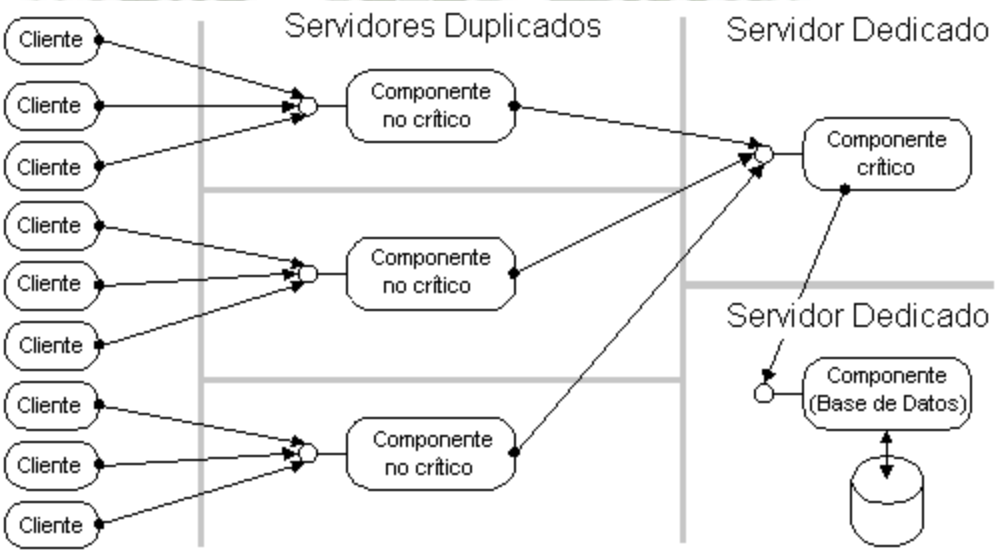


Figura 12: Distribución de un componente crítico. Fuente Propia

2.4.5. Control de la Conexión

Las conexiones a través de la red son más delicadas que las conexiones dentro de una máquina, por esta razón es necesario que los componentes de una aplicación distribuida sean avisados cuando un cliente deja de estar activo o cuando se produce un fallo en la red o en el hardware. DCOM

controla las conexiones entre componentes que están dedicados a un sólo cliente así como las de aquellos que están compartidos por varios clientes. Cuando un cliente establece una conexión con un cliente el contador de referencias se incrementa en una unidad, decrementándose cuando el componente libera al conexión. En caso de que el contador sea cero el componente puede liberar la memoria que ocupa.

DCOM utiliza un protocolo de *ping* en el cual el cliente envía de forma periódica un mensaje para comprobar que los componentes permanecen activos. Si transcurren más de tres periodos sin que el componente reciba estos mensajes DCOM considera rota la conexión y procederá a decrementar el contador de referencias, liberando el componente si el contador llega a cero. Desde el punto de vista del componente, el proceso es el mismo independientemente de que sea el cliente el que se desconecte por sí mismo o de que se produzca un fallo en la red o en la máquina cliente. Todo este mecanismo de liberación de memoria por parte de componentes no utilizados es transparente a las aplicaciones clientes.

2.4.6. Ancho de Banda y Retardo

Aunque en teoría DCOM oculta el hecho de que en las aplicaciones distribuidas los componentes se ejecutan en diferentes máquinas, en la práctica es necesario considerar las limitaciones de una conexión a través de una red:

- Ancho de banda: el tamaño de los parámetros en una llamada a una función afecta directamente al tiempo necesario para completar esa llamada.
- Retardo: La distancia física y el número de elementos de la red, tales como routers o enlaces, introducen un retardo que puede llegar a ser significativo. En el caso de una red global como Internet los retardos pueden ser del orden de segundos.

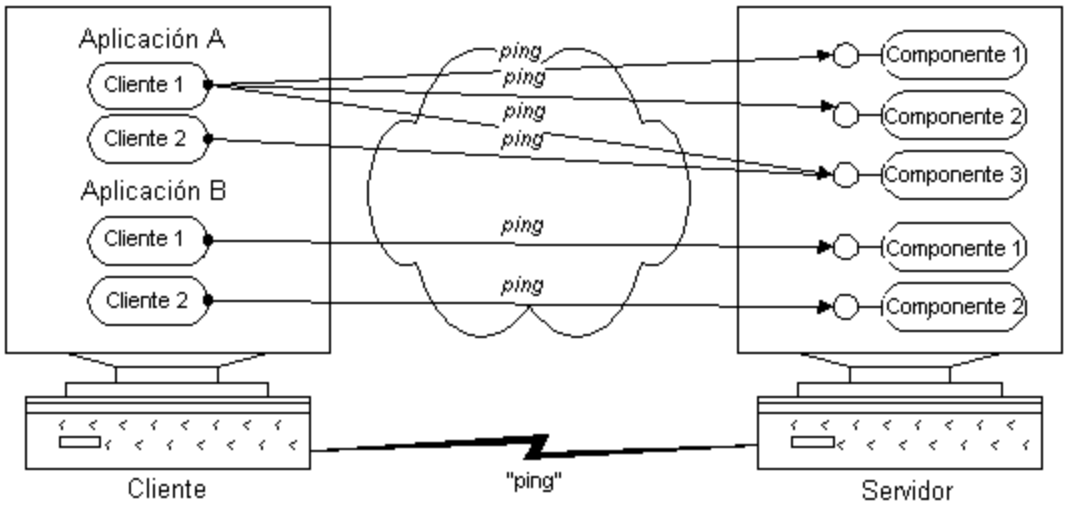


Figura 13: Uso de ping para controlar las conexiones. Fuente Internet

DCOM consigue superar estas limitaciones minimizando cuando sea posible el número de saltos en la red para evitar retardos. El protocolo de transporte más usado por DCOM es el protocolo no orientado a conexión *UDP*, lo que permite una mayor optimización mezclando paquetes de asentimiento con datos y mensajes *ping*. Incluso en el caso de que DCOM emplee protocolos orientados a conexión se consigue una significativa mejora con respecto a protocolos desarrollados específicamente para cada aplicación.

2.4.7. Seguridad

Al utilizar una red de comunicaciones para distribuir una aplicación no sólo nos encontramos con limitaciones físicas, ancho de banda y retardo, sino también con problemas de seguridad entre clientes y componentes. Al ser las funciones accesibles físicamente por cualquiera con acceso a la red, es necesario establecer restricciones en un nivel superior. DCOM implementa su propio sistema de seguridad con el fin de evitar que cada aplicación se vea obligada a desarrollar el suyo de manera independiente.

Una plataforma distribuida como DCOM debe facilitar un entorno seguro que permita distinguir entre clientes y grupos de clientes de modo que el sistema o la aplicación pueda conocer quién intenta acceder a determinados servicios de un componente. DCOM utiliza el sistema de seguridad de Windows NT que consiste en una serie de *security providers* basados en métodos tradicionales tales como dominios o claves públicas en sistemas no centralizados. Una de las partes fundamentales de este sistema de seguridad es el directorio de usuarios que almacena la información necesaria para comprobar las credenciales de éstos.

DCOM proporciona seguridad a las aplicaciones distribuidas sin necesidad de incluir en el cliente o en el componente elementos de codificación o diseño específicos. Al igual que ocurría con la localización de los componentes DCOM también oculta sus necesidades de seguridad. El mismo código ejecutable en un entorno centralizado, con una sola máquina, donde la seguridad no plantea ningún problema puede ser utilizado en un entorno distribuido de manera segura. El diseñador simplemente tiene que fijar los requisitos de seguridad de cada componente, indicando que usuarios tienen derecho de acceso a sus servicios.

2.4.8. Rendimiento y Prestaciones

Como hemos visto hasta ahora la arquitectura DCOM ofrece herramientas que proporcionan flexibilidad tanto para la integración de componentes COM ya desarrollados, como para la utilización de diferentes lenguajes de programación, además de permitir el crecimiento de las aplicaciones. Sin embargo, podemos preguntarnos cómo afectan estas características al rendimiento de la arquitectura. En COM/DCOM el cliente nunca tiene acceso a la implementación del objeto servidor, lo que se consigue, en la mayor parte de los casos, sin tener que utilizar componentes intermedios. Esta transparencia de acceso se obtiene mediante un mecanismo de

comunicación entre componentes a través de llamadas a función mediante punteros. La única sobrecarga que introduce este método frente a los utilizados en otros lenguajes de programación es la búsqueda de la dirección de función en las tablas virtuales (vtables).

Tanto si las llamadas a función se realizan entre procesos situados en la misma máquina o a través de la red, DCOM utiliza su propio mecanismo de *RPCs* orientado a objetos. Microsoft publica una serie de cifras que miden el rendimiento objetivo de este sistema: indican una mejora de aproximadamente el 35% sobre *TCP/IP* para *RPCs* vacías, aunque para *RPCs* reales este porcentaje bajan hasta el 15%.

2.5. COM+

2.5.1. Definición

COM+ es la etapa final de la evolución COM, proporcionan una sólida plataforma de desarrollo. Esta plataforma se compone de diversas tecnologías esenciales que proporcionan los componentes básicos para generar aplicaciones empresariales de varios niveles [20]. Facilita el desarrollo de aplicaciones corporativas distribuidas utilizando COM.

COM+ ofrece las ventajas de las aplicaciones de varios niveles y reduce su complejidad inherente. Cualquier aplicación de varios niveles que pueda parecer irrelevante requiere elementos como un soporte para las transacciones, seguridad integrada, un servidor Web, un sistema de mensajería y una entrega de notificaciones de eventos.

Para entender mejor COM+, hay que conocer acerca de Windows Distributed interNet Applications (Windows DNA). Windows DNA es un framework que habilita a los desarrolladores para construir aplicaciones

[20] Rofail Ash y Shohoud Yasser, *Mastering COM and COM+*, Sybex, USA, 2000

distribuidas usando tecnologías y servicios que son parte del sistema operativo Windows.

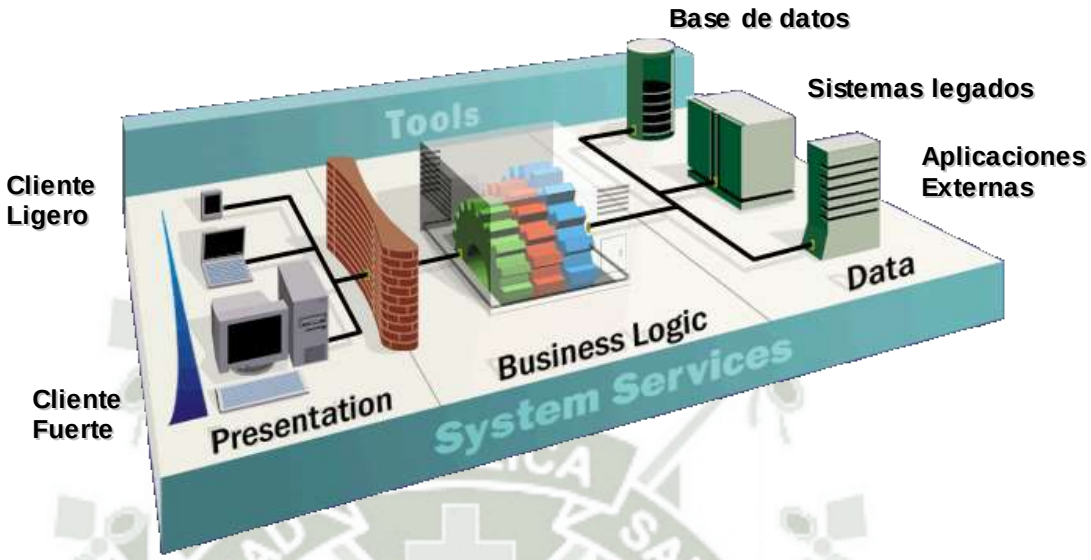


Figura 14 : Windows DNA. Fuente Internet

2.5.2. Arquitectura COM+

La arquitectura del software abarca un conjunto de decisiones significativas sobre la organización de un sistema de software[21].

- Selección de los elementos estructurales, y sus interfaces, que componen el sistema.
- Comportamiento de esos elementos en conjunto, según especificaciones.
- Composición de esos elementos estructurales y su comportamiento en grandes subsistemas.
- Estilo de arquitectura que guía esta organización.

La esencia de la arquitectura del software es la partición del sistema en elementos estructurales.

[21] Booch Grady, Jacobson Ivan y Rumbaugh Jim, *The Unified Software Development Process*, Addison Wesley, España, 1999

Arquitectura de tres capas

Todas las aplicaciones para empresas realizan esencialmente las mismas cosas: *almacenan*, *procesan* y *muestran*.

Para maximizar la reutilización y hacer que sea lo más fácil posible implantar la aplicación, debe hacer que los elementos software de su arquitectura sean lo más independientes posibles. Esto se puede conseguir empaquetando funcionalidades relacionadas en un mismo elemento. La forma más fácil de hacerlo es particionar la aplicación en las mencionadas tres capas: el almacenamiento de información o *capa de acceso de datos*, el proceso de información o *capa de negocio*, y la visualización de información o *capa de presentación*. A esto se le llama arquitectura de tres capas.

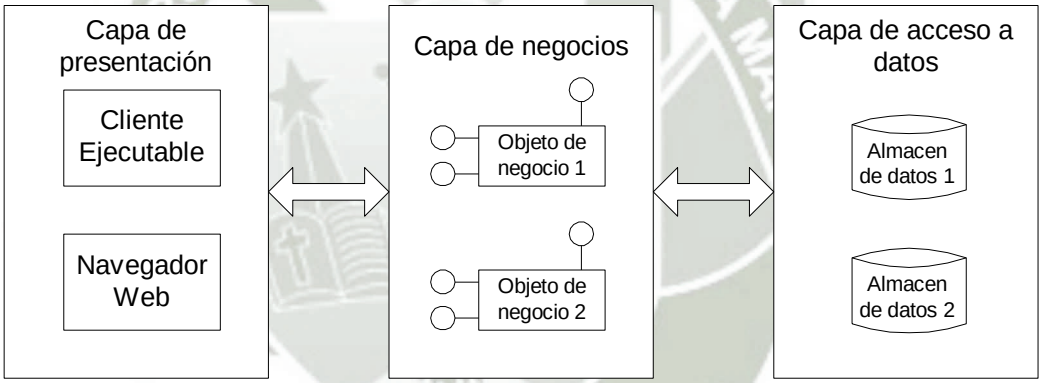


Figura 15 : Arquitectura de tres capas. Fuente Propia

2.5.3. Servicios COM+

Además de proporcionar un entorno en tiempo de ejecución, COM+ incluye varios servicios integrados de gran importancia para los programadores que generan aplicaciones de varios niveles. Puede que algunas aplicaciones de varios niveles necesiten aplicar uno o dos de estos servicios. Otros, sin embargo, necesitarán la utilización de todos ellos. Resulta conveniente tener una perspectiva global de la interrelación entre

estos componentes para poder tomar las decisiones adecuadas durante la fase inicial de diseño. Entre los servicios más importantes de la plataforma para las aplicaciones distribuidas son las siguientes:

1. Servicios de Internet Information Server
2. Servicio de Microsoft Message Queue
3. Servicio de eventos COM+

2.5.4. Transacciones

Una transacción es una serie de operaciones que tiene las siguientes características (ACID):

- *Atomicidad*: Todas las operaciones deben terminarse correctamente o todas deben fallar.
- *Consistencia*: La misma información que inicia una transacción debe ser la que termine.
- *Aislamiento*: Otra transacción, no podrá ver un estado intermedio de la actual transacción.
- *Durabilidad*: Después que se ha consignado una transacción, las modificaciones que se haya realizado no se deben perder.

Muchos sistemas de Gestión de Bases de Datos (DBMS) tienen incluidas primitivas de Inicio, Terminación y vuelta al estado inicial (*Begin, Commit, Rolback*) para manejo de transacciones.

La operación *Begin* inicia la transacción. Cuando sus operaciones se han completado, se llama a la función *Commit* para terminar la transacción. Si esta falla en cualquier momento, puede volver atrás con la función *Rollback*, que deshace cualquier cosa que se haya hecho desde la llamada a *Begin*.

3. ANÁLISIS DE ANTECEDENTES INVESTIGATIVOS

El presente trabajo de investigación tiene antecedentes en trabajos aplicados sobre la construcción de componentes en sistemas distribuidos que ya existen en la actualidad. Uno de los mejores lugares donde se encuentra estos trabajos es en www.researchindex.com de donde se han extraído los más relacionados a nuestro trabajo.

Un primer trabajo de investigación encontrado es: "Lenguaje de Especificación de Calidad de Servicio de Componentes Distribuidos", Autor: Jorge A. Torres Jiménez (jtorresj@itesm.mx), de la Universidad Politécnica de Madrid.

El Resumen del trabajo de Jorge Torres indica, que, en el mundo globalizado y de competitividad da lugar a modelos de negocio emergentes cada vez más complejos. Esto supone un entorno de cooperación entre redes empresariales que dan lugar a redes de actividades empresariales distribuidas, que al integrarse producen una fuerza productiva. Estos modelos de negocios se encuentran inmersos en una dinámica de cambio, que obligan a que los procesos de negocio se ajusten de forma dinámica y den lugar a nuevos procesos de negocio.

Para apoyar este dinamismo, los Servicios Web es una alternativa que permite la integración y colaboración de las aplicaciones a través de la red, que facilita la construcción de sistemas middleware. En este momento, la definición de estándares es vital para lograr la integración de aplicaciones y dada la importancia notable que esto tiene, se ha abierto un tema de investigación y desarrollo relevante.

Los objetivos de trabajo giraron en torno a, dar una visión de los elementos que están en el entorno de los Servicios Web, entender las arquitecturas básicas y avanzadas, para implementar aplicaciones basadas en Servicios Web y de forma particular, caracterizar la calidad de un Servicio Web que facilite la definición de lenguajes para especificar la calidad de los Servicios Web

Otro trabajo de investigación relacionado es: “Un Modelo de Componentes para el Desarrollo de Aplicaciones Distribuidas”, Autor: Antonio Vallecillo Moreno, de la Universidad de Málaga.

El trabajo propone un modelo de componentes específicamente diseñado para la construcción modular de aplicaciones abiertos, separando los diferentes aspectos que componen los requisitos específicos de las aplicaciones (propiedades), y en su adición modular e independiente a los componentes que forman parte de ellas (mediante el uso de "controladores reutilizables"). Así es posible simplificar tanto a los componentes como a los propios sistemas, pues los primeros sólo han de encapsular computación, mientras que los segundos se encargan de la creación y la comunicación entre los componentes.

Sus principales contribuciones fueron:

1. definir un modelo general de componentes para sistemas abiertos con capacidades reflexivas, que define los conceptos de componentes y de controladores reutilizables, y ofrece una serie de mecanismos específicos para el desarrollo de aplicaciones en este tipo de sistemas;
2. implementar un prototipo suyo en Java para probar su viabilidad y mostrar su expresividad;
3. identificar algunas de las propiedades de especial importancia en los sistemas abiertos, y que pueden ser implementadas mediante controladores reutilizables;
4. construir un marco formal de trabajo en Object-Z y lógica temporal que permite especificar los controladores y razonar sobre su comportamiento cuando se añaden a los componentes;
5. definir una metodología para el desarrollo de aplicaciones distribuidas basadas en componentes y en controladores;
6. definir un lenguaje de descripción de componentes y controladores para configurarlos y construir aplicaciones con ellos;
7. definir los mecanismos necesarios para permitir la integración de componentes heterogéneos de otros modelos (CORBA, DCOM, JavaBeans) en aplicaciones construidas con su modelo; y

4. OBJETIVOS

4.1. Usar un modelo de componentes con programación paralela para aplicaciones distribuidas.

4.2. Optimizar las operaciones de manejo de información en aplicaciones distribuidas.

5. HIPÓTESIS

Dado que el manejo de información en aplicaciones distribuidas y la reutilización de código en programación orientada a objetos se realizan de una forma limitada, es probable que mediante el uso de un modelo de componentes en programación paralela se optimice el manejo de información y operaciones en procesamiento distribuidas.

PLANTEAMIENTO OPERACIONAL

1. TECNICAS, INSTRUMENTOS Y MATERIALES DE VERIFICACION

1.1.Técnicas

La técnica a utilizarse será la Observación primaria, planificada, individual y de laboratorio, así como la manipulación.

1.2. Instrumentos

El instrumento que se utilizará será el programa de evaluación que interactué con el uso de componentes con hilos, a través de la ficha de observación.

1.3. Materiales de verificación

Los materiales serán la computadora y todos sus periféricos así como suministros.

Cuadro de Estructura del Instrumento

Variable	Indicadores	Items
Uso de Modelo de componentes en Programación Paralela	Granularidad Speedup.	Facilidad de realización de tareas en la programación. Respuesta de acceso al programa
Aplicaciones Distribuidas	Demora Latencia	Acceso a los datos locales y remotos Facilidad de acceso

2 CAMPO DE VERIFICACIÓN

2.1.Ubicación Espacial

La Delimitación espacial se centra en el laboratorio de Ing. de Sistemas de la Universidad Católica de Santa María de la Región de Arequipa.

2.2.Ubicación Temporal

Se trata de una investigación de índole Coyuntural por ser en el 2003.

2.3.Unidades de Estudio

Las unidades de estudio son la programación paralela y aplicaciones distribuidas. La investigación experimental no usa población ni muestra, sino estudio de casos.

3 ESTRATEGIA DE RECOLECCIÓN DE DATOS

El ámbito para la recolección de datos es de tipo bibliográfico y de internet, de los cuales se obtendrá información inicial y teórica, a continuación será de laboratorio donde se elegirá la herramienta y plataforma para desarrollar en aplicaciones distribuidas.

El recurso humano estará formado por el autor del trabajo de investigación y del personal practicante de laboratorio, el recurso económico, material y logístico instrumentos, etc. correrán por cuenta del autor, pudiendo usar los equipos que cuenta la Facultad de Ingeniería.

De otro lado el nombre de la recogida de la información será COMPROPARDIS que servirá de fuente de datos, y que significa Componente en Programación Paralela Distribuida.

CRONOGRAMA

Tiempo / Tarea	2do semestre 2002				1er semestre 2003				2do semestre 2003			
	1	2	3	4	1	2	3	4	1	2	3	4
Recolección de Información	x	x	x	x								
Análisis			x	x	x							
Construcción de programas de evaluación					x	x	x	x				
Estructuración de Resultados							x	x	x			
Informe Final									x	x	x	x

ANEXO 2: METODOLOGIA

La metodología utilizada fue la de COMET[22] (Concurrent Object Modeling and architectural design mEThod) con la notación de UML (Unified Modeling Language).

La elección de la metodología COMET se basa en que está orientada a las aplicaciones concurrentes, en particular, aplicaciones distribuidas.

La construcción de las aplicaciones se ha enfatizando en el desarrollo de prototipos[23]

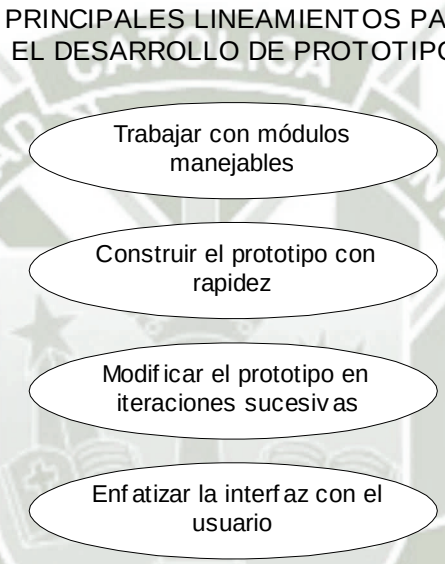


Figura 16: Lineamientos principales para el desarrollo de prototipos. Fuente[15]

a) Trabajar con módulos manipulables.

Los módulos nos permite relacionarnos con otras características, a parte que su construcción es independiente de otros módulos del sistema. Aquellas características que se consideren poco importantes quedarán fuera del desarrollo del software.

[22] Hassan Gomaa, Designing Concurrent, Distributed, and Real Time Application with UML, Addison-Wesley, USA, 2000, pag.110
[23] Kendall y Kendall, Analisis y Diseño de Sistemas, Prentice Hall, Mexico

Los componentes son ideales para dichos fines dado que presenta los siguientes beneficios:

- Componentes COM son fácilmente Reemplazables:
    - Sin COM, necesitaríamos actualizar el programa entero.
    - Con COM, no necesitamos actualizar la aplicación completa. Simplemente reconstruir un componente único que así lo requiera.
- Componentes COM son ideales para los cambios de requerimientos de negocios:
    - Los requerimientos de negocios de software son usualmente fluidos. Durante el desarrollo de software, nuevos requerimientos y cambios constantemente aparecen.
    - Ya que los componentes COM son fácilmente reemplazables, podemos localizar las reglas de negocio dentro de pocos componentes y actualizarlos.

b) Construir el software con rapidez

El Desarrollo del Software se realizará con rapidez en su fondo esencial, lo que nos permitirá contar con una excelente visión sobre la manera en que debería desarrollarse el resto de la implantación. Y para esto las herramientas de desarrollo rápido que se ha utilizado en la construcción del cliente ha sido Microsoft Visual Basic 6.0© y HTML.

c) Modificar el software en interacciones sucesivas

El Software permitirá modificarse en repetidas ocasiones por medio de varias iteraciones, tales cambios acercarán más a los objetivos a alcanzar, para cada modificación se requerirá de nuevas evaluaciones.

La comprobación del software de aplicación es un proceso continuo que se da en cada etapa de su desarrollo. Ahora bien, las pruebas que un software debe cumplir satisfactoriamente para ser considerado como tal son:

- Prueba Técnica
- Prueba de Optimización de Acceso información.

La prueba técnica verifica la no existencia de errores lógicos ni de programación en el software. La prueba de Optimización de tiempo verifica efectividad el obtener un acceso eficiente a la información.

Las pruebas y correcciones se repetirán muchas veces. Por ello, aunque se mencionen aquí las correcciones realizadas como si hubiesen ocurrido una detrás de la otra, lo cierto es que serán el resultado de muchas pruebas, evaluaciones y opiniones de usuarios.

Las reformulaciones y correcciones de orden técnico que se harán serán las siguientes:

- Interfaz: rediseñar la interfaz poca intuitiva.
- Se permitirá agregar objetos y/o componentes para el proceso de información.
- Se añadirá al final diversas opciones.
- Se corregirán muchas veces las operaciones para el manejo de información.

d) Enfatizar la Interfaz con el Usuario

Para que los usuarios sean capaces de interactuar con el mínimo de adiestramiento, y sin complicaciones; para lo cual se diseñaron una interfaz donde el usuario cuente con el máximo control sobre las funciones presentadas. Interfaz que se detallará más adelante.

La selección del sistema operativo + hardware (o plataforma) sobre la que se evalúen las aplicaciones de prueba, se hizo teniendo en cuenta los siguientes criterios:

- Difusión: (la plataforma seleccionada debe ser ampliamente conocida y utilizada por los usuarios).
- Potencia y flexibilidad (facilidades propias de la plataforma, para el desarrollo del software diseñado).

Windows 2000/XP cumple con todos los criterios mencionados ya que es un Sistema Operativo de 32 bits, Multitarea real: control asíncrono, múltiples hebras de ejecución, Velocidad alta, Memoria plana direccionable directamente, API mejorada, que incluye multimedia, El S.O. gestiona a todos los dispositivos de E/S, Variedad de herramientas de desarrollo.

Ya definida la plataforma de trabajo, se procedió a seleccionar las herramientas de software más apropiadas La herramienta de programación a usar será los que componen el conjunto Visual Studio 6.0 ©. especialmente Visual C++, esto debido a la robustez y facilidad que posee este lenguaje para el manejo de componentes, así mismo incorpora características tanto de alto como de bajo nivel, es bastante estructurado, y optimiza bastante el código haciéndolo más pequeño, compacto y veloz que cualquier otro lenguaje. Se optó por el producto Visual C++, porque incorporaba un práctico asistente de generación de código (Class Wizard) e incluía las Clases Básicas de Microsoft (Microsoft Fundation Classes), un potente conjunto de bibliotecas de objetos reutilizables.

Arquitectura del Modelo realizado

a) Aplicación simple

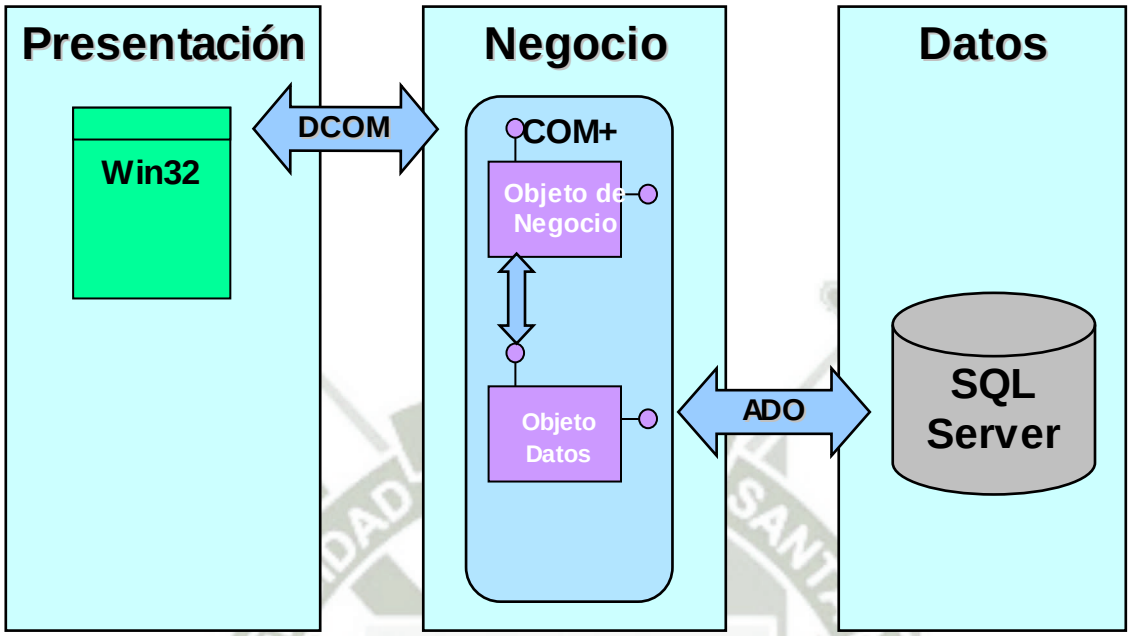


Figura 17: Arquitectura para una aplicación Simple. Fuente Propia

b) Aplicación Web

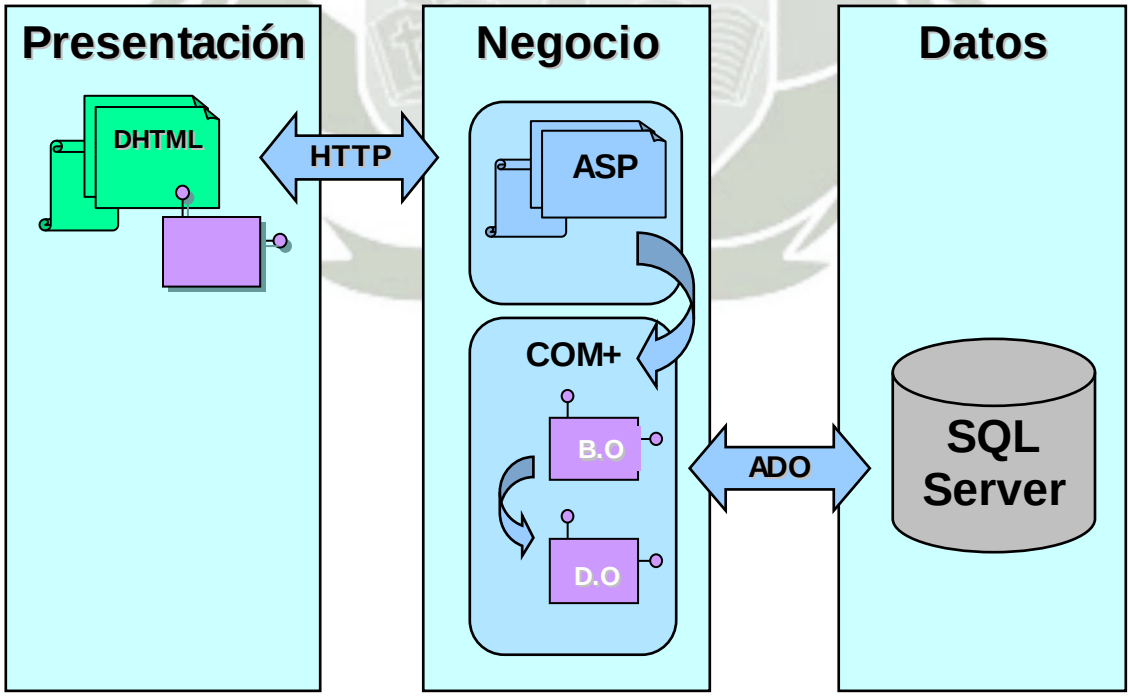


Figura 18: Arquitectura para una aplicación Web. Fuente Propia

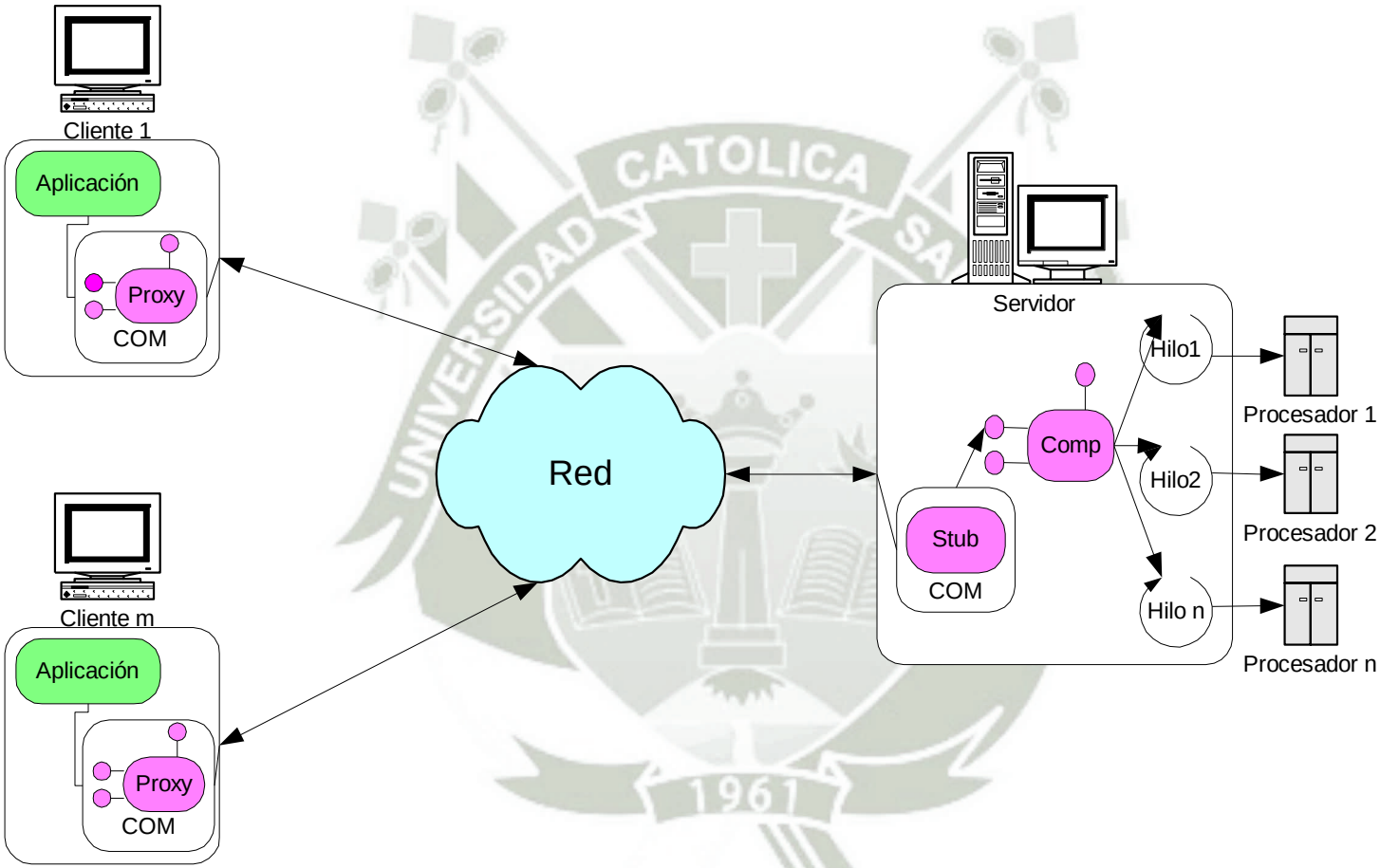


Figura 19: Modelo de Componentes en programación paralela para aplicaciones distribuidas. Fuente Propia

Una vez creado el Servidor de Componentes se procede a registrarlo en el sistema

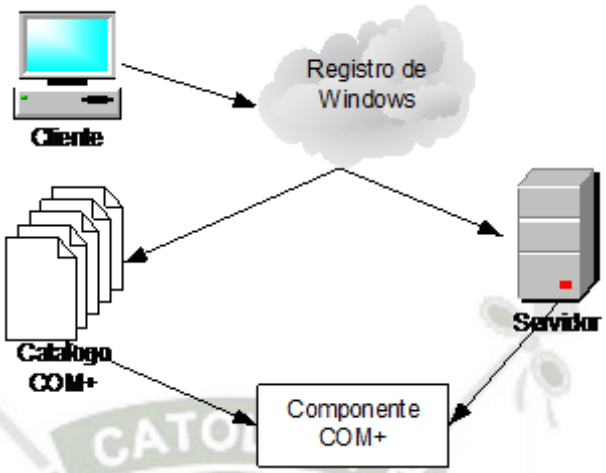


Figura 20: Pasos en la creación de un objeto COM+. Fuente Propia

El catalogo COM+ vendrá a ser una base de datos con información relativa a todos los componentes COM+ instalados en el sistema. Cuando un cliente precisa los servicios de un cierto componente, éste es creado no tan solo a partir del servidor, sino como resultado de sumar a éste los atributos almacenados en el mencionado catalogo COM+.

Finalmente las aplicaciones de negocios se comunicarán a través de las 3 hileras lógicas:

- Cliente,
- La lógica de negocio (Servidor de componentes) y
- El almacenamiento de los datos.

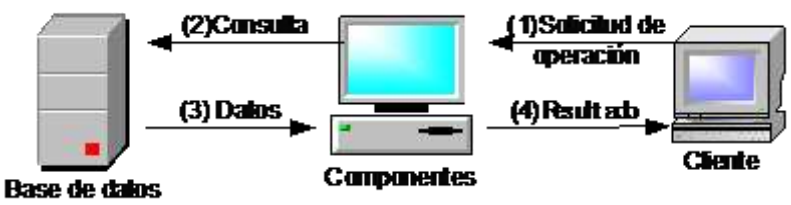


Figura 21: Comunicación en la arquitectura. Fuente Propia

ANEXO 3: INSTRUMENTO

FICHA DE OBSERVACIÓN

PRUEBA Nro.: .....

FECHA: .....

LUGAR: .....

PLATAFORMA: .....

TIEMPO DE EJECUCION DE PROCESOS: .....

SPEEDUP: .....

ANEXO 4: PRUEBAS DE EVALUACION

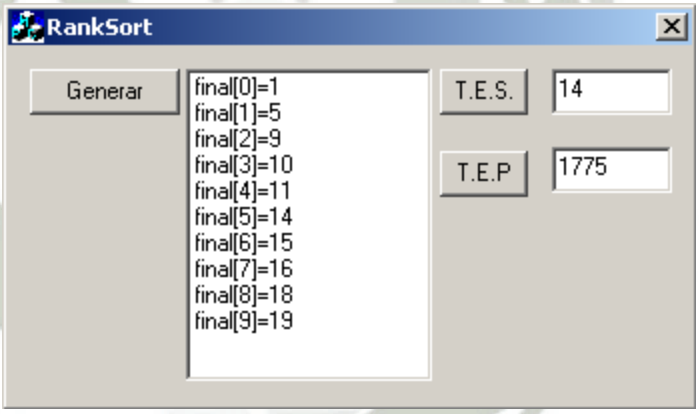
Interfaz de las pruebas de evaluación

Las pruebas se de evaluación se realizaron en los laboratorios del Programa Profesional de Ingeniería de Sistemas de la Universidad Católica de Santa María, estos laboratorios cuentan con pcs Pentium III 166 Mhz, con 64 de RAM.

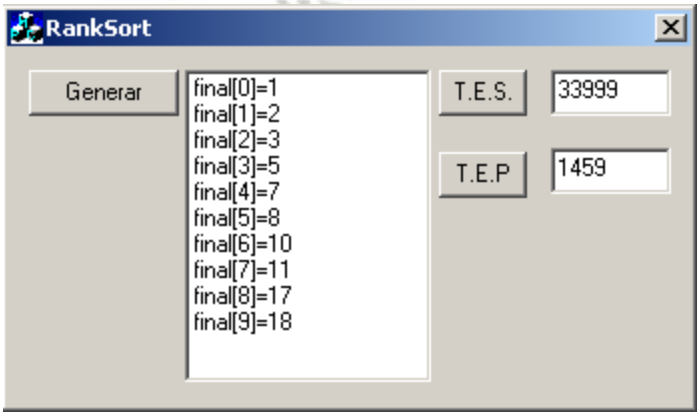
Es muy probable que los resultados varien en cuanto a máquinas distintas, por la velocidad de procesador, RAM, etc.

a) En cuando al uso de componentes en programación paralela:

- **Granularidad baja**



- **Granularidad alta**

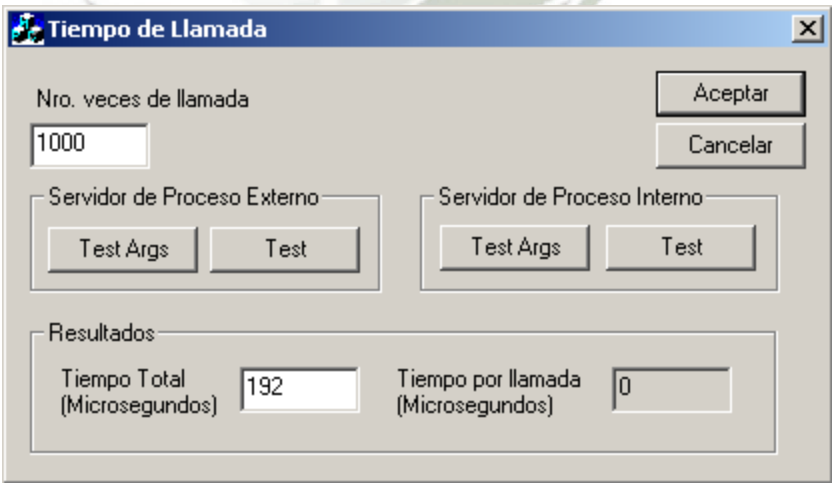


Granularidad						
Baja				Alta		
Prueba	Con Hilos	Sin Hilos		Prueba	Con Hilos	Sin Hilos
1	1775	14		1	1459	33999
2	1402	14		2	1421	35848
3	1426	15		3	1344	38965
4	1312	14		4	1214	33409
5	1191	14		5	1280	32675
6	1408	14		6	1444	35611
7	1426	13		7	1477	31352
8	1342	13		8	1497	32429
9	1316	14		9	1332	31890
10	1197	13		10	1218	37287
Promedio	1379	14			1368	34346

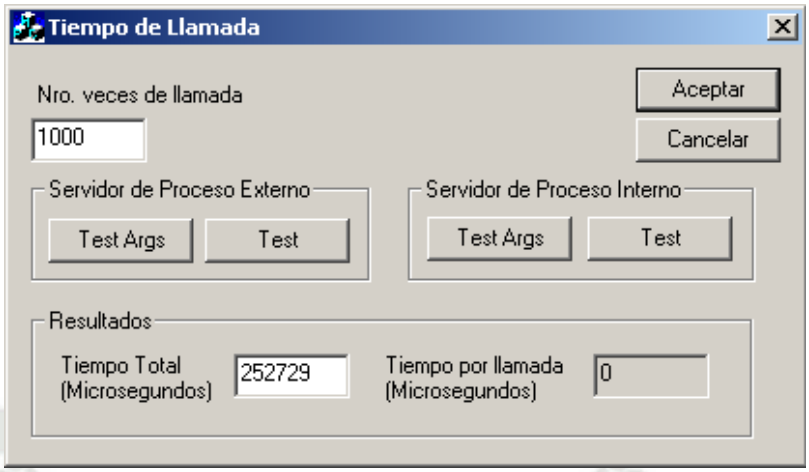
- **Speedup:** Es el Tiempo de ejecución Secuencial (sin hilos) entre el Tiempo de ejecución Paralela (con hilos).

Granularidad	Speedup
Baja	0.0101
Alta	25.15

- **Servidor de Proceso Interno con demora de llamada a argumentos.**



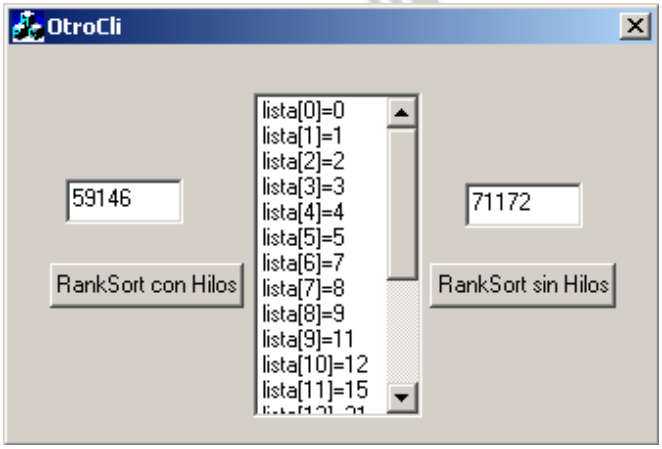
- **Servidor de Proceso Externo con demora de llamada a argumentos.**



Servidor IN Process				Servidor OUT Process		
	Argumento				Argumento	
Prueba	Con	Sin		Prueba	Con	Sin
1	192	191		1	252729	131962
2	196	190		2	220549	134934
3	192	190		3	219729	128781
4	192	190		4	221916	130689
5	191	185		5	220786	125574
6	195	190		6	222480	131379
7	193	191		7	217397	131887
8	192	186		8	221552	131680
9	191	189		9	220742	130963
10	192	187		10	218564	130427
Promedio	192	188			223644	130827

b) En cuando a aplicaciones distribuidas.

- **Demora Cliente - Servidor Local**



Prueba	Con Hilos	Sin Hilos
1	59146	71172
2	68559	69271
3	60810	73478
4	62459	76845
5	52708	66849
6	53728	73147
7	67555	73302
8	51608	73463
9	56713	74032
10	54549	70707
	58783	72226

- **Latencia Cliente - Servidor Remota**

Prueba	Con Hilos	Sin Hilos
1	110614	130914
2	101885	128913
3	99682	130749
4	98707	132379
5	99307	126300
6	103227	136119
7	122461	132210
8	101505	122636
9	114651	135205
10	112726	126206
Promedio	106476	130163

Aplicación de uso distribuida

a) Aplicación Simple

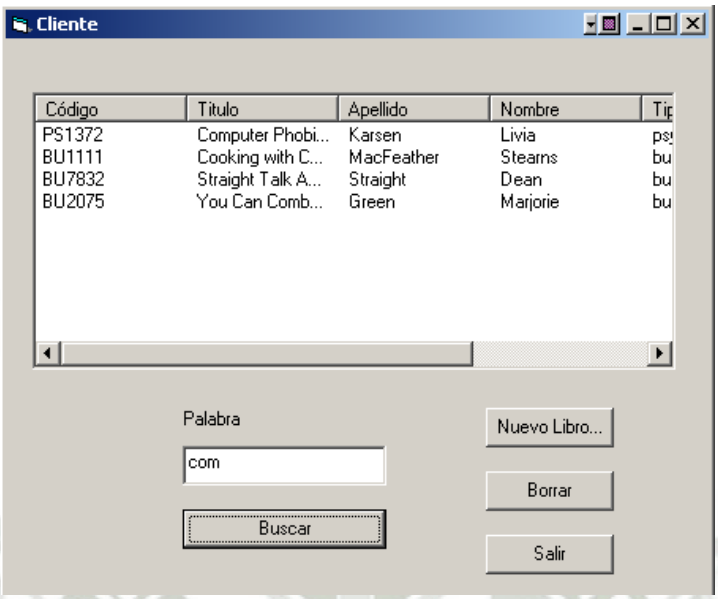


Figura 22: Ventana Principal de consulta con la Biblioteca. Fuente Propia

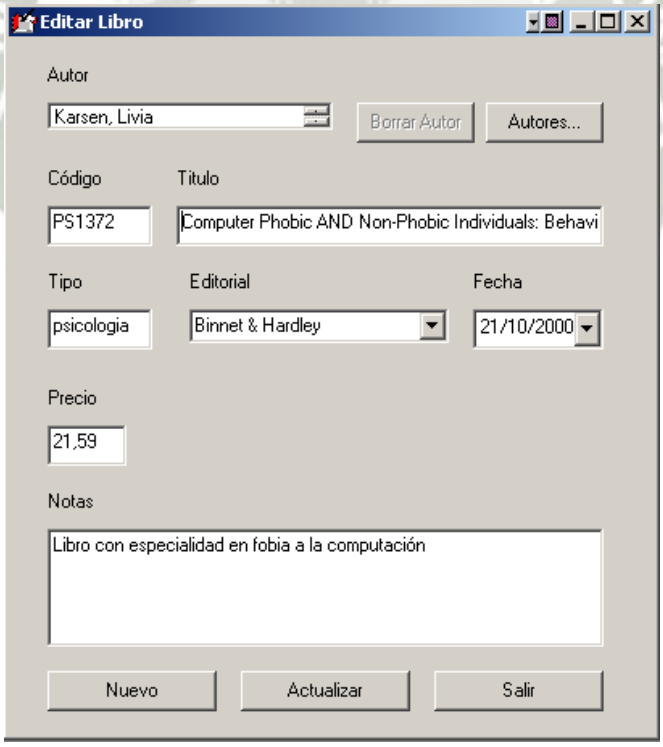


Figura 23: Ventana de Edición de Libros. Fuente Propia

b) Aplicación Web

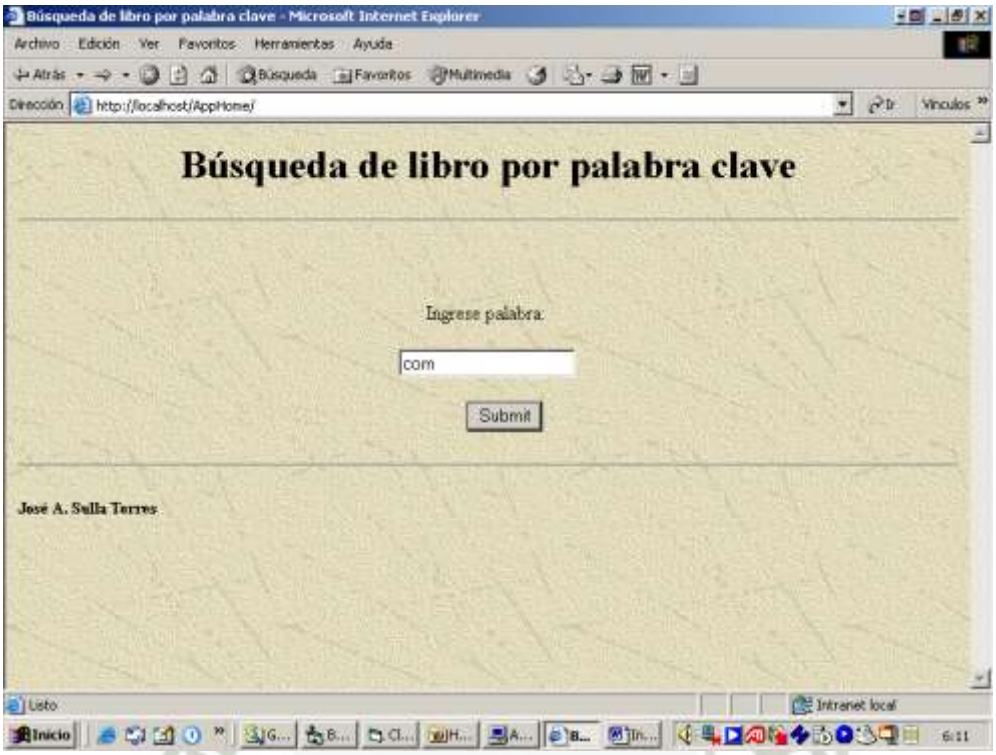


Figura 24: Ventana Principal de consulta Web con la Biblioteca. Fuente Propia

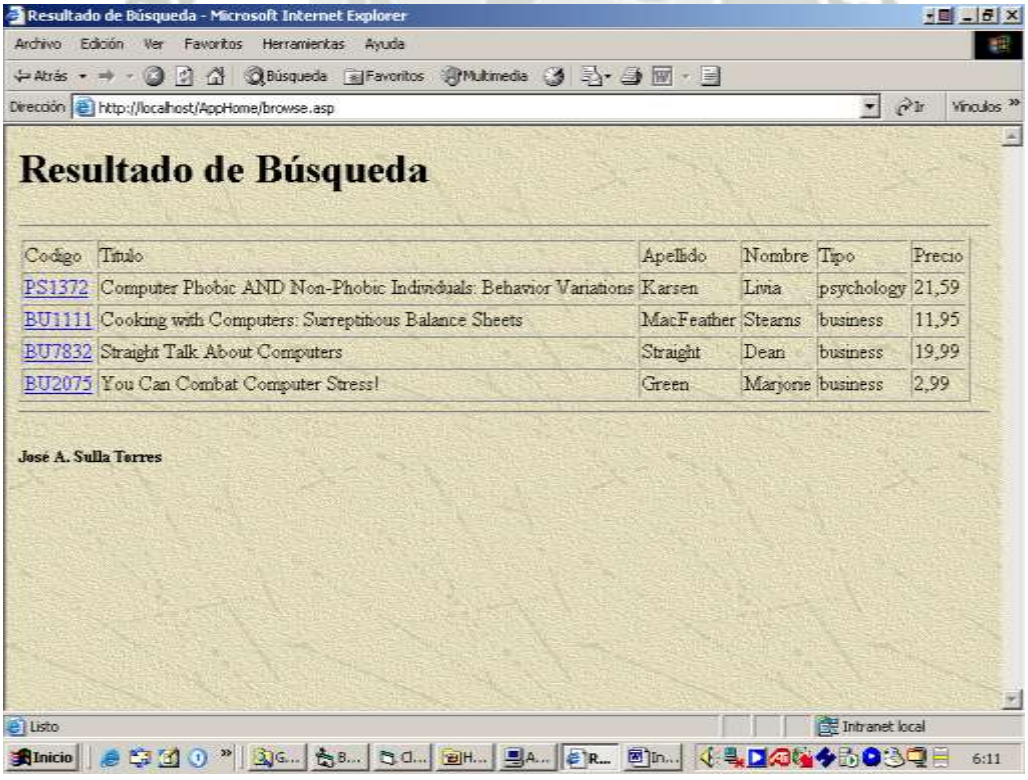


Figura 25: Ventana de Resultado de la búsqueda Web. Fuente Propia

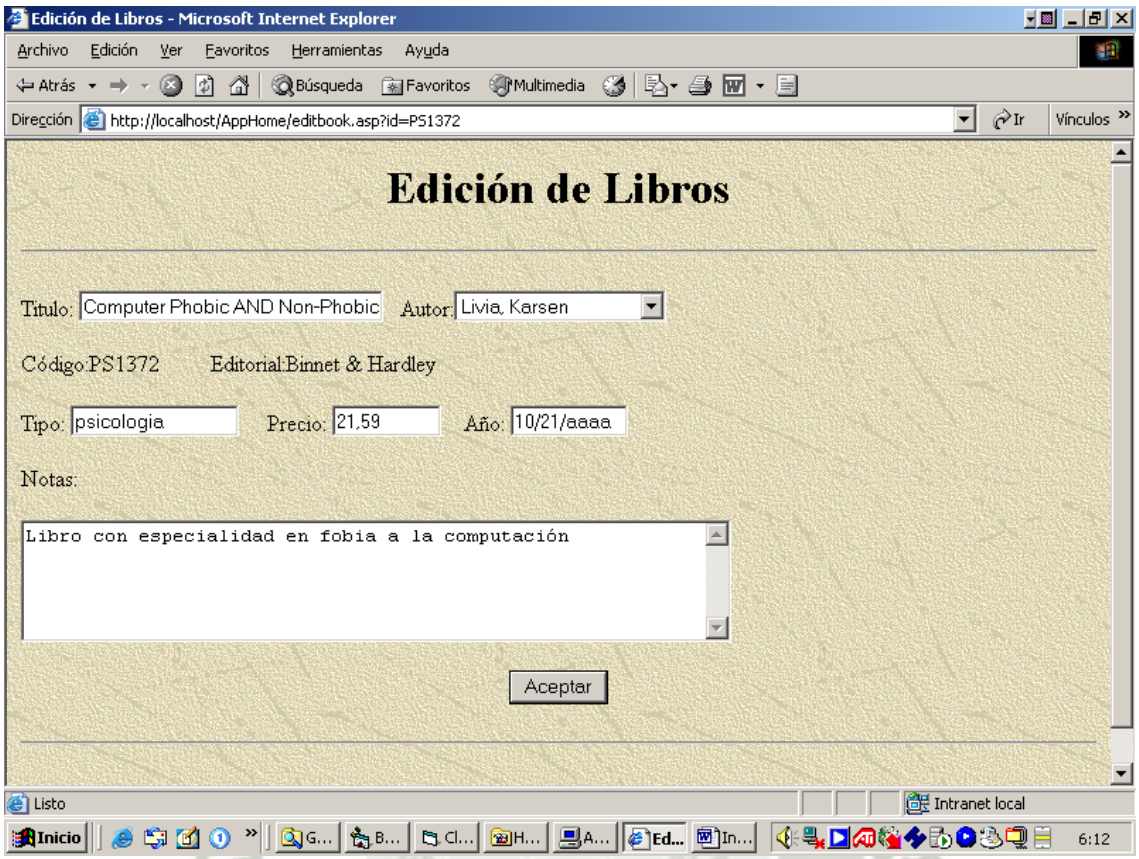


Figura 26: Ventana Web de Edición de Libros. Fuente Propia

ANEXO 5: ENCUESTA PREVIA PARA DETERMINAR EL PROBLEMA

ENCUESTA

Sr. Desarrollador, la presente encuesta tiene por objetivo determinar los beneficios del uso de componentes con programación paralela. Agradecemos su colaboración.

1. Sus aplicaciones permiten interactuar con otras aplicaciones distintas a las que ha desarrollado Ud.
   a) Si
   b) No
   c) Algunas Veces
2. Sabe si sus métodos(funciones, subprogramas) pueden ser utilizados por otros lenguajes de programación
   a) Si
   b) No
   c) Algunas veces
3. Los servicios que brinda su aplicación están preparados para ser accedidos desde cualquier punto de una red
   a) Si
   b) No.
   c) Algunas veces
4. Qué modelo de componentes conoce y usa en las situaciones que así lo requieran
   a) COM
   b) CORBA
   c) Java Beans
   d) Otra:……..
   e) Ninguna
5. Para Ud. la optimización en el tiempo de ejecución de sus aplicaciones es de vital importancia
   a) Si.
   b) No.
6. Conoce y utiliza los hilos (threads) dado el caso que así lo requiera?
   a) Si
   b) No
7. Cuan a menudo utiliza procesamiento distribuido?
   a) Siempre.
   b) Alguna veces.
   c) Nunca.
8. Qué tan importante es la velocidad de respuesta en la transacciones en Redes (Internet, Intranet, Extranet) para su trabajo?
   a) Muy Importante.
   b) Mas o menos importante.
   c) No es importante.
9. Sus aplicaciones desarrolladas esta preparadas para maximizar el uso de una arquitectura multiprocesadores?
   a) Si.
   b) No.
10. Conoce algunos formas para medir la eficiencia en el tempo de ejecución de una aplicación.
    a) Si, mencionelos……..
    b) No

Resultados de las encuestas

Estas encuestas fueron realizadas a distintos desarrolladores de software en la ciudad de Arequipa.

1. A la pregunta: "Sus aplicaciones permiten interactuar con otras aplicaciones distintas a las que ha desarrollado Ud", respondieron que Si el 73%, No el 15 % y A veces el 12%, lo que indica que la mayoría si sabe que sus aplicaciones interactúa con otras aplicaciones.
2. A la pregunta: "Sabe si sus métodos(funciones, subprogramas) pueden ser utilizados por otros lenguajes de programación", respondieron que Si el 67%, No el 24% y A veces el 9%, lo que indica que la mayoría sabe que sus métodos puede ser utilizados por otros lenguajes de programación.
3. A la pregunta: "Los servicios que brinda su aplicación están preparados para ser accedidos desde cualquier punto de una red", respondieron que Si el 65%, No el 24% y A veces el 11%, lo que indica que la mayoría sabe que los servicios de su aplicación están preparados para trabajar en red.
4. A la pregunta: "Qué modelo de componentes conoce y usa en las situaciones que así lo requieran", respondieron que COM el 69%, CORBA el 2%, JavaBeans el 9%, Ninguna 17%, lo que indica que la mayoría conoce el modelo de componentes COM y en un buen porcentaje (17%) no conoce ningún modelo de componentes.
5. A la pregunta: "Para Ud. la optimización en el tiempo de ejecución de sus aplicaciones es de vital importancia", respondieron que Si el 95%, No el 5%, lo que indica que para la mayoría es de vital importancia el tiempo de ejecución.
6. A la pregunta: "Conoce y utiliza los hilos (threads) dado el caso que así lo requiera?", respondieron que Si el 47%, No el 53%, lo que indica que la mayoría no conoce ni utiliza los threads.
7. A la pregunta: "Cuan a menudo utiliza procesamiento distribuido", respondieron que Siempre el 11%, Algunas veces el 61% y Nunca el 28%, lo que indica que la mayoría algunas veces utiliza el procesamiento distribuido.
8. A la pregunta: "Qué tan importante es la velocidad de respuesta en la transacciones en Redes (Internet, Intranet, Extranet) para su trabajo?", respondieron que Muy importante el 64%, Mas o menos importante el 32% y No es importante el 4%, lo que indica que para la mayoría es muy importante la velocidad de respuesta en una operación distribuida.
9. A la pregunta: "Sus aplicaciones desarrolladas esta preparadas para maximizar el uso de una arquitectura multiprocesadores?", respondieron que Si el 17%, No el 83%, lo que indica que la mayoría de aplicaciones desarrolladas no están preparadas para maximizar el uso de la programación paralela.
10. A la pregunta: "Conoce algunos formas para medir la eficiencia en el tiempo de ejecución de una aplicación", respondieron que Si el 39%, No el 61%, lo que indica que la mayoría no conoce como medir la eficiencia en el tiempo de ejecución de una aplicación..

ANEXO 6: CREACIÓN DE HILOS

Cuando se crea un proceso, se genera automáticamente un hilo primario. A su vez, este hilo, en C++, puede invocar a la función *CreateThread* cuando sea necesario crear otros hilos adicionales para ejecutarse dentro del espacio de direcciones del proceso que invoca a dicha función. La función retorna un descriptor al nuevo hilo. Su sintaxis es:

```
HANDLE CreateThread(
LPSECURITY_ATTRIBUTES lpThreadAttributes, // atributos de seguridad
DWORD dwStackSize, // Tamaño inicial de la pila para este thread
LPTHREAD_START_ROUTINE lpStartAddress, // puntero a la función del thread
LPVOID lpParameter, // argumento para el nuevo thread
DWORD dwCreationFlags, // atributos de creación
LPDWORD lpThreadId ); // puntero para recibir el ID del thread
```

suponiendo que deseamos esperar hasta que la hebra termine su ejecución, el código adecuado sería:

```
::WaitForSingleObject(hThread, INFINITE);
```

el segundo parámetro determina el número de milisegundos que se debe esperar. La constante INFINITE instruye al Sistema Operativo a esperar hasta que la hebra concluya sin importar el tiempo que eso signifique.

En relación a la prioridad de un thread podemos decir que: existe una prioridad por defecto, pero nosotros podemos modificarla de la siguiente forma:

```
::SetThreadPriority(hThread, THREAD_PRIORITY_ABOVE_NORMAL);
```

Si deseamos suspender temporalmente una hebra activa podemos utilizar la función **SuspendThread** de la siguiente forma:

```
::SuspendThread(hThread);
```

así mismo, si deseamos continuar la ejecución de una hebra suspendida lo podemos hacer de la siguiente forma:

```
::ResumeThread(hThread);
```

Uso de threads con el algoritmo ranksort

Tomando como ejemplo el algoritmo de ordenamiento *Rank Sort*, considere el problema de ordenar una lista de *n* números. El *rango* de cada número en la lista esta definido como el número total de elementos de la lista que son menores que dicho número. Para calcular el *rango* de algún número *x* en la lista, se compara *x* con cada elemento de la lista y se guarda el resultado de los números menores que *x*. En una lista totalmente ordenada el *rango* de cada elemento es la posición *actual* en la lista (ascendente).

LISTA	RANGO
15	4
10	3
39	7
8	2
22	6
4	0
19	5
6	1

→

FINAL	
[0]	4
[1]	6
[2]	8
[3]	10
[4]	15
[5]	19
[6]	22
[7]	39

Este algoritmo puede ser optimizado, ya que el rango de cada elemento de la lista puede ser calculado independientemente por diferentes threads. El thread 1 puede calcular RANGO[1] tomando de LISTA[1] y compararlo con cada elemento de la lista. Mientras este cálculo se esta llevando a cabo en el thread 1, el thread 2 puede simultáneamente estar calculando RANGO[2] de LISTA[2] con cada elemento de la lista. Si hay *n* elementos en la Lista y *n* threads, entonces a cada thread *i* se le puede asignar para calcular RANGO[*i*] usando elementos de LISTA[*i*].

Una forma adecuada de crear threads para un arreglo de *n* elementos utilizando **CreateThread** sería:

```
HANDLE Hilo[n];
DWORD IdHilo[n];
…
    for( ind = 0; ind < n; ind++)
        Hilo[ind]=::CreateThread(NULL,0,Ubicar,(LPVOID)ind,0,&IdHilo[ind]);
    WaitForMultipleObjects(n, Hilo, TRUE, INFINITE);
```

Donde *n* representa el número de hilos.

Considerando la función ***Ubicar*** para conocer el rango de un elemento de la lista sería:

```
DWORD WINAPI Ubicar(LPVOID  num)
{
        int  src = (int)  num;
        int  testval,  j, rank;
        testval  = lista[src];
        j = src;                        //j se moverá  a través  de todo el arreglo
        rank = 0;
        do
        {
                j = (j+1)%n;
                if(testval  >= lista[j])
                        rank++;
        }while(j  != src);
        final[rank-1]  = testval;  //coloca  en la posición  ordenada
        return  0;
}
```

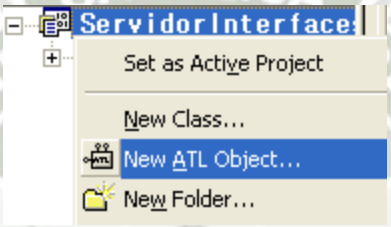
ANEXO 7: CREACIÓN DE COMPONENTES

Para crear componentes se hace uso del Asistente de componentes para facilitarnos la creación de los mismos:

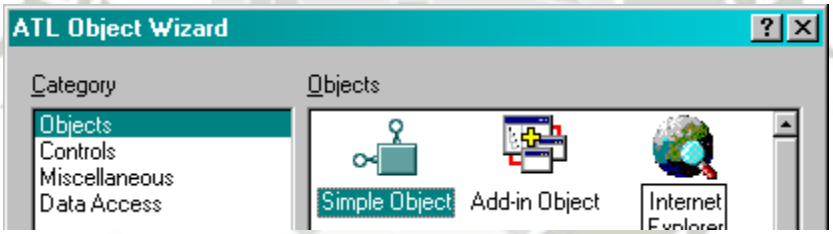
1. Ejecute Visual C++ y seleccione **New** en el menú **File**.
2. Seleccione la ficha **Projects** del cuadro de dialogo **New**.
3. Seleccione **ATL COM AppWizard** en la lista de tipos de proyecto.
4. Teclee: **ServidorInterfaces**, en el cuadro **Project Name**.
5. Hacer clic en el Botón OK (aparece la **ATL/COM AppWizard**)
6. Seleccioinar **Dynamic Link Library** Bajo el tipo del Servidor **(Server Type).**

El proyecto no contendrá clases, pero tendrá CComModule y todos los puntos de entrada de COM requeridos para implementar un servidor COM de proceso Interno.

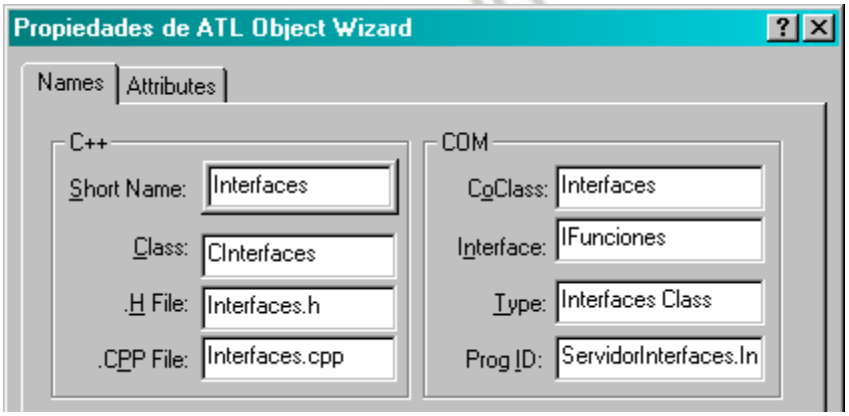
7. Presione botón derecho sobre el Servidor y elija **New ATL object…**



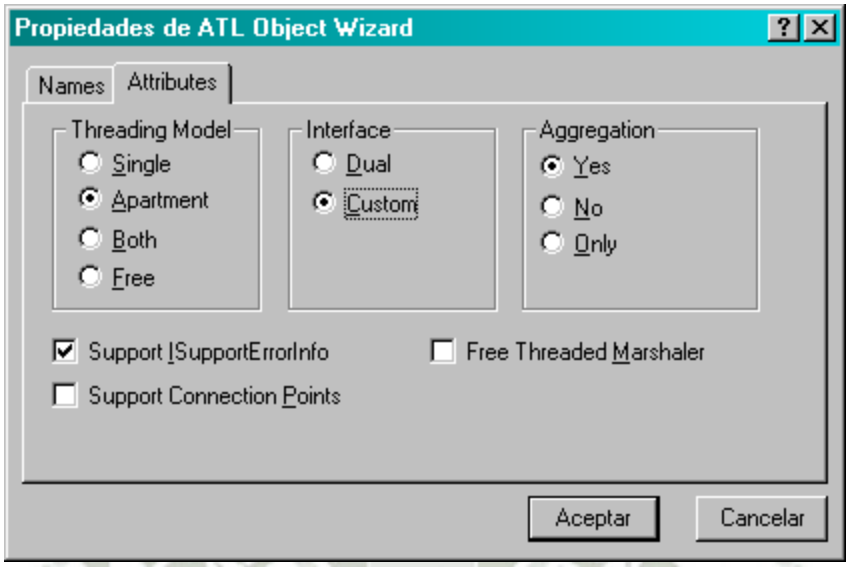
8. Añadir el Objeto COM, seleccionando **Simple Object** y luego presionamos **Next>**



9. Introduzca en el campo *Short Name*: el nombre **Interfaces** lo que generará los otros nombre respectivos., cambie en el campo *Interface*: con el nombre **IFunciones**. Presione luego el botón **Aceptar**



10. Elija la pestaña **Attributes** y en el marco Interface elija la opción **Custom** y active la opción check **Support ISupportErrorInfo** para que genere el código respectivo que soporte los errores. Luego presione el botón **Aceptar.**



El ATL Object Wizard Genera los Archivos Interfaces.cpp y Interfaces.h

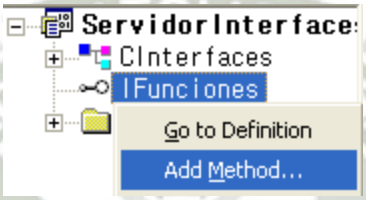
Interfaces.cpp
CComCoClass implementa el objeto de clase para su clase COM; Implementa la interfaz IClassFactory.
La clase CInterfaces también hereda de las interfaces IFunciones e ISupportErrorInfo

ServidorInterfaces.IDL.
Define las Interfaces del los objetos utilizando el lenguaje IDL(Interface Definition Lenguaje).

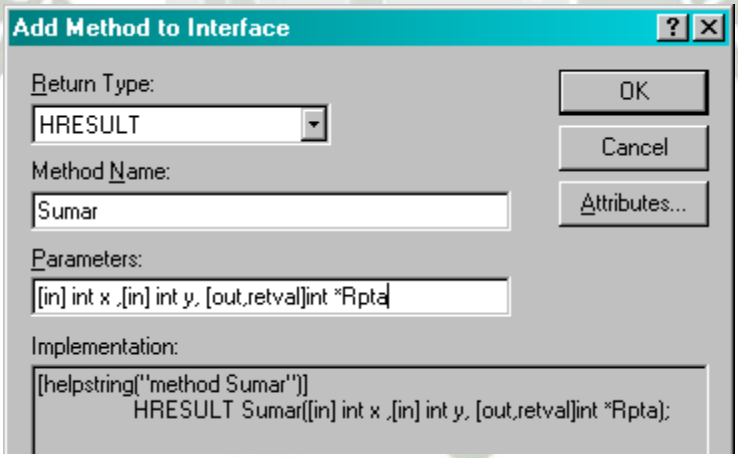
```
import "oaidl.idl";
import "ocidl.idl";
    [
        object,
        uuid(D9528696-9BAF-4A37-B8B2-C0BF39BE528A),
        helpstring("IFunciones Interface"),
        pointer_default(unique)
    ]
    interface IFunciones : IUnknown
    {
    };

[
    uuid(58EE6B61-2326-4A61-917E-67711973EA61),
    version(1.0),
    helpstring("ServidorInterfaces 1.0 Type Library")
]
```

```
library SERVIDORINTERFACESLib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");
    [
        uuid(8B727BAC-64DB-45B3-BE53-37BCDBAE5176),
        helpstring("Interfaces Class")
    ]
    coclass Interfaces
    {
        [default] interface IFunciones;
    };
};
```

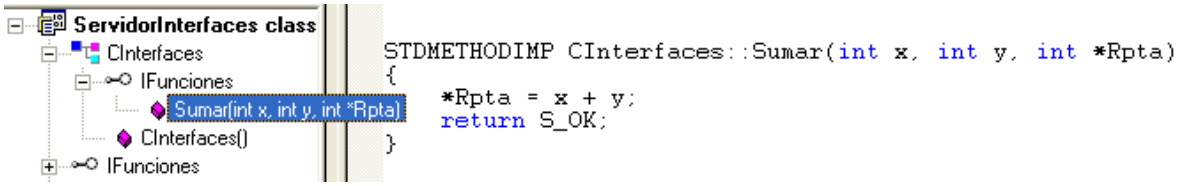
11. Presione botón derecho sobre la interface IFunciones y elija la opción Add Method… para agregar los Metodos Sumar.



12. Añada lo siguiente:
    Method Name : **Sumar**
    Parameter : **[in] int x, [in] int y, [out,retval] int *Rpta**



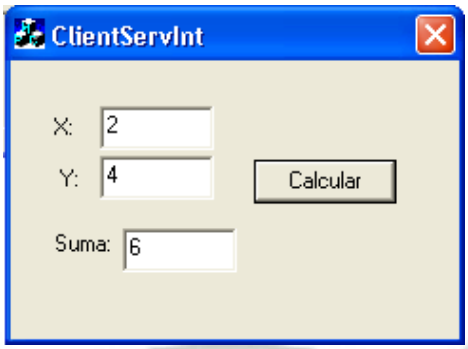
13. Implementar el método Sumar, para ello presione doble clic en el método Sumar de la interfaz IFunciones en la clase CInterfaces:



14. Construya el servidor (F7) esto permitirá registrarlo en el sistema de tal forma que pueda ser usado por distintos clientes.

CONSTRUCCION DE UN CLIENTE CON VISUAL C++

1. Cree un Proyecto basado en dialogo empleando el MFC AppWizard (exe), denominado DlgCliente.
2. Diseñe el siguiente cuadro de dialogo como se muestra a continuación:



3. Añada Código para soporte COM. En la función InitInstance del DlgClienteApp.cpp

```
BOOL CDlgClienteApp::InitInstance()
{
    AfxEnableControlContainer();
    if(!AfxOleInit())
        AfxMessageBox("No se pudo Inicializar COM");
    //…
}
```

4. Importe la Librería Tipo para su Servidor en el Archivo stdafx.h

```
#import "..\ServidorInterfaces\ServidorInterfaces.tlb"
```

5. Compile el proyecto y verificar la creación de los archivos SevidorInterfaces.tlh y SevidorInterfaces.tll (Clases envolventes que facilitan el uso de COM) en la carpeta Debug del Proyecto Cliente.
6. Añada una instancia de un puntero inteligente a la clase de diálogo.

```
class CDlgClienteDlg : public CDialog
{
// Construction
private:
    SERVIDORINTERFACESLib::IFuncionesPtr m_IFunciones;
```

7. Instancie un Objeto COM. Haga esto en la función OnInitDialog de la clase de la vista

```
BOOL CClientServIntDlg::OnInitDialog()
{
    //…
   // TODO: Add extra initialization here
```

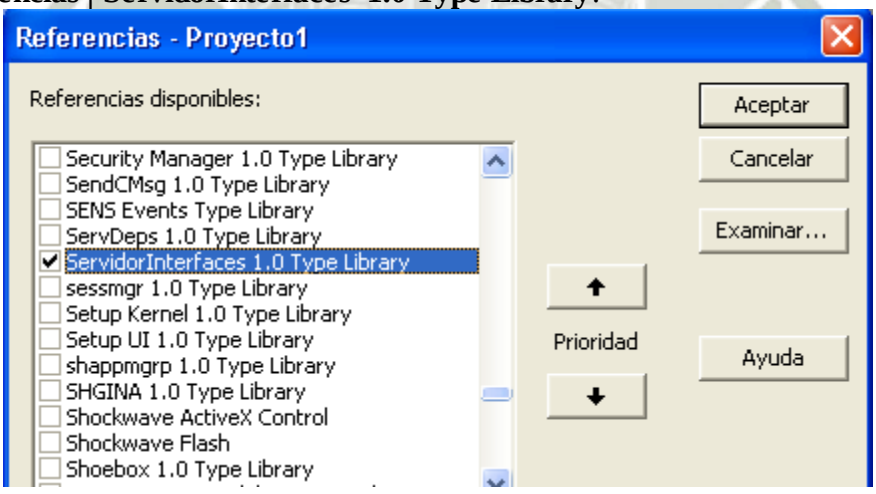
```
    HRESULT
hr=m_IFunciones.CreateInstance("ServidorInterfaces.Interfaces");
    if(FAILED(hr))
    {
        _com_error err(hr);
        AfxMessageBox(err.ErrorMessage());
    }
    return TRUE;  // return TRUE  unless you set the focus to a control
}
```

8. Implemente la Interfaz de Usuario.

```
void CDlgClienteDlg::OnCalcular()
{
  CString Var1,Var2;
  GetDlgItemText(IDC_X,Var1);
  GetDlgItemText(IDC_Y,Var2);
  Var2.Format("%d",m_IFunciones->Sumar(atoi(Var1),atoi(Var2)));
  SetDlgItemText(IDC_SUM,Var2);
}
```

CONSTRUCCION DE UN CLIENTE CON VISUAL BASIC

1. Abra Visual Basic luego elija un Nuevo Proyecto **EXE Estándar** y diseñe el formulario como se muestra en el punto 2 de la **Construcción de un cliente con Visual C++.**
2. Haga una referencia a la librería del Servidor a través del menú: **Proyecto | Referencias | ServidorInterfaces 1.0 Type Library**.



3. Ahora agregue el siguiente código:

```
Dim funcion As Interfaces

Private Sub Command1_Click()
   Text3.Text = funcion.Sumar(Text1.Text, Text2.Text)
End Sub
```

```
Private Sub Form_Load()
   Set funcion = New Interfaces
End Sub

Private Sub Form_Terminate()
   Set funcion = Nothing
End Sub
```

4. Ejecute su aplicación y verifique la interacción entre el Cliente de Visual Basic y el servidor COM implementado con Visual C++.