# Un administrador de procesamiento distribuido utilizando Windows Sockets

*Ing. R. Chaer[1], Ing. A. Fonseca[1], Lic. M. Naiouf[2], Ing. A.De Giusti[3]*

*Laboratorio de Investigación y Desarrollo en Informática[4]*
*Instituto de Ingeniería Eléctrica[5]*

## Resumen

Se presenta un administrador de procesamiento distribuido utilizando Windows Sockets, que permite la comunicación y distribución de tareas entre máquinas de una red, para la ejecución de algoritmos paralelos.

La plataforma del sistema es una red en la cual la única restricción es que las estaciones estén ejecutando Windows. El soporte de comunicaciones se realiza a través de una librería que permite utilizar canales de comunicación sobre la red, administrando la señalización mediante el envío de mensajes a ventanas.

La importancia del desarrollo consiste en la posibilidad de simular una "máquina paralela" de bajo costo con recursos disponibles, y experimentar distintas soluciones cooperativas.

Finalmente, se discute un ejemplo de aplicación, desarrollado en conjunto en el Instituto de Ingeniería Eléctrica y el LIDI.

[1] Ingeniero. Docente con dedicación en el Instituto de Ingeniería Eléctrica.
E-mail randall@iie.edu.uy
[2] Prof. Adjunto con Ded. Excl. LIDI. Dpto. de Informática, Facultad de Ciencias Exactas, UNLP.
E-mail mnaiouf@ada.info.unlp.edu.ar
[3] Inv. Principal CONICET. Profesor Tit. Ded. Excl., Dpto. de Informática, Facultad de Cs. Exactas, UNLP.
E-mail degiusti@ada.info.unlp.edu.ar
[4] Facultad de Ciencias Exactas. Univ. Nacional de La Plata. Argentina.
Calle 50 y 115 Primer Piso, (1900) La Plata. TE 54-21-227707 E-mail lidi@ada.info.unlp.edu.ar
[5] Instituto de Ingeniería Eléctrica. Universidad de la República. Montevideo. Uruguay.
Julio Herrera y Reissig 565. Montevideo. E-mail randall@iie.edu.uy

## Introducción

Una de las áreas de mayor interés y desarrollo dentro de la Informática en los últimos años es la referida al procesamiento paralelo y distribuido. Gran parte de los problemas reales contienen un paralelismo implícito, que puede ser explotado a través de la distribución de tareas que trabajen cooperativamente en distintos procesadores [Andr91] [Coff92]. Los temas de investigación y desarrollo son variados, e incluyen arquitecturas orientadas al multiprocesamiento [Hwan93], problemas de especificación y verificación de algoritmos [Fort85] [Hoar85], optimización algorítmica, etc [Heer91] [Leig92] [Mors94].

Las áreas de dominio del procesamiento paralelo son numerosas. Entre ellas se encuentran aplicaciones científicas y matemáticas, simulación de sistemas, procesamiento de señales digitales, tratamiento de imágenes, visión por computadora, bases de datos distribuidas, etc [Laws92] [Huss91].

Dentro de las arquitecturas de soporte para el procesamiento paralelo, los costos de las máquinas con gran número de procesadores son, en muchos casos, un obstáculo que impide contar con ellas. Por este motivo, una opción son los sistemas de multiprocesamiento distribuido localmente (LDMS: Locally Distributed Multiprocessing Systems) [Niel90]. Un sistema de tipo LDMS es un conjunto de nodos de procesamiento, como por ejemplo microcomputadoras (eventualmente heterogéneas), interconectadas en un área local y orientadas a soportar la misma función de procesamiento. Estas arquitecturas son de costo accesible, y resultan adecuadas para el estudio de problemas de multiprocesamiento.

Una variante de LDMS es utilizar una LAN (donde bajo un sistema operativo convencional, cada máquina es funcionalmente independiente de las demás) y crear un kernel de sistema operativo distribuido para convertirla en un LDMS. Para esto, se deben desarrollar un conjunto de primitivas de manejo de comunicaciones de la LAN y definir técnicas de scheduling que permitan coordinar las computadoras independientes para que cooperen en la resolución de la función de procesamiento.

Existen distintas alternativas para el desarrollo del administrador. Una de ellas es la utilización de Netbios sobre una red local Ethernet [Tine93]. Otra opción, en el caso de disponer de un sistema Unix, es la Parallel Virtual Machine (PVM). Este trabajo presenta una implementación utilizando Windows Sockets [Micros], que permite la comunicación y distribución de tareas entre máquinas de una red.

## Objetivo

El objetivo del trabajo es presentar el administrador de procesamiento distribuido NetTopos, cuyas facilidades de comunicación y distribución de tareas permiten la ejecución de algoritmos paralelos. La plataforma es una red con estaciones que ejecutan Windows, y el soporte de comunicaciones se realiza a través de una librería que permite utilizar canales de comunicación sobre la red, administrando la señalización mediante el envío de mensajes a ventanas.

La importancia del desarrollo está dada en la posibilidad de contar con una "máquina paralela" utilizando recursos disponibles, lo cual permite experimentar diferentes soluciones cooperativas.

## Motivación

El administrador NetTopos se desarrolló en el marco del curso sobre computación distribuida, con el propósito principal de paralelizar el cálculo de los simuladores de sistemas energéticos.

Se desea que el software desarrollado permita distribuir procesos, programados en forma tradicional, sin mezclar al usuario con la necesidad de programación avanzada en ambiente Windows y con los problemas inherentes a la comunicación entre procesos por la red. Con tal fin se desarrolló una interface con Windows y con la red de manera de poder especificar la paralelización de procesos utilizando semáforos e indicando la máquina donde se desea correr un proceso. Con este fin se crea un *demonio*, que estando en una máquina se active frente a mensajes enviados desde otra, y sea capaz de correr un procedimiento ordenado remotamente y conteste los resultados.

Se ha elegido utilizar el soporte de comunicación por red brindado en la librería Windows.dll, que permite utilizar canales de comunicación sobre una red administrando la señalización mediante envío de mensajes a ventanas de Windows. Obviamente se requiere que la red esté ejecutando Windows en cada máquina y que se disponga del soporte para WinSocket.

En el LIDI y en el IIE se encuentran disponibles los programas que implementan la distribución de una función que genera una variable aleatoria a partir de una semilla en la red NetTopos. En este caso, en lugar de un simulador energético se ha usado una función muy sencilla (Xrandom) que está en la librería Rand01.dll.

## Análisis del Administrador

NetTopos significa "red de topos", pues la idea es tener un kernel de sistema operativo distribuido residente en cada nodo, que maneje en forma "subterránea" los servicios de comunicación de paquetes entre nodos, con el protocolo definido en el standard WinSocket. En cada máquina "vive" un topo con su mismo nombre.

Funcionalmente, se tiene en cada nodo tres módulos: *topo.exe, nettopo.dll y topito.exe*, como muestra la Fig. 1:

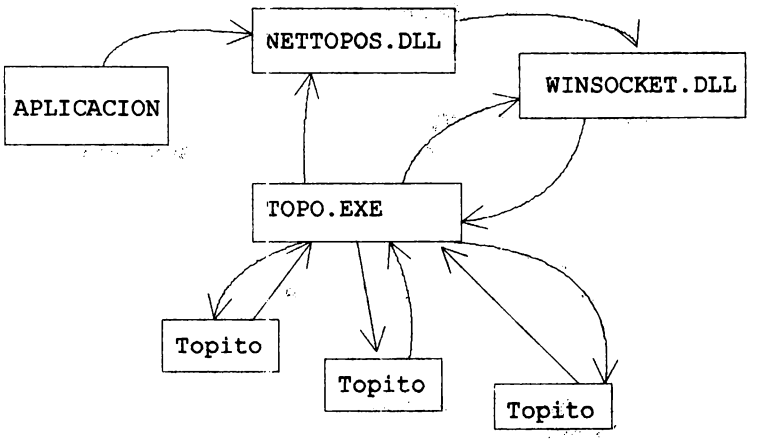


*Fig. 1*

*Topo.exe*, debe ser ejecutado en cada máquina que quiera participar de la red. Permanece residente y su función es crear un canal de comunicación (socket) por el cual la máquina queda escuchando en la red, alerta a mensajes que le puedan venir desde otros nodos.

*Nettopos.dll* es una librería en la cual se brindan todos los servicios que necesita el usuario para construir sus aplicaciones, como por ejemplo el manejo de semáforos de control de procesos, despacho de procesos remotos y recepción de resultados.

*Topito.exe* es un proceso esclavo utilizado por el Topo de una máquina para ejecutar un servicio de una librería solicitado en forma remota. De esta manera, se logra que cada pedido se ejecute en forma independiente de los demás como un proceso Windows.

NetTopos está diseñado para permitir la distribución de la ejecución de funciones de programas convencionales, y se ha intentado evitar que el usuario deba ser un experto en la programación bajo Windows. Por otra parte, se ha intentado diseñar el kernel de modo de evitar grandes cambios en el código de programas ya desarrollados que se quieran ejecutar en forma distribuida utilizando esta herramienta.

El usuario de NetTopos debe poder especificar la paralelización de procesos utilizando semáforos, e indicando la máquina donde se desea correr un proceso.

La administración de los procesos y la comunicación entre máquinas se centralizó en la librería NetTopos.dll. Para facilitar la utilización de la librería desde aplicaciones escritas en Turbo Pascal para Windows, se cuenta con la unidad *utopos.pas*. Para utilizar la librería desde otro lenguaje de Windows se debe crear el módulo de importación de servicios correspondiente.

## Los servicios de NetTopos

### *A- Manejo de semáforos*

***function CrearSMF : integer;***
Crea un nuevo semáforo devolviendo un número de identificación del mismo; dicho número debe utilizarse para referirse al semáforo creado.
Si no hay semáforos disponibles el resultado es -1, indicando que no fue creado.
Si la operación es exitosa, el número identificador del semáforo es mayor o igual a uno.

***function DestruirSMF( hs : integer ) : integer;***
Destruye el semáforo con número identificador *hs*.
Si la operación es exitosa el resultado es 0. Si *hs* ya fue destruido, el resultado es -1.

***function IncSMF( hs : integer ) : integer;***
Incrementa en 1 la cuenta de eventos del semáforo *hs*. El resultado es el valor de la cuenta previo a ser incrementada.

***function DecSMF( hs : integer) : integer;***
Decrementa en 1 el valor de la cuenta de eventos del semáforo *hs*. El resultado es el valor de la cuenta previo a ser decrementado.

***function EsCeroSMF( hs : integer ) : boolean;***
Devuelve True si la cuenta de eventos del semáforo *hs* es cero.

### *B- Manejo de altas y bajas*

***function DarAltaAplicación : integer;***
Debe ser llamado al inicio de cualquier aplicación que pretenda usar NetTopos.
Mediante esta llamada se registra la aplicación en NetTopos, y permite que cuando la misma termine se liberen los recursos asignados a ella.

***function DarBajaAplicación : integer;***
Este servicio da de baja la aplicación liberando los recursos asignados a ella.
Para que los recursos estén asociados a una aplicación es necesario que esta llame a *DarAltaAplicación*.

Para facilitar la programación, y dado que es difícil prever todos los casos de terminación de una aplicación, es aconsejable inscribir el servicio *DarBajaAplicación* en la cadena de procedimientos de Exit de la misma, de manera de asegurar que se la da de baja y se liberan los recursos reservados por esta.

Para los usuarios Pascal se ha incluido en la unidad utopo los procedimientos *AltaAplic* y *BajaAplic*. AltaAplic llama a *DarAltaAplicación* e inscribe BajaAplic en la cadena de procedimientos de finalización de la aplicación. BajaAplic da de baja a la aplicación llamando a DarBajaAplicación y permite que se llame al siguiente de los procedimientos en la cadena de terminación.

***C- Manejo de tareas***

***function DespacharServicio( SMF : integer; Maquina, NombDll, NombFunc : string; hde : Thandle; nde : longint ) : integer;***
Despacha una tarea en *Máquina* y bajo el control del semáforo *SMF*.
Cuando la tarea es despachada se incrementa el semáforo llamando (internamente) a IncSMF(*SMF*).
El llamado a esta función desencadena un proceso por el cual la aplicación está "llamando" al topo de la máquina destino e indicándole que se desea ejecutar la función *Nombfunc* de la librería *Nombdll*, pasando como parámetros de entrada el identificador de un bloque de memoria reservado a través de Windows (*hde*) de longitud *nde* bytes.
El topo intentará cargar desde el directorio especificado en la variable de ambiente *temp* la librería y la función; si tiene éxito contestará que acepta el trabajo y se preparará para recibir los datos con los cuales debe llamar a la función. Registrará la nueva tarea en ejecución en una lista prevista a tal propósito, y devolverá a la aplicación que solicita el servicio un identificador que le permite referirse a la tarea.
El resultado de esta función, si es mayor que 0, es el identificador de la tarea para propósitos de referencias futuras por la aplicación. Si por alguna razón no se logró despachar la tarea el resultado de la función es un código de error. Las posibles causas de error son: que no haya más identificadores de tareas libres en la máquina que solicita el servicio, que falle el intento de comunicarse con el topo de la otra máquina pues éste no responde, o que no queden identificadores de tareas libres en la máquina destino.
En caso de resultado exitoso, el semáforo *SMF* se verá incrementado al retorno de la función.

***function ResultServicio( tarea :integer; var hds : THandle; var nds : longint ) : integer;***
Si la tarea terminó exitosamente, se devuelve en *hds* un identificador de memoria de Windows, en el cual se encuentra el resultado de la tarea, y en *nds* la cantidad de bytes del resultado. El resultado de esta función es cero en este caso, y los recursos asignados internamente por NetTopos a la tarea son liberados.
Es responsabilidad del usuario liberar el bloque de memoria (usando GlobalFree(*hds*)) una vez que no necesite más los resultados.
Para acceder a los resultados el usuario debe bloquear primero el bloque de memoria obteniendo un puntero válido al mismo mediante la función GlobalLock(*hds*) de Windows.

***function DestruirServicio( ht : Integer ) : integer;***
Este servicio le indica a NetTopos que cancele la tarea despachada con identificador *ht*.

En el Anexo 1 se muestra como despachar una función *MinFunc* de la librería *MiLibrería* en la máquina *TopoViejo*.

## Breve análisis del desarrollo del ambiente

## Modelo Objeto

Se describen en esta sección un modelo de clases y un diagrama de instancias.

### *Clases*

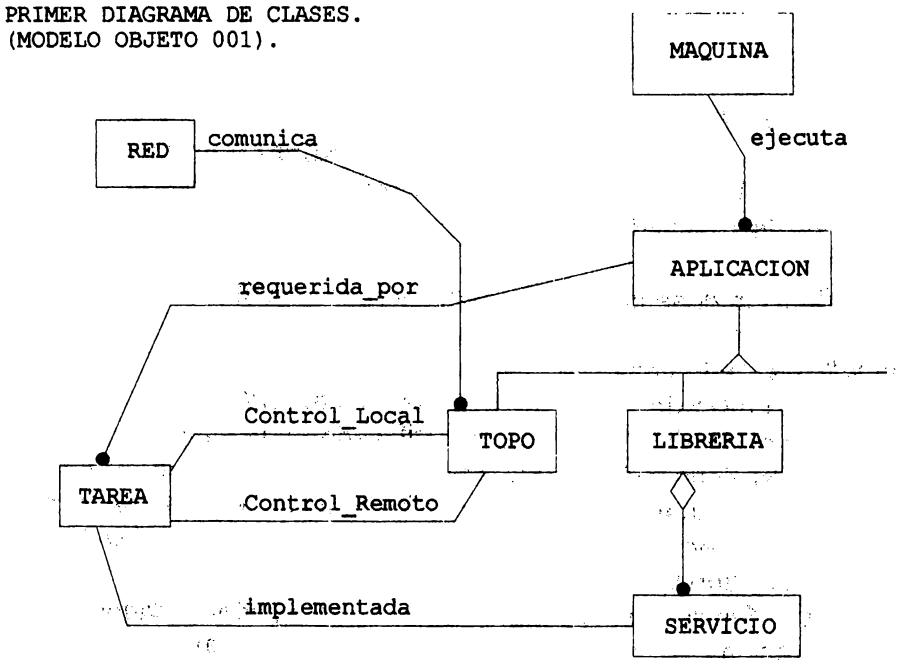


*Fig. 2*

Este diagrama sirve para realizar una primera descripción de las clases de objetos que se identifican y la relación entre las mismas:

- Un objeto de la clase *Red*, comunica un conjunto de instancias de la clase *Topo*.
- Una *Máquina* puede ejecutar varias *Aplicaciones*.
- Cada instancia de la clase *Aplicación* se ejecuta en una sola máquina.
- Las clases *Topo y Librería* son dos especializaciones de la clase *Aplicación*.
- La clase *Librería* esta constituida por un conjunto de *Servicios*.
- La clase *Topo* se utiliza para el control de *Tareas*.
- Una *Tarea* es requerida por una *Aplicación*.
- Una *Aplicación* puede requerir varias *Tareas*.
- Una *Tarea* está implementada como un servicio de una librería. Se identifica una clase *Tarea* para permitir llamar los servicios de una librería en forma asíncrona y desde una máquina distinta a la que ejecuta el servicio.
- Una *Tarea* es controlada necesariamente por dos *Topos*, uno en la misma máquina indicado en el diagrama con la relación *control_local*, y otro que puede estar en la misma máquina o en otra máquina indicada en el diagrama con la relación *control_remoto*

## *Instancias*

PRIMER DIAGRAMA DE INSTANCIAS.

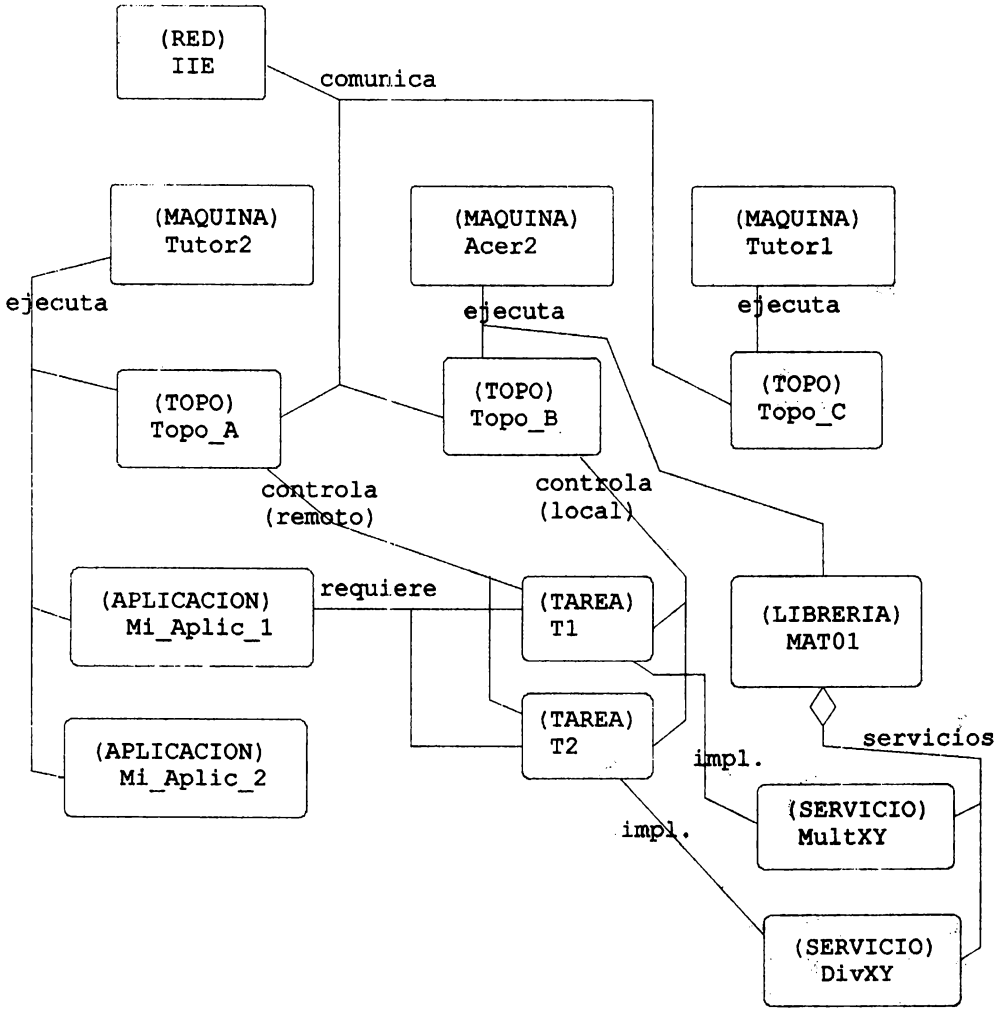


*Fig. 3*

Este diagrama de instancias puede ser una fotografía en un momento determinado. Hay tres máquinas, identificadas como *Tutor2*, *Acer2* y *Tutor1* ejecutando cada una el Topo *Topo_A*, *Topo_B* y *Topo_C* respectivamente. Los tres topos están comunicados por la Red *IIE*.

En *Tutor2* se está ejecutando también la aplicación *Mi_Aplic_1* que ha requerido la ejecución de dos tareas, *T1* y *T2*. Estas dos tareas están implementadas como los servicios *MultXY* y *DivXY* de la librería *MAT01* que se está ejecutando en *Acer8*. Las tareas *T1* y *T2* están siendo controladas por *Topo_B* (en forma local) y *Topo_A* (en forma remota).

Analizando los dos diagramas anteriores, surge la necesidad de comunicarse desde un topo a otro en la gestión de una tarea. La misma tiene vínculos con el topo remoto y con el local. Por esto, se puede considerar una Tarea como constituida por dos partes: *FREMOTA* y *FLOCAL*. La primera es la parte que está bajo control directo del topo remoto, mientras que la segunda es la parte bajo control directo del topo local. Aquí, "remoto" y "local" son respecto de la máquina que ejecuta el servicio que implementa la tarea. El diagrama de clases de la Fig. 4 es equivalente al de la Fig. 2, excepto en el detalle agregado a la clase *TAREA*.

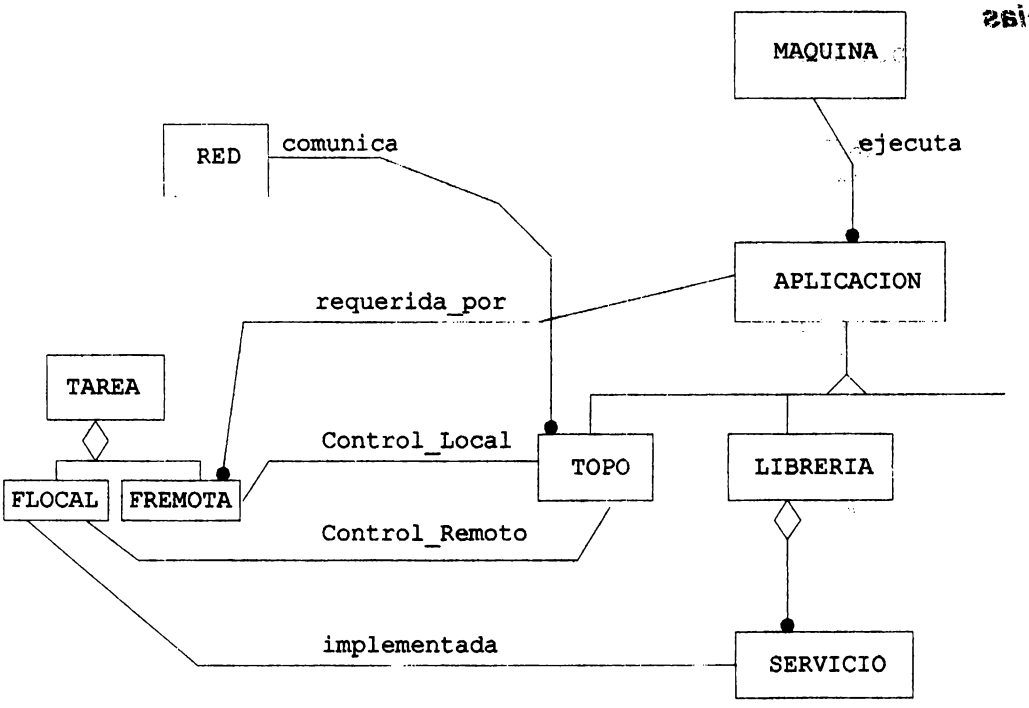


*Fig. 4*

## Modelo Dinámico

En esta sección se describen los diferentes eventos que se identifican, y la respuesta de los distintos actores a los mismos. Para identificar los eventos necesarios podemos describir una cadena posible. Por ejemplo:

*Mi_Aplicación* (corriendo en *Tutor2*) desea solicitar (por intermedio del *Topo_A*) la ejecución de las dos tareas *T1* y *T2* (en forma asíncrona) en la máquina *Acer8*, y continuar con la ejecución de la aplicación hasta un punto de sincronización, en que se especifica que deben estar los resultados de las dos tareas para continuar.

El seudo código de esa parte de Mi_Aplicación sería:

1. procesamiento previo;
2. solicitar servicio *multxy* de la librería *MAT01* en la máquina *Acer8*;
3. solicitar servicio *divxy* de la librería *MAT01* en la máquina *Acer8*;
4. continuar procesamiento;
5. esperar finalización de los dos servicios solicitados en 2 y 3;
6. continuar;

La cadena de eventos comienza en el paso 2. *Mi_Aplicación* decide solicitar el servicio *multxy* de la librería *MAT01* en la máquina *Acer8*. Para lograr esto, se comunicará con el *Topo_A* con un mensaje en que solicita la creación de la tarea. El *Topo_A* creará la parte remota de la tarea (*FREMOTA)* y se comunicará con el *Topo_B* instruyéndole sobre la solicitud. El *Topo_B* buscará la librería y el servicio solicitados, si los encuentra intentará crear la parte local de la tarea (*FLOCAL*), lanzará la ejecución del servicio y comunicará al *Topo_A* del éxito de la operación.

Durante la ejecución del paso 2, los topos *Topo_A* y *Topo_B* pasaron por una sucesión de estados hasta volver al estado original, y se crearon las instancias de *FLOCAL* y *FREMOTA* que forman la instancia de TAREA que soporta el servicio solicitado. También se crearon instancias de *SERVICIO* y *LIBRERIA*, si aún no existían las solicitadas.

El paso 3 desencadena una secuencia similar de eventos, en este caso para instanciar la tarea *T2*.

Como resultado de los pasos 2 y 3 se estarán ejecutando en paralelo los servicios *multxy* y *divxy* en la máquina *Acer8*. Con la finalización de cada uno de estos servicios se generan cadenas de eventos desde *Acer8*.

El paso 4 continua con el procesamiento de rutina de *Mi_Aplicación*.

El paso 5 espera por la finalización de las dos tareas lanzadas para continuar con el procesamiento de rutina.

Las tareas pueden terminar antes de alcanzar el paso 5 o no. Por ejemplo, supongamos que *T1* terminó durante la ejecución del paso 4 y que *T2* no. Luego, dado que se debe sincronizar, al llegar al paso 5 se deberá esperar la finalización de *T2* para poder continuar.

Cuando finaliza cada una de las tareas se desencadena la siguiente serie de eventos desde *Acer8*: el *Topo_B* (de control local) es avisado de la finalización del servicio requerido, por lo cual se contacta con el *Topo_A* (de control remoto) y lo notifica de la finalización del servicio y de los resultados del mismo. En este punto, el *Topo_B* podría eliminar la parte *FLOCAL* de la tarea finalizada. El *Topo_A* anota en la parte *FREMOTA* de la tarea que el servicio finalizó y pasa los resultados del mismo.

## Modelo Funcional. Implementación

La Fig. 1 muestra los módulos de una posible implementación. El módulo Topo.exe es una aplicación Windows con su ventana principal, y recibe los mensajes que provienen de los eventos de la red vía el módulo Winsocket.dll.

El módulo NetTopos.dll contiene el conjunto de servicios que es necesario prestar a las aplicaciones que pretendan ser usuarias del servicio NetTopos. Los módulos "topito" son procesos despachados en forma remota, que están ejecutándose bajo la supervisión del topo que se encuentra en su misma máquina.

Las flechas en el diagrama indican el sentido en que se transmiten las acciones. Para una descripción completa de los flujos de control posible nos basaremos en la Fig. 5, en la que se ven dos topos.

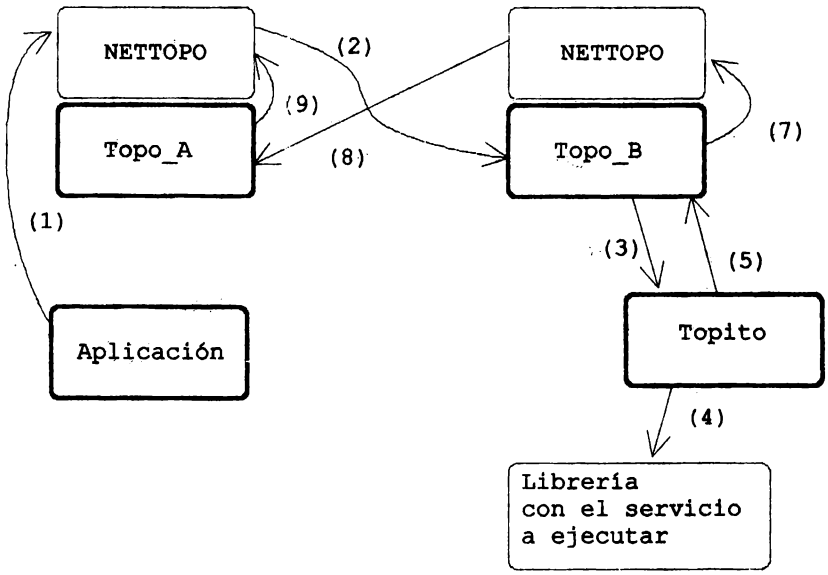


*Fig. 5*

En la Fig. 5, las cajas con borde grueso son procesos (aplicaciones bajo Windows) y las de borde fino son librerías. La "Aplicación" que se está corriendo en la máquina del *Topo_A* solicita que se ejecute un servicio de una librería creada por el usuario en la máquina del *Topo_B*.

## Ejemplo de aplicación. Resultados obtenidos

Como se expresó, el propósito principal al desarrollar el administrador NetTopos fue el de paralelizar el cálculo de los simuladores de sistemas energéticos. Estos sistemas tienen entradas que están descriptas como fuentes aleatorias, y la simulación global consiste, por lo general, de un gran conjunto de simulaciones determinísticas (con una crónica de los datos de entrada), que permita construir los histogramas de las variables de interés.

Abstrayéndonos de la aplicación en particular, se puede ver la misma como la necesidad de crear un histograma de una variable aleatoria, para lo cual se cuenta con una función que devuelve un valor posible de la variable en cada llamada.

Este tipo de cálculo se presta naturalmente para su paralelización en una red (que se comporta como una arquitectura multiprocesador distribuida), dado que la simulación del comportamiento del sistema en cada una de las crónicas es independiente de las otras, y los tiempos de simulación de cada crónica son (en una microcomputadora convencional tipo 486) del orden de 15 minutos, lo cual hace que el tiempo de comunicación y transferencia de datos entre los procesos distribuidos sea despreciable en relación al tiempo total de cálculo.

Dado que cada evaluación de la función es independiente de las demás, es posible realizar en paralelo varias evaluaciones. De esta manera estamos (teóricamente) dividiendo el tiempo de cálculo entre el número de máquinas disponibles (*N*) para realizar las evaluaciones. Naturalmente el "speed-up" real es menor que *N*, pero el tipo de aplicación hace que para máquinas homogéneas crezca casi linealmente con N.

El simulador SimEnerg se encuentra en uso actualmente en el IIE, ejecutando sobre la red de máquinas heterogéneas del Instituto.

## Líneas de Trabajo

El presente trabajo se enmarca dentro del proyecto de Procesamiento Concurrente y Paralelo que se desarrolla en el LIDI. En el mismo, entre otros temas, se estudia el uso de sockets como soporte de comunicaciones [Ramo96] [Russ96] y su utilización para el desarrollo de algoritmos paralelos. En esta línea, también se están realizando experiencias con PVM y MPI.

Por otra parte, dentro de los temas de procesamiento paralelo en el LIDI, se están utilizando máquinas paralelas basadas en transputers, sobre las cuales se han desarrollado diversos algoritmos, y en DSP (Digital Signal Processors), en particular trabajando sobre simuladores y en el desarrollo de una máquina Multi-DSP (en conjunto con la Universidad de la República y con la UNCPBA) [Ramo94] [Cose96].

## Conclusiones

La utilización de una red de computadoras heterogéneas para "simular" una máquina paralela, en muchos casos es la única alternativa posible teniendo en cuenta los costos de una máquina real. De todos modos, es una herramienta útil para la experimentación sobre distintos algoritmos distribuidos.

En este sentido, es importante el contar con un administrador que permita distribuir tareas sobre los distintos nodos, brindando un nivel de abstracción al usuario que en la mayor parte de su trabajo programa en un lenguaje convencional, y solo debe especificar en qué momento desea comunicarse.

Este tipo de soporte permite obtener resultados, que si bien no son absolutos pues una máquina paralela real contiene, por ejemplo, vías de comunicación más veloces que una LAN, al menos son relativos al estudiar la paralelización de algoritmos.

Los datos, material y algoritmos mencionados en el trabajo se encuentran disponibles en el LIDI y el IIE.

## Bibliografía


[Andr91] G. Andrews, "Concurrent Programming", Benjamin/Cummings, 1991.

[Coff92] M. Coffin, "Parallel programming. A new approach", Prentice Hall, Englewood Cliffs, 1992.

[Cose96] A. Cosentino, L. Acosta Burllaile, M. De Andrea, A. De Giusti, "Evaluación de Algoritmos de procesamiento paralelo sobre DSP y multi-DSP", enviado al 2° CACIC, 1996.

[Fort85] P. Fortier, "Design and analysis of distributed real-time systems", McGraw-Hill, 1985.

[Heer91] D. Heermann, A. Burkitt, "Parallel Algorithms in Computational Science", Springer-Verlag, 1991.

[Hoar85] C. A. R. Hoare, "Communicating Sequential Processes", Pentice-Hall, 1985.

[Huss91] Z. Hussain, "Digital Image Processing. Practical Applications of Parallel Processing Techniques", Ellis Horwood Limited, 1991.

[Hwan93] K. Hwang, "Advanced Computer Architecture. Parallelism, Scalability, Programmability", McGraw Hill, 1993.

[Laws92] H. Lawson, "Parallel processing in industrial real time applications", Prentice Hall 1992.

[Leig92] F. T. Leighton, "Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes", Morgan Kaufmann Publishers, 1992.

[Micros] Manuales de Microsoft Windows.

[Mors94] H. S. Morse, "Practical Parallel Computing", AP Professional, 1994.

[Niel90] K. Nielsen, "Ada in Distributed Real-Time Systems", McGraw-Hill, 1990.

[Ramo94] H. Ramón, C. Russo, F. Tinetti, M. Naiouf, A. De Giusti, "Procesamiento Paralelo: Experiencias con Transputers y DSP", First International Congress of Information Engeneering, Bs. As., 1994.

[Ramo96] H. Ramón, C. Russo, A. Anderson, D. Dirazar, A. De Giusti, "Paralelización de un algoritmo de compresión que utiliza diccionario estático", enviado al 2° CACIC, 1996.

[Russ96] C. Russo, H. Ramón, A. Anderson, D. Dirazar, A. De Giusti, "Algoritmo Distribuido de Compresión de Datos. Una Experiencia Comparativa con Sockets y PVM", enviado al 2° CACIC, 1996.

[Tine93] F. Tinetti, A. De Giusti, "Una aplicación de procesamiento paralelo sobre LAN", Cuarto Encuentro Académico Tecnológico, 1993.

## Anexo 1

```
program Ejemplo;
uses
  utopo, ...;

var
  s, tarea : integer;
  hde, hds : Thandle;
  nde, nnds : longint;
  p : pointer;
.
begin
   AltaAplic;

   .
   { Preparación de datos de entrada }
   nde:= Numero de bytes de datos.
   hde:= GlobalAlloc( GMEM_MOVEABLE, nde);
   p:= GlobalLock(hde);
   Cargar datos en P^;
   GlobalUnLock(hde);
   s:= CrearSMF;          { creación de un semáforo para control }
   if s>0 then begin
                tarea:= DespacharServicio( s, TopoViejo, MiLibrería, MiFunción, hde, nde );
                GlobalFree(hde); {liberamos los datos de entrada }
                if tarea<= 0 then begin
                                  DestruirSMF(s);
                                  Error('No pude despachar servicio');
                                 end
                           else begin
                                  { Esperar, haciendo otra cosa, hasta que el semáforo
                                   indique que se terminó la tarea despachada }
                                  repeat
                                     Hacer otra cosa, dar control a Windows, etc.
                                  until EsCero(s);
                                  ResultadoServicio( tarea, hds, nds );
                                  p:= GlobalLock(hds);
                                  Procesar resultados p^;
                                  GlobalUnlock(hds);
                                  GlobalFree(hds);
                                 end
                end
end.
```