La recursión como un estilo de programación dentro de los Sistemas Distribuídos

Roberto Gómez Cárdenas *
INRIA-Rocquencourt
Dom. Voluceau, Rocquencourt BP 105
78153 LE CHESNAY Cedex
France
e-mail : Roberto.Gomez-Cardenas@inria.fr

Abstract

El presente trabajo presenta un nuevo esquema para la concepción de algoritmos dentro de los sistemas distribuídos llamado ola recursiva. Esta basado en la recursión, la herramienta RPC, y una primitiva concurrente. El objetivo principal es dotar al diseñador de un esquema que le permita desarrollar algoritmos con un alto grado de abstracción. Aparte de mostrar el esquema, el artículo presenta tres variantes de la ola recursiva, asi como un ejemplo de aplicación de cada una de ellas.

Introducción

La computadora es una herramienta que día a día esta expandiendo su uso en diversas áreas de la vida diaria. A medida que el hombre se sirve de ella, y aprende a explotarla exige más. Esto ha provocado una *evolución* de la computadora. No podemos comparar el potencial de las computadoras de hoy en día con las de hace 10 o 15 años, (hace 12 años nos sorprendíamos con los 128K que la Apple IIe nos ofrecía). Un resultado de esta evolución son los sistemas distribuídos. Los algoritmos desarrollados dentro de este tipo de sistemas estan basados en el envio y recepción de mensajes. Esto provoca que la concepción e implementación de algoritmos dentro de estos sistemas no sea una tarea simple, y obliga al programador a utilisar una técnica diferente a la utilizada en las computadoras secuenciales. El programador de algoritmos distribuídos debe de tomar en cuenta diversos factores, como por ejemplo la topología de la red, la sincronización en el envío y recepción de mensajes.

El presente artículo presenta un nuevo esquema para la concepción de algoritmos dentro de los sistemas distribuídos. Se encuentra basado en el concepto de recursión, así como en la herramienta RPC (de sus siglas en ingles: *Remote Procedure Call*). El esquema proporciona al usuario un alto grado de abstracción. Tomando en cuenta que en un ambiente secuencial el

*Trabajo desarrollado en el laboratorio INRIA, bajo la dirección de los profesores Ivan Lavallée y Gerard Florin

programador no se preocupa por la gestión del almacenamiento de registros ni de las direcciones de la computadora; nuestro trabajo tiene por objetivo que el programador no se preocupe por el manejo del envio/recepción de mensajes, y se concentre en la solución del problema para lo cual está elaborando el algoritmo.

El trabajo se encuentra organizado de la siguiente forma: la primera parte presenta la *ola recursiva* explicando cada una de las diferentes herramientas necesarias para su especificación. Cada una de las tres secciones siguientes muestran una variante de la *ola recursiva* así como un ejemplo de aplicación de cada una. Por último presentamos nuestras conclusiones.

1 La "ola" recursiva

El esquema de programación propuesto tiene por nombre *ola recursiva*. Este nombre es debido a las técnicas que utiliza: *los algoritmos de tipo ola*, y a la *recursividad*. Los algoritmos de tipo *ola* fueron presentados por la primera vez en [Chan82]. Estos tipos de algoritmos estan asociados al concepto de difusión de calculo, y esta compuesto de dos fases. La primera fase permite una propagación, (broadcasting), de información a todos los sitios que componen una red; la segunda realisa un recolecta de resultados. Estos resultados pueden ser utilisados como una simple notificación de que la propagación termino, o puede incluir información sobre ciertos sitios. Bajo este contexto es fácil comprobar que los dos recorridos efectuados por la *ola* no son forzosamente los mismos, (el camino de la propagación puede ser diferente al del regreso). Encontramos diferentes trabajos sobre estos algortimos, (ver [RaHe88] o [Tel94], nosotros presentamos un nuevo tipo de *ola* que utilisa el RPC (Remote Procedure Call) en combinación con una primitiva concurrente para su propagación, y el concepto de recursión para recuperar la información.

Una *ola recursiva* es antes que nada un procedimiento el cual, durante su ejecución, efectua n o más ejecuciones paralelas de él mismo en otros lugares.

La ejecución de este tipo de procedimiento sobre una red de procesadores genera una topología virtual de árbol, con las características siguientes:

- La raíz del árbol es el procesador que realizó la primera ejecución del procedimiento;
- los diferentes nodos del árbol representan procedimientos que esperan la terminación de las llamadas precedentes;
- las hojas del árbol representan ya sea la ejecución activa del procedimiento o un procedimiento que no hace ninguna llamada.

Tomemos como ejemplo la red representada en la figura 1. En la parte (a) el procesador B lanza la primera ejecución sobre los procesadores A y D. D a su vez ejecuta sobre E el cual lo hace sobre H. Lo mismo ocurre con C, F y G.

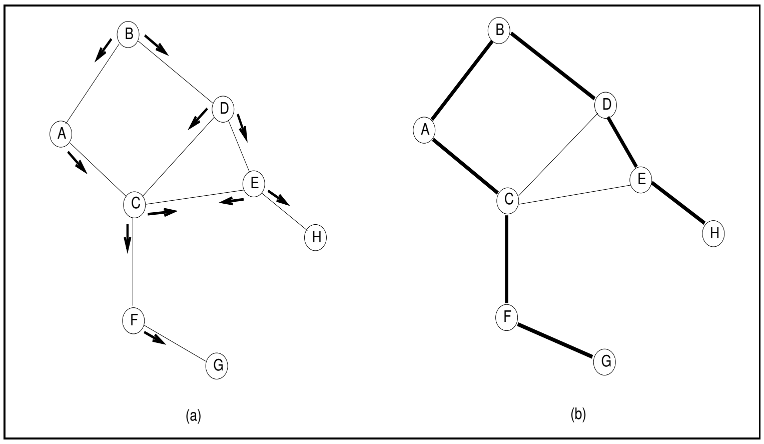

Figure 1: *Generación de una topología virtual de árbol a partir de la ejecución de una ola recursiva*

El procedimiento que define una *ola recursiva* puede formar parte de un conjunto de procedimientos realizados para un objetivo en común. Este conjunto puede englobar una estructura de datos, como por ejemplo el tipo *package* del lenguaje ADA, o el concepto de encapsulación definido en los sistemas orientados a objetos. Esta estructura de datos puede incluir el identificador del procesador activo, los identificadores de los vecinos, así como datos propios al problema a tratar.

En el presente trabajo asumimos que todas las comunicaciones y sincronizaciones entre las diversas ejecuciones se encuentran regidas por el esquema del RPC. Las líneas de comunicación son seguras, y ningun tipo de falla es considerado. Con el fin de controlar las diferentes ejecuciones del procedimiento sobre un mismo procesador, utilisamos variables comunes a las ejecuciones, cuyo acceso se controla mediante un candado.

1.1 Definición de una ola recursiva

Para definir una *ola recursiva* se necesita una primitiva de concurrencia; así como una primitiva para la ejecución de un procedimiento definido en otro procesador, (es decir el RPC).

1.1.1 Primitiva de concurrencía

Esta primitiva nos sirve para la generación de tareas paralelas, y debe de respetar la sintaxis siguiente:

par i **in** <campo> **do**
    < BLOQUE DE INSTRUCCIONES >
enddo ;

Esta instrucción activa una "tarea" por cada elemento incluido dentro de <campo>. Esta instrucción es considerada como terminada cuando todos las "tareas" generadas terminan su trabajo.

1.1.2 Sintaxis del RPC

Como se dijo anteriormente la *ola recursiva* necesita de un mecanismo de RPC. Este debe de cumplir con la sintaxis siguiente:

<nombre_pcd> (<lista_paramétros>) **on** <idt_prcs>;

Es decir un procedimiento llamado <nombre_pcd> es ejecutado en el procesador indicado por <idt_prcs>, con los parámetros definidos dentro de <lista_paramétros>. Esta primitiva y la anterior se encuentran ligadas en el sentido de que <idt_prcs> es considerado como una variable que puede tomar su valor de la variable <campo> definida en la primitiva anterior.

1.2 Esquema general de la Ola Recursiva

En la figura 2 encontramos el esquema general de la *ola recursiva*. Este esquema utilisa un lenguaje de descripción próximo al lenguaje ADA.

```
1  procedure ola_recursiva( <parámetros> ) is
2  <Bloque de declaraciones>
3  begin
4      <Bloque de instrucciones A>
5      if <condición> then  - Condición para detener la propagación de la ola
6          <Bloque de instrucciones B>
7          par i in <campo> do
8              <Bloque de instrucciones C>
9              ola_recursiva(<parámetros>) on i; - Llamado del RPC
10             <Bloque de instrucciones D>
11         enddo;
12         <Bloque de instrucciones E>
13     endif ;
14     <Bloque de instrucciones F>
15 end ola_recursiva;
```

Figure 2: *Esquema formal de la ola recursiva.*

El esquema se encuentra enumerado para su mejor entendimiento. Los <Bloque de instrucciones> representan cualquier secuencia de intrucciones. En la línea 2 se encuentra la declaración de variables locales. La condición de la línea 5 va a determinar cuando va a dejar de propagarse la *ola*[1]. Una vez que la *ola* se detiene el bloque de instruciones en la línea 14 se encarga de realizar los últimos cálculos y de enviar los resultados al procesador que activó el procedimiento.

Entre las líneas 7 y 11 se realizan las tareas paralelas, y es el lugar donde se efectua la propagación de la *ola recursiva*. En efecto en la línea 9 se ejecuta el RPC en los diferentes procesadores. Los identificadores de los procesadores, a donde es propagada la ola, son tomados de la varibale <campo> de la línea 7.

A continuación presentamos tres variantes de la *ola recursiva*. En cada una presentamos un ejemplo de aplicación.

2 La Ola Recursiva Secuencial

La forma más simple de realizar una llamada recursiva de un procedimiento dentro de un sistema distribuído el de llamar/activar ese mismo procedimiento una sola vez sobre otro procesador. En este caso el árbol de ejecución se reduce a una lista encadenada, donde el primer elemento es el iniciador del calculo.

En algorítmica distribuída la única estructura de control, que se presta a este tipo de cálculos, (aparte de la lista encadenada), es el anillo virtual. Si suponemos que tal estructura

[1] El equivalente a la condición utilisada para detener la recursión dentro de un procedimiento recursivo. Cabe resaltar que esta condición es diferente a la condición de terminación del algoritmo

existe, y que cada procesador conoce su sucesor y su predecesor, entonces la *ola recursiva* puede ser implementada muy fácilmente. Por otro lado, no habra más que una sola ejecución activa del procedimiento: la efectuada por el último procesador de la cadena. El esquema general de la ola recursiva secuencial se encuentra definido en la figura 3.

```
procedure ola_sec( <parámetros> ) is
    < Bloque de declaraciones >
begin
    < Bloque de instrucciones A >
    if < condición > then
        <Bloque de instrucciones B>
        ola_sec(< parámetros >) on sucesor;
        <Bloque de instrucciones D>
    endif
    <Bloque de instrucciones F >
end ola_sec;
```

Figure 3: *Esquema de la ola recursiva secuencial*

Comparando el esquema de la figura de arriba con el de la figura 2 podemos constatar la ausencia de los bloques de instrucciones C y E. En realidad el bloque B y el C estan agrupados dentro de B. Lo mismo ocurre con los bloques E y F. Por otro lado no se incluye la primitiva *par* debido a que el RPC no se realiza en paralelo (solo es activado un procedimiento a la vez).

2.1 Ejemplo de la ola recursiva secuencial

El ejemplo escogido es la implementación del algoritmo de Chang & Roberts para la elección sobre un anillo. Esta solución parte de la hipótesis de que todos los identificadores de los procesadores son diferentes, y tiene como objetivo el eligir el procesador cuyo identificador sea el más grande (o el más chico según el caso). El identificador del iniciador es utilisado como condición de detención de la propagación de la *ola*. Esta propagación se detendra una vez que el predecesor del procesador iniciador ha sido visitado [2]. A cada ejecución del procedimiento el parámetro *max* es actualizado, con el valor del identificador más grande. En todo momento solo una "copia" del procedimiento esta siendo ejecutada. Una vez terminada la llamada recursiva la *ola* "regresa" el paramétro *max*, que contiene el valor del más grande identificador de todos los procesadores visitados por la ola. El hecho de que no existan identificadores repetidos da como resultado que *max* no solo contenga el identificador de un procesador único, sino que además sea el elegido. El algoritmo se encuentra descrito en la figura 4.

Un ejemplo de ejecución se encuentra en la figura 5. En este caso es el procesador 3 el que inicia el cálculo. Al final el procesador 25 termina siendo el elegido. El valor incial del parámetro *max* es 3 y como se puede constatar es actualizado en los sitios 10 y 25. Una vez que

[2]También podríamos detenernos una vez que el iniciador recibe la llamada recursiva que él mismo inicio.

```
ego: id_proc; - Identificador del procesador
sucesor: id_proc; - Identificador del sucesor en el anillo
elegido: id_proc; - Identificador del procesador elegido
procedure elección(inic, max:  id_proc) is
begin
    max := sup(max,ego);
    if  succ ≠ inic then
        elección(inic, max) on sucesor;
    endif
    elegido:=max;
end elección;
```

Figure 4: *Algoritmo de elección*

la ola llega al sitio 8 detiene su propagación y regresa el valor de *max*, (25), a todos los sitios.

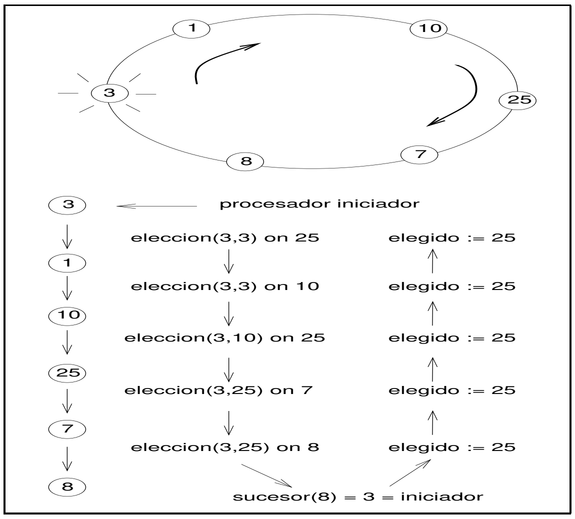

Figure 5: *Ejemplo de ejecución del algoritmo de elección*

3 La Ola Recursiva sobre una topología de arbol

Después de haber analizado el caso de la estructura más simple, veremos como se comporta la *ola recursiva* sobre un conjunto de procesadores estructurados en una topología de árbol, (estamos hablando de una estructura virtual y no física).

La estructura de árbol es bastante utilisada dentro de los algoritmos de control cuando se requiere recolectar un estado global o difundir una información. Esta forma de actuar requiere un excelente uso de los recursos de comunicación, pero considera que una topología virtual de árbol esta definida durante el cálculo, lo que supone también que la topología física es estable.

Para construir tal topología podemos utilisar los algoritmos descritos en [LaLa89] o [GHS83] .

```
procedure ola_árbol( <parámetros> ) is
< Bloque de declaraciones >
begin
    < Bloque de instrucciones A >
    if < condición > then
        <Bloque de instrucciones B>
        par i in hijos do
            <Bloque de instrucciones C>
            ola_árbol(<parámetros>) on i;
            <Bloque de instrucciones D>
        enddo
        <Bloque de instrucciones E >
    endif
    <Bloque de instrucciones F >
end ola_árbol;
```

Figure 6: *Esquema de la ola recursiva en una topología de árbol*

En este tipo de estructura cada procesador conoce su padre y sus hijos dentro del árbol, solo la raíz no tiene padre siendo esta la que inicia el cálculo. La especificación de este tipo de ola se encuentra en la figura 6.

3.1 Ejemplo de una ola recursiva en una topología de árbol

Como ejemplo presentamos un algoritmo de difusión. Supongamos que dentro de una red un procesador desea enviar cierta información al resto de los procesadores. Se considera que cada procesador conoce el identificador de sus vecinos, a los cuales tiene que enviar la información. El algoritmo se muestra en la figura 7.

Cada vez que un procesador recibe un mensaje borra el identificador del emisor de su lista de vecinos, (línea 5). Una vez hecho esto, envía el mensaje y su propoio identificador, (*ego*), al resto de sus vecinos (líneas 7 a 9). Esto se repite hasta que el mensaje llega a un procesador que no tiene a quien enviar el mensaje, (línea 6).

Es después de que todos los vecinos del procesador recibieron el mensaje, que el tratamiento de este se lleva a cabo (línea 10). Este tratamiento podría situarse en las líneas 5, 11 o 7 del algoritmo dependiendo que se quiera hacer con el mensaje.

```
procedure mensajero(mensaje:texto; padre:id_proc) is
i,ego: id_proc;
vecinos, destino: set of id_proc;
begin
    destino := vecinos - padre;
    if (destino ≠ ∅) then -Condición para detener la ola
        par i in destino do
            mensajero (mensaje, ego) on i;
        enddo ;
        < Tratamiento del mensaje >
    endif ;
end mensajero;
```

Figure 7: *Algoritmo de difusión.*

4 La Ola Recursiva Inundante

Las *olas recursivas* presentadas en las secciones precedentes tienen como característica el que trabajan sobre topologías definidas anteriormente. Esto implica la utilisación de un algoritmo que construya la topología necesaria, además las topologías de las redes construídas deben de ser lo suficientemente estables para que los algoritmos sean eficientes.

Cuando la topología de la red es suceptible de cambios, (por ejemplo en el caso de sistemas orientados a objetos, donde los objetos son creados y destruídos al mismo tiempo), necesitamos de otro tipo de herramientas. Con el fin de poder construir algoritmos bajo estas circunstancias, presentamos un tipo de *ola recursiva* que construye dinamicamente su estructura de control. Este tipo de ola construye su propia estructura, utilisando una estrategia conocida con el nombre de inundación. Es por esto que decidimos llamar a este tipo de ola: *ola recursiva inundante.*

Existen diferentes formas de controlar una estrategia de inundación de mensajes dentro de una red de computadoras. La diferencia entre cada una de ellas se encuentra en la manera de detener la propagación de los mensajes, es decir la forma en que es detenida la propagación de la *ola recursiva.* Podemos distinguir tres tipos de estrategias:

1. *limitar el numéro de llamadas recursivas:* cada mensaje es enviado al menos n veces. Esta limitación puede ser controlada utilisando un contador el cual es decrementado en cada llamada.
2. *activar solo una vez un procédimiento en cada procesador:* esto consiste en ejecutar un procedimiento sobre un procesador solamente la primera vez que este es activado. Si otros mensajes de activación llegan después al procesador, estos no seran tomados en cuenta.
3. *acarrear la lista de procesadores visitados:* una lista ordenada de procesadores es pasada como parámetro durante la propagación de la *ola recursiva.* De esta forma, la ola es

propagada solo al procesador cuyo identificador no se encuentre en la lista. Mediante esta técnica el número de mensajes es reducido, sin embargo el tamaño de estos aumenta.

Otra solución posible es la combinación de todos estos tipos de mensajes. Esto se puede constatar en la solución de Lamport, Shostak y Pease al problema de los Generales Bizantinos, (ellos utilisan una combinación de la primera y última estrategia). Con el fin de especificar la *ola recursiva inundante* utilisamos la tercera estrategia.

4.1 Especificación de la ola recursiva inundante

A continuación presentamos el esquema general de la *ola recursiva inundante*. La estructura de control construída por la estrategia de inundación es un árbol. El esquema se encuentra definido en la figura 8.

```
procedure ola_inunda(<parámetros>) is
i:id_procs;
vecinos, procs_a_visitar: setof id_procs;
< Bloque de declaraciones >
begin
    procs_a_visitar:= vecinos - procs_a_visitar;
    < Bloque de instrucciones A >
    if < condición > then
        <Bloque de instrucciones B>
        par i in procs_a_visitar do
            <Bloque de instrucciones C>
            ola_inunda(<parámetros>) on i;
            <Bloque de instrucciones D>
        enddo
        <Bloque de instrucciones E >
    endif
    <Bloque de instrucciones F >
end ola_inunda;
```

Figure 8: *Esquema de la ola recursiva inundante*

Notemos que el esquema de este tipo de ola utilisa todos los bloques de instrucciones. Por otro lado la ola termina su propagación después de visitar todos los sitios. Dado que los identificadores de los procesadores visitados anteriormente son pasados como paramétro, la instrución: *procs_a_visitar:=vecinos - procs_a_visitar*, da como resultado el conjunto vacío lo que ocasiona que la ola detenga su propagación.

4.2 Ejemplo: un algoritmo de ruta mínima

Como ejemplo de este tipo de *ola recursiva* presentamos un algoritmo de cálculo de ruta mínima. Consideremos una red de procesadores constituída de n procesadores, entre los cuales se encuentra un procesador iniciador i. Esta red puede ser vista como un grafo, en el cual los nodos representan los procesadores, y las ramas o lados representan las líneas de comunicación. Al final del calculo el procesador i debe de conocer la ruta optimal para llegar a cualquier otro procesador. Una de las formas de tratar este problema es generar un arbol de ruta mínima que tiene como raíz al procesador iniciador.

Existen dos tipos de problema de ruta mínima, uno simétrico y otro no simétrico. La diferencia entre uno y el otro reside en el costo de la línea de comunicación. Consideremos dos nodos a y b que representan dos procesadores unidos entre ellos por una línea de comunicación. En el caso simétrico el costo de la línea de comunicación que une a los dos nodos es el mismo en ambos sentidos. Esto implica que si existe un arco (a, b) existe un arco (b, a). En el caso no-simétrico el costo entre los nodos a y b no es el mismo que el costo entre b y a.

Como ejemplo de aplicación elijimos el caso simétrico. Existen diversas soluciones al problema de ruta mínima que tratan el caso de un grafo simétrico, (por ejemplo [ChMi82], [MeSe79]). En vista de que un costo único es asignado a una línea, el cual es conocido por los dos procesadores unidos por esta línea basta con que cada procesador conozca su padre dentro del árbol de ruta mínima.

Para lograr esto último cada procesador inicia el cálculo para que, por simetría, al final de este cada procesador tenga la lista de todos los procesadores que forman el árbol de ruta mínima. A partir de este árbol cada procesador sabra cual es la ruta mínima para llegar de un procesador a otro.

A continuación mostraremos como construir este árbol con la ayuda de la *ola recursiva*. Para una mayor comprensión dividimos la explicación en dos partes.

4.2.1 Los datos

La *ola recursiva* necesita que cada sitio contenga los siguientes datos globales :

1. **vecinos:** los identificadores de los procesadores vecinos al procesador activo;
2. **costo_lin_vecinos(i):** costo de la línea que une al procesador activo con su vecino i;
3. **costo_opt:** costo optimal calculado hasta un momento dado;
4. **ego:** identificador del procesador activo;
5. **padre:** identificador del padre del procesador activo.

4.2.2 Esquema de la solución

El esquema de la ola recursiva se encuentra descrito en la figura 9.

```
procedure ola_ruta( costo:int; activador: id_procs ) is
i,ego: id_procs;
vecinos: setof id_procs;
begin
    if (costo < costo_opt) then
        padre := activador;
        costo_opt := costo;
        par  i in  vecinos - activado do
            ola_ruta (costo + costo_lin_vecino(i), ego) on i
        enddo
    endif
end ola_ruta;
```

Figure 9: *Algorimo de ruta mínima utilisando la ola recursiva*

En general el algoritmo se comporta de la siguiente forma: Cada vez que un procesador es activado verifica si el costo que le llega como parametro de entrada, (*costo*), es mejor que el costo que había calculado hasta ese momento. Si este no es el caso, no se efectua operación alguna, la ola no es propagada. En caso afirmativo el procesador actualiza el valor del costo, y tomo como padre al procesador que lo activṕ.

5 Conclusión

Mucho tiempo ha pasado desde que una computadora se programaba a base de tarjetas perforadas. Las necesidades de hoy en día nos obligan a buscar nuevas formas de explotación de la computadora. El esquema de programación presentado es una propuesta para la concepción de algoritmos dentro de los sistemas distribuidos.

Este esquema ha sido probado e implementado en diversas computadoras, presentando un buen desempeño. Los esquemas de los algoritmos son claros, lisibles y presentan todos la misma estructura. La diferencia entre las diversas implementaciones de una *ola recursiva* varía en la implementación del RPC y de la primitiva *par*. Algunos de los algoritmos implementados siguiendo este esquema son el de elección, ruta óptima, construcción de una topología de árbol sobre una red, multiplicación de matrices siguiendo el metodo de Strassen, y una variante del algoritmo SSS*.

La principal ventaja que presenta la *ola recursiva* es el nivel de abstracción que proporciona, así como la transparencia de todo lo concerniente al manejo de mensajes, y de los incovenientes relacionados con la administración del sistema, como por ejemplo la topología de la red de procesadores. Esta abstracción permite al usuario de concentrarse más en la naturaleza del problema mismo, que en el manejo de las herramientas para la solución del problema.

Diversos trabajos se han efectuado alrededor de este esquema. Se diseño una metodología especial para la concepción de algoritmos siguiendo el esquema de la *ola recursiva*. Esta metodología presenta la ventaja de que al mismo tiempo que se construye el algoritmo, se sientan las bases para la prueba del mismo. La recursión en un ambiente centralisado tiene una reputación de "elegante", pero en muchos casos no es eficiente; el objetivo de la *ola recursiva* es demostrar que en un ambiente distribuído la recursión es una herramienta elegante y eficiente.

References

[BiNe83] Birrell D.Andrew and Nelson Jay Bruce - *Implementing Remote Procedure Calls.* ACM Trans. on Computing Systems, Vol. 1, No. 3 August 1983, pp. 222-328.

[ChMi82] Chang K.M., Misra I. - *Distributed computation on graphs, Shrotest path algorithm* Comm. ACM, Vol. 25, No. 11, nov 1982. pp. 833-837

[Chan82] Chang Ernest - *Echo algorithms: depth parallel operations on general graphs* IEEE Trans. on Soft. Eng. Vol. se-8 No.4 July 1982

[GHS83] Gallager R.G., Humblet P.A., Spira P.M. - *A distributed Algorithm for Minimum Spanning Tress* ACM Trans. on Prog. Lang. and Systems Vol 5:66-77, Jan, 1983

[JoGo88] Jones G, Goldsmith M. - *Programmin in OCCAM2* Prentice Hall UK 1988

[LSG82] Lamport L., Shostak R., Pease M. - *The Byzantine Generals Problem* ACM Toplas, Vol. 4, No. 3 July 1982, pp. 382-401.

[Lav90] Lavallée Ivan - *Algorithmique parallèle et distribuée.* Hermès ed. Paris 1990.

[LaLa89] Lavallée Ivan, Lavault Christian - *Yet Another Distributed election and Spanning Tree Algorithm* Rapport de Recherche No. 1024, INRIA-Rocquencourt April 89, France

[MeSe79] Merlin F, Segall A. - *A failsable distributed routing protocol* IEEE Trans. on Comm. Vol. 27, No. 9, sept 1979, pp. 1280-1287

[RaHe88] Raynal M., Helary J.M. - *Synchronisation et contrôle des systèmes et des programmes réspartis* Eyrolles ed. Paris 1988.

[Tel94] Tel Gerard - *Introduction to Distribued Algorithms* Cambridge Universite Press, 1994