El Protocolo CBCAST en Comunicación de Objetos.

Luis A. Alvarez G.
Instituto de Informática
Universidad Austral de Chile.
e_mail : lalvarez@uach.cl

Raúl Monge A
Departamento de Informática
Univ. Técnica Federico Santa. María.
e_mail : rmonge@inf.utfsm.cl

Resumen. En este trabajo se implanta el algoritmo de *multicast* CBCAST [Bir+,89] al modelo de objetos desarrollado por [Alv+,96a]. Previamente se consideran aspectos relacionados con la administración de grupos de objetos identificados ya sea por una lista ordenada de sus OIDs (*Object IDentifiers*) o con un identificador de grupo GID (*Group IDentifier*) que involucra la existencia de un Administrador de Grupos. Finalmente se adapta el algoritmo CBCAST y se entregan las conclusiones.

Palabras Claves : Sistemas Distribuidos, Protocolos de *Multicast*, Objetos.

1. Introducción.

El modelo de objetos de sistema distribuido presentado en [Alv+,96a] y la comunicación de objetos del mismo [Alv+,96b] permite comunicación de grupos de objetos de granularidad fina. Para una buena comunicación se requiere de un protocolo que logre mantener consistente el estado global del sistema distribuido. Los protocolos de *multicast* mas comunes son:

a) CBCAST y ABCAST algoritmos de orden causal y orden total respectivamente [Bir+,89],

b) *Token Passing* [Cha+,84] un algoritmo de *broadcast*; y

c) el algoritmo de grafo de propagación [Gar+,89] orientado a redes de área amplia.

Un estudio comparativo se encuentra en [Alv+,95].

1.1 Motivación.

Todos los protocolos anteriores no están especificados para objetos distribuidos, sino para procesos distribuidos.

Electra [Maf,95], una plataforma reciente para la comunicación entre objetos, hace uso de protocolos existentes para la comunicación de procesos como ISIS y HORUS [Bir+,90], sin emplear el despacho a los objetos destinos.

Este trabajo, presenta una adaptación del protocolo de *multicats* CBCAST para despacho ordenado de mensajes en objetos encapsulados por procesos de un modelo de sistema distribuido.

2. El Algoritmo CBCAST.-

Para despachar mensajes en forma ordenada, cada mensaje lleva una marca de tiempo, llamada comúnmente *timestamp* (o TS); existen dos alternativas, usar relojes de Lamport [Lam,78] o vectores de tiempo; por facilidad de implantación el autor elige esta última opción.

El algoritmo mantiene la siguiente regla: Si el envío de un mensaje m causa el envío de otro mensaje llamado m', entonces m tiene un menor TS y por lo tanto se despacha antes que m'.

Esto se denota :

Si, $send(m) \rightarrow send(m') \Rightarrow TS(m) < TS(m')$, luego $dlv(m) < dlv(m')$

En el algoritmo CBCAST usando vectores de tiempo, cada proceso del grupo tiene asociado un vector de tiempo VT (*vector time*) que corresponde a un vector de enteros, con tantos elementos como procesos tenga el grupo. A su vez, cada elemento representa el número de eventos que un determinado proceso conoce de los miembros del grupo, incluyendo sus eventos internos. La Tabla 1 muestra el algoritmo usando vectores de tiempo.

Tabla 1.- Algoritmo CBCAST usando vectores de tiempo

1. Antes de enviar un mensaje m, el proceso P_i incrementa el elemento i del vector de tiempo y le coloca la marca de tiempo VT_m al mensaje ($VT_m = VT_{Pi}$).
2. El proceso P_j recibe el mensaje m ($P_j \neq P_i$). Pj despacha m, si conoce:
   - hasta el evento de envío anterior de P_i y
   - a lo menos los eventos que P_i conoce de los restantes procesos.

   Dicho de otra forma, debe cumplirse que:

   $\forall\ k:1..n\ ,\ VT_{Pj}[k] = VT_m[k] - 1 \quad para \quad k = i\ ;$ (1)

   $VT_{Pj}[k] \geq VT_m[k] \quad para \quad k \neq i \wedge k \neq j$ (2)

   El proceso P_j no necesita aplicar la condición para sus eventos internos.
3. Cuando se despacha un mensaje m, $VT_{Pj}[k]$ se actualiza a $VT_m[k]$.

2.1 Ejemplo del Algoritmo CBCAST.

Se supone que el sistema distribuido de la Fig. 1, contiene un grupo formado por tres procesos denominados Pi_a, Pj_c y Pk_c, asociándoles los enteros 1, 2 y 3. Entonces cada proceso del grupo tendrá un VT de 3 elementos. Se asume que no existen eventos anteriores, por lo tanto todos los elementos de los VT de los procesos se encuentran en cero. La Fig. 2 muestra el ejemplo usando despacho causal. Las líneas punteadas a continuación de las puntas de flecha, indican despacho. Notese que m_2 en Pk_c, se debe retardar hasta que se cumplan las condiciones (1) y (2).

Se debe recordar que los vectores de tiempo tienen tantos elementos como objetos existan. Cada objeto está asociado a una posición y el valor indica el número de eventos que conoce de los objetos del grupo. El elemento asociado al objeto, se incrementa con cada evento [Bab+,93].

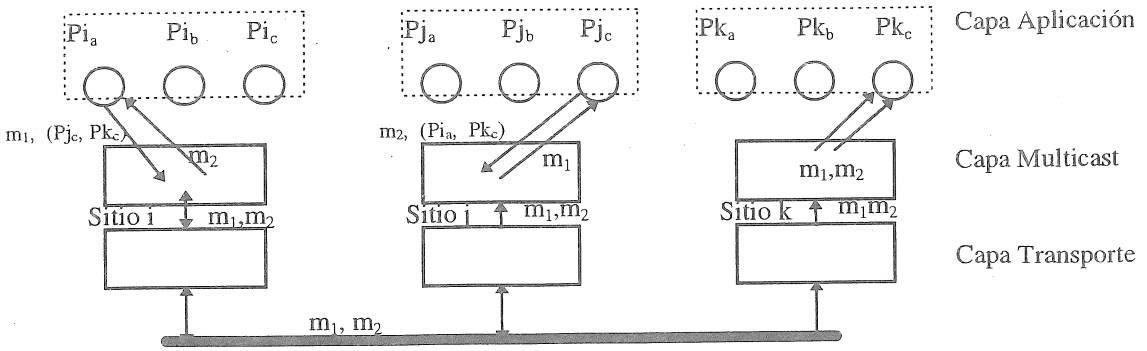

Fig.1.- Modelo y subsistema de comunicaciones.-

Explicación.- Originalmente los tres vectores de tiempo están en cero. Antes de un evento de *multicast*, la componente asociada se incrementa a 1 en cada VT. Así previo a e_i de Pi_a (*multicast* del mensaje m_1 al grupo), el primer elemento de VT_1 se coloca en 1, es decir, $VT_i = (1,0,0)$ y m_1 se marca con $VT_{m1}=VT_1$. La capa *multicast* del sitio j recibe m_1, el algoritmo verifica que se cumplen las condiciones (1) y (2) y se despacha hacia al proceso Pj_c, actualizándose $VT_2 = VT_{m1}$ (evento e_j). Previo al evento e_{j+1} (*multicast* del mensaje m_2 al grupo), VT_2 incrementa el segundo elemento quedando como $VT_2=(1,1,0)$ y m_2 se marca con $VT_{m2}=VT_2$. La capa *multicast* del sitio k toma m_2, el algoritmo se da cuenta que no cumple con la condición (2) ($VT_{m2}[1] > VT_3[1]$) por lo tanto el despacho se debe postergar. Notar que la condición (1) ($VT_{m2}[2]=VT_3[2]+1$) se satisface.

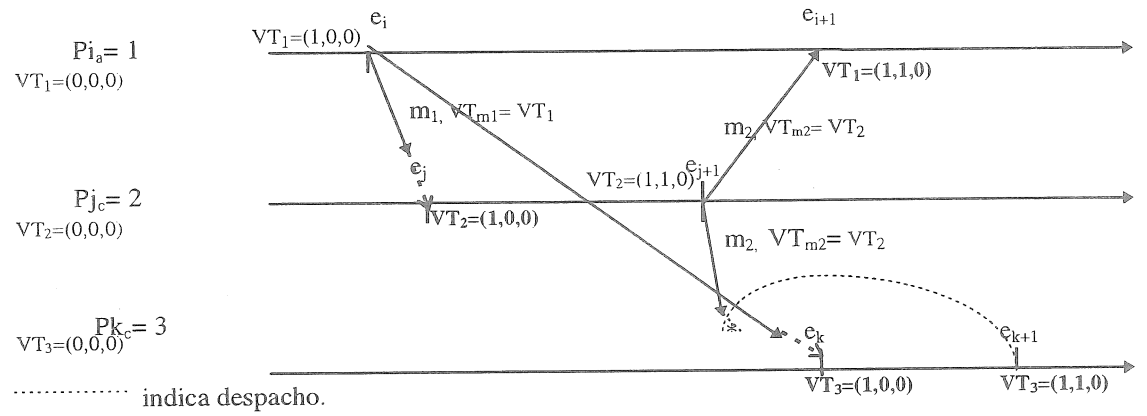

Fig. 2.- *Multicast* causal aplicando el algoritmo CBCAST

Así entonces la capa *multicast* del sitio k despacha m_1 a Pk_c, se actualiza $VT_3=VT_{m1}$ y posteriormente despacha m_2 y actualizar $VT_3=VT_{m2}$.

3. Administración de Grupos y Aspectos de Implantación.

El algoritmo CBCAST se desarrolla para *multicast* de procesos y el modelo donde se aplicará se basa en objetos, entonces previo a su adaptación se deben considerar aspectos relacionados con la administración de grupos de objetos.

El modelo de comunicaciones de [Alv+,96b] se basa en objetos de comunicaciones, correspondiendo a: OOCs (*Objects for Objects Communication*), OPCs (*Objecs for Process Communication*) y el ONC (*Object for Nodes Communication*) fragmentado en cada uno de los nodos del sistema distribuido, siguiendo las ideas planteadas por [Mak+,91].

El objeto origen envía a su OOC correspondiente, además del mensaje, información suficiente que le permita determinar las direcciones de los objetos del grupo destino. Para grupos explícitos (los miembros se conocen) envía el OID de cada uno los objetos y para grupos anónimos (los miembros no se conocen) un identificador GID (*Group IDentifier*).

Existen por lo tanto dos posibles formas de enviar esta información, mediante :

- Un listado con todos los objetos miembros del grupo. Posible de usar sólo para grupos explícitos.
- Un identificador de grupo GID. Posible de usar para grupos explícitos o anónimos.

3.1 Lista de Objetos

En la primera alternativa el objeto origen envía al OOC la lista con los OIDs de todos los objetos del grupo. El algoritmo necesita la lista ordenada para asignar las posiciones en el vector de tiempo. Es posible lograr esta lista concatenando todos estos identificadores. Se considera el OID como función de la localización de los objetos, es decir, si el objeto nombrado como `o(i,j,k)` tiene como OID al vector `(i,j,k)`, donde `i` corresponde al nodo, `j` al proceso y `k` al objeto dentro del proceso (ver [Alv+,96a]). Entonces diremos que:

$$OID_m > OID_n \text{ si } i_m > i_n \vee (i_m = i_n \wedge j_m > j_n) \vee (i_m = i_n \wedge j_m = j_n \wedge k_m > k_n)$$

De esta forma si un grupo tiene o objetos y se sigue un orden ascendente, el objeto de menor OID se sitúa como primer objeto de la lista :

$$Lista = \{OID_1, OID_2, \ldots, OID_o\}$$

Lo anterior presenta las siguientes ventajas:

- A partir de la lista se puede obtener fácilmente el grupo de objetos GO, de procesos GP y de nodos GN.
- Es posible ubicar en forma directa la localización de los objetos destinos y por lo tanto un rápido envío del mensaje.

Sin embargo, tiene a lo menos tres desventajas :

- La lista puede llegar a ser muy extensa si el grupo está compuesto de muchos objetos.
- Cada ingreso o salida de miembros al grupo obliga al objeto origen actualizar la lista.
- Como en el modelo, el OID depende de la localización, entonces cualquier migración del objeto implica modificar su OID y en consecuencia la lista del grupo.

3.2 Uso de un Identificador GID

El identificador GID debe ser usado por los objetos pertenecientes a grupos anónimos, puesto que éstos no conocen a los demás miembros del grupo; en cambio su uso es optativo para los grupos explícitos para mantener transparencia respecto a los destinatarios del mensaje.

En términos generales se requiere emplear un administrador de grupos para: conocer la membrecía de cada grupo, generar nuevos grupos, eliminar objetos cuando estos presentan fallas, ingresar nuevos objetos, destruir grupos de objetos, migrar objetos, etc. Esto último puede resultar muy complejo al usar la localización de los objetos como su OID.

Puesto que un grupo puede estar formado por objetos distribuidos en todo el sistema, los GIDs deben ser números enteros únicos no asociados a una localización como es el caso de los OIDs,

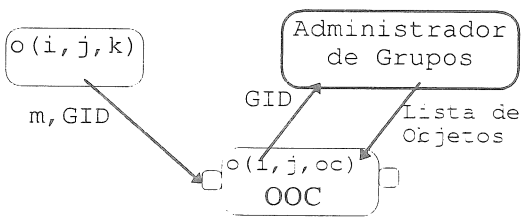

Fig. 3.- El OOC y el Administrador de Grupos

De esta forma y como lo indica la Fig. 3, el objeto origen envía el mensaje m y el identificador GID al objeto de comunicaciones de objetos OOC. El OOC envía éste al Administrador de Grupos, el cual devuelve la lista de objetos de GO. A partir de esta lista cada uno de los objetos de comunicaciones asociado a los grupos GO, GP y GN, resolverán las direcciones de los objetos destinos.

Las funciones básicas de un Administrador de Grupos son :

`GID = create()` : Crea un grupo vacío de objetos al que le asigna un identificador GID único.

`join(GID,OID)` : Agrega un objetos con identificador OID al grupo de objetos identificado por GID.

`leave(GID,OID)` : Retira el objeto de identificador OID del grupo de objetos identificado por GID.

`destroy(GID)` : Destruye el grupo con identificador GID.

Existirá un solo Administrador de Grupos por cada nodo, perteneciente al proceso de comunicaciones del modelo [Alv+,96b], siendo único en cada nodo que a su vez debe mantener información de todos los grupos de objetos del sistema distribuido. Ante cualquier modificación en la membrecía de alguno de los grupos, el administrador responsable del cambio deberá informar (usando algún algoritmo seguro) a todos los administradores de grupos, para efectos de actualización.

4. El Rol de los Objetos de Comunicación.

En el modelo de [Alv+,96a] los grupos pueden estar constituidos por combinaciones de objetos de un mismo proceso, de otros procesos del mimo nodo o de otros nodos.

La Fig. 4, ilustra el rol de los objetos de comunicación.

El envío de un mensaje desde un objeto origen a un grupo de objetos GO_L se realiza a través del objeto de comunicaciones OOC. El mensaje estará acompañado de la marca de tiempo y la lista de los objetos de GO_L o el GID. El OOC resuelve las direcciones de los integrantes de GO_L a partir de la lista de OIDs o por consulta al administrador de grupos. Si en GO_L existen objetos del mismo proceso despachará el mensaje a todos ellos.

Por otra parte, si existen objetos de GO_L que pertenecen a otros procesos, pasará el mensaje con la marca de tiempo y el OID de los restantes objetos al OPC, éste resolverá la existencia de objetos en otros procesos del mismo nodo y enviará el mensaje con las marcas de tiempo y la lista de objetos destinos a los OOC involucrados, los cuales despacharán el mensaje.

Finalmente, si existen objetos de GO_1 en otros nodos, entonces el OPC pasará el mensaje con la marca de tiempo y el OID de los objetos de GO_L a su fragmento de ONC, y éste lo entregará a cada uno de los restantes fragmento de ONC de los nodos pertenecientes a GN_L.

Cada uno de los fragmentos ONC de los nodos restantes pertenecientes a GN_1, enviarán el mensaje con la marca de tiempo y el OID de los objetos a sus OPCs, para que éstos a su vez los envíe a los OOCs de los procesos pertenecientes a GP_L. Los OOC por su parte despacharán el mensaje a cada uno de los objetos pertenecientes a GO_L.

4.1 El Algoritmo CBCAST en el Modelo.

Si se considera el algoritmo CBCAST con vectores de reloj, el vector VT debe contener un elemento asociado al objeto origen y, en consecuencia formar parte del grupo de objetos GO_L. Cada objeto de GO_L a su vez tiene asociado un VT con tantos elementos como objetos tenga el grupo.

El algoritmo CBCAST aplicado al modelo se muestra en la Tabla 2, y se desarrolla considerando que el objeto origen envía el GID de GO_L, y no la lista de objetos. Todo despacho se rige por las reglas dadas en la Tabla 3.

Tabla 2.- Algoritmo CBCAST para el modelo.

1. El objeto cliente `o(i,j,c)` envía al objeto de comunicaciones llamado `o(i,j,oc)`:
   - el mensaje `m`,
   - el identificador del objeto cliente OID_c, y
   - el identificador de grupo de objetos GID.

   Si el mensaje es una solicitud a varios servidores, para la respuesta deben conocer el OID_c. Si el mensaje sólo se disemina, el OID_c, puede ser innecesario.
2. El objeto de comunicaciones `o(i,j,oc)` recibe `m`, OID_c y GID, luego consulta al Administrador de Grupos por la lista con los OIDs del grupo GO_L, para obtiener el vector de tiempo VT. Posteriormente se incrementa el elemento asociado a `o(i,j,c)` de VT y marca el mensaje con

VT_m (VT_m=$VT_{o(i,j,c)}$). Despacha m a todos los objetos del mismo proceso pertenecientes a GO_L y envía m, VT_m,OID_c, y la lista con los OIDs de los restantes objetos de GO_L al objeto de comunicaciones o(i,pc,oc).

3. El objeto o(i,pc,oc) envía m, VT_m,OID_c, y la lista con los OIDs de los restantes objetos de GO_L a todos los objetos de comunicaciones o(i,j',oc) con j≠j', que despachan m a sus respectivos objetos. Si existen objetos del grupo de objetos GO_L en otros nodos se envía ésta misma información al fragmento o(nc,pc,oc) del nodo de comunicaciones n(nc).
4. El fragmento o(nc,pc,oc) perteneciente al objeto origen envía a cada uno de los restantes fragmentos de n(nc) pertenecientes al grupo de nodos GN_L: m, VT_m, OID_c y la lista de OIDs de los restantes objetos de GO_L.
5. Cada fragmento de nodo envía la información precedente a sus correspondientes objetos o(i',pc,oc) y éstos a su vez a los objetos o(i',j,oc), que pertenezcan al grupo de procesos GP_L.
6. Cada objeto de comunicaciones o(i',j,oc) con i≠i' recibe m, VT_m,OID_c, y la lista de OIDs., los cuales despachan m.

Tabla 3.- Condiciones para el despacho de mensajes en algoritmo CBCAST.

Si un objeto o(i,j,c) envía un mensaje a los objetos de su grupo GO_L, entonces cada objeto OOC perteneciente al grupo de procesos GP_L lo despacha, si cada objeto conoce:

- hasta el evento de envío anterior de o(i,j,c) y
- a lo menos, los mismos eventos de envío de los restantes objetos conocidos por o(i,j,c).

Dicho de otra forma, se debe cumplir que:

$$\forall h : 1..n \quad , \qquad VT_m[h] = VT_{o(i',j,k)}[h] + 1 \quad \text{para } h = c ;$$
$$VT_m[h] \leq VT_{o(i',j,k)}[h] \qquad \text{para } h \neq c$$

Donde,

n : número restante de objetos de GO_L ,
c : elemento de VT asociado a o(i,j,oc) y ,

Cuando un mensaje m se despacha, $VT_{o(i',j,k)}$[h] se actualiza a VT_m[h].

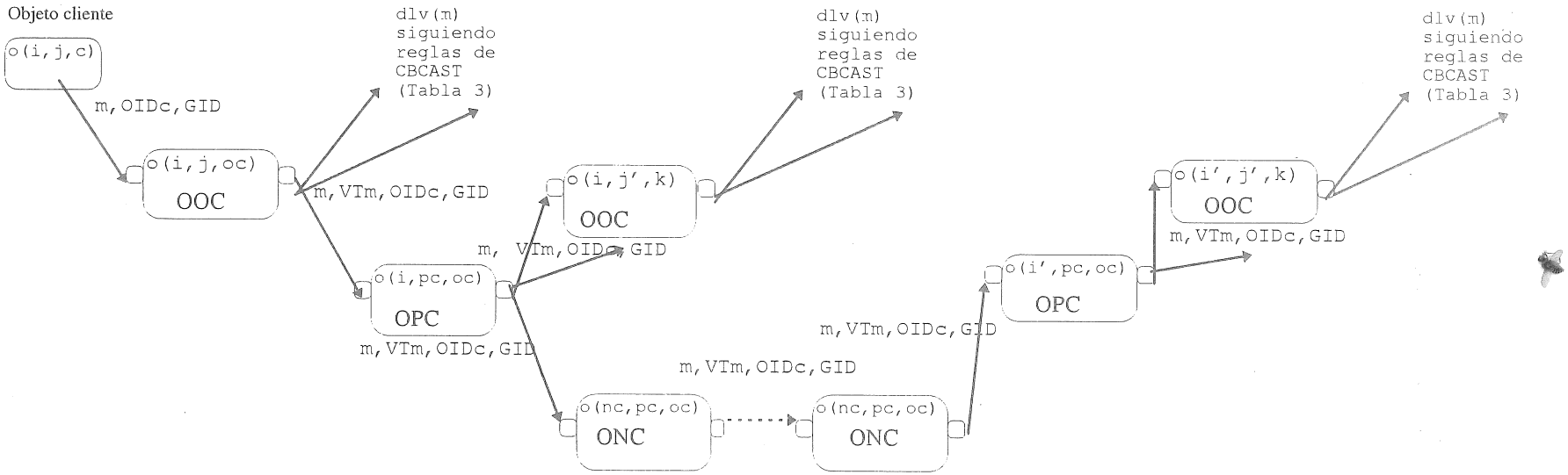

Fig. 4.- Protocolo de Comunicación en el Modelo.

5. Conclusiones

El trabajo desarrollado, permite el despacho de mensajes a nivel de objetos logrando un mejor aprovechamiento del paradigma basado en objetos a diferencia de los algoritmos tradicionales definidos en plataformas para comunicación de procesos.

El costo a pagar se refleja en la complejidad del algoritmo adaptado que incorpora nuevas entidades de despacho (objetos de comunicación).

El algoritmo CBCAST desarrollado originalmente para comunicación entre procesos se adapta al protocolo a comunicación entre objetos, con un sistema de soporte de comunicaciones diferente, basado en Objetos de Comunicación.

Los autores consideran relevante el aporte teórico en el área de comunicaciones entre objetos.

Como primera aproximación, se demuestra que el modelo de comunicación presentado por [Alv+96b] es abierto a cualquier protocolo, con tan solo reescribir los métodos necesarios en cada objeto de comunicación.

Una implantación final de un prototipo requiere de una serie de herramientas y servicios de soporte ; entre estos están lenguajes de distribución y configuración de objetos, servidores de nombres de objetos, protocolos de *broadcast* seguros para la actualización de Administradores de Grupos, etc. Los productos mencionados se deben desarrollar ad-hoc al modelo en particular implicando un gran esfuerzo de desarrollo por parte de un equipo de profesionales. Es probable que sea factible el empleo de lenguajes y plataformas recientes basadas en objetos como Java y CORBA.

Queda por desarrollar la generalización del algoritmo a grupos con objetos compartidos, desafío teórico de gran volumen y generar nuevos algoritmos al modelo.

6. Referencias.

[Alv+,95] Alvarez G., Luis y Monge A. Raúl : "Protocolos de Multicast Ordenados en Sistemas Distribuidos" Congreso y Exposición Internacional de Informática y Telecomunicaciones INFOCOM '95. pp. 101- 110. Buenos Aires, Argentina, Junio de 1995.

[Alv+,96a] Alvarez G. Luis A., Monge A. Raúl. " Un modelo de Comunicación para un Sistema Distribuido", IV Encuentro Chileno de Computación. Actas de la Sociedad Chilena de Computación, pp. 34 - 41. Universidad Austral de Chile, Valdivia, Chile, Noviembre de 1996.-

[Alv+,96b] Alvarez G. Luis A., Monge A. Raúl. "Objetos de Comunicación para un Modelo de Sistema Distribuido Basado en Objetos", IX Simposio Internacional en Aplicaciones en Informática, Anales INFONOR'96. Sección 7 pp. 1-8. Antofagasta Noviembre de 1996.-

[Bab+,93] Babaoglu, Özalp and Marzullo, Keith . " Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms". Chapter 4, Distribuited Systems, Second Edition, edited by Sape Mullender, Addison-Wesley, 1993. pp.55-96.

[Bir+,90] Birman, K., Cooper, T., Marzullo, K., Makpangou, M., Kane, K., Schmuck, F., and Wood, M., " The ISIS System Manual, Version 2.0" , May 8, 1990.

[Bir+,89] Joseph, T.A. and Birman, K.P.: "Reliable Broadcast Protocols", Distributed System edited by Sape Mullender, Addison Wesley, 1989, Chapter 14, pp 293- 317.-

[Cha+,84] Chang, Jo-Mei and Maxemchuk, N.F. " Reliable Broadcast Protocols,". ACM Transaction on Computer System 2(3): 251-273, Aug., 1984.-

[Gar+,89] García-Molina, Héctor and Spauster, Annemarie. "Message Ordering in a Multicast Enviroment", Proceeding Ninth International Conference on Distributed Computing Systems, June 1989.

[Lam,78] Lamport, Leslie. "Time, clock and the ordering of events in a distributed systems," Communications of the ACM 21(7): pp 558-565, July, 1978.

[Maf,95] Maffeis, Silvano : "Run-Time support for Object-Oriented Distributed System" Fakultät der Universität Zürich, Ph.D. Thesis, February 1995.

[Mak+,91] Makpangou, M., Gourhant I.,, Le Narzul, Jean-Pierre and Shapiro, Marc: "Fragmented Object for Distributed Abstractions " Technical Report , INRIA , France, October, 1991.-