Dynamical Grouping Model for Distributed Real Time Causal Ordering

Abderrahim BENSLIMANE and Abdelhafid ABOUAISSA

LaRIS de l'Université de Technologie de Belfort-Montbéliard
Rue du château 90010 Belfort Cedex France
Email : { abder.benslimane, abdelhafid.abouaissa}@utbm.fr

ABSTRACT

This paper proposes a dynamic hierarchical architecture of k-local groups, where k represents the number of local groups composing the group communication system. Each local group is defined as a finite set of processes, and computed by performing some tests. The proposed architecture ensures real-time causal ordering, and guarantees a dynamic reconfiguration between alive local groups members. This structure allows us to compensate for the local clocks drift by synchronizing the local clocks, and ensures real-time causal delivering. We validate this model by using CPN (Coloured Petri Nets) to obtain a state space graph. The objective is to study the behaviour of this model, by verifying different properties such as, boundedness, liveness. The simulation result shows that the hierarchical architecture respects all studied properties. Therefore the real-time causal ordering is guaranteed.

***Keyword :** causal order, multimedia systems, delta-causality, group communication systems, state space graph, Coloured Petri Nets.*

1. Introduction

Recently, the new high-speed networks based on optic fibres allow to integrate different types of applications, such as video-conference, tele-teaching, etc... With the increase use of distributed real-time systems, the ability of networks to handle real-time traffic is becoming more and more important. Due to the nature of applications and the asynchronous of the communication in the network such as ATM, the temporal relationships among media units tend to be disrupted that the playback discontinuity will be perceptible to humans.
Causal ordering [18] reduces much of the communication medium asynchrony. This abstraction guarantees that whenever a message is delivered to a site, then all causally prior messages have been already delivered to it. However, the causal ordering can not support real-time applications that is governed by the need for low delivery-time reliable multicast underlying network.
Several protocols [3, 7] have been presented to implement causal ordering of messages. These protocols use the concept of the logical clocks and time-stamps [13]. Logical clocks are a good tool to capture causal ordering messages at a logical level, but they increment only when an event happens, such as the transmission or reception of a message. So, they have little correlation to physical time. The logical clocks are not appropriate for real-time systems because these systems require the notion of physical time as constraints imposed by the deadline on the data. In other hand, multimedia data has a limit lifetime after which it can not be used. Consequently, the dependency on previous messages applies to it only for a limited time. This time constraints is referred to as Δ *-causality* [4, 21], and is defined formally in the next section.
In multimedia applications, a simple way to connect processes is to build a virtual complete graph, which allows a fast and a direct communication between two processes. However, complete graph may cause a waste of buffers, that the price of running multimedia applications will become too high to be feasible, because receivers must buffer media separately [23].
Several paper have been presented to build a hierarchical structure. [20] proposes a hierarchical communication architecture and a mixing synchronization algorithm. However, it dose not take into account the dynamic aspect of the architecture and how they resolve the clock drift in the case of a complex architecture. [23] proposes a hierarchical architecture for managing the group

communication system. Their approach, however, yields an architecture that is not fully hierarchical, mainly between subgroups. This will not reduce the control message when a distributor wants leave a subgroups, and they do not explicitly show how the new distributor makes connection with other distributors. Other protocols have been presented for reducing the amount of control information [5, 18]. These approaches do not describe the topological structure within the local group. Both these protocols do not address issue of real-time delivery that supports fault-tolerance.

In this paper, we propose a new hierarchical architecture for the group communication system. This model supports real-time causal ordering. Our solution is based on a hierarchical structure of k-groups, where k represents the number of local groups that compose the system. The local group is defined as a set of the processes, by performing some tests. Within each local group, there is only one single server that plays the role of the local root of that group. The set of local servers forms a virtual group, where one of them will play even the role of the Master server for that group system, and consequently the global root of this hierarchical architecture.

The rest of the paper is structured as follows: In section 2, we present the formal definitions of causal order and Δ-causality. In section 3, we introduce the grouping algorithm to divide the group communication system in k-local groups, and to elect the local and master servers. In section 4, we provide the correctness of the proposed architecture. In section 5, we study all behavioural properties, such as boundednes and liveness. Finally, concluding remarks are given in section 6.

2. Causal ordering concept

2.1. Causal Order

In overlapping groups, the concept of causality is of major importance in the design of the distributed systems. It ensures that messages sent before other messages should be received in that order. As there is no global clock in distributed systems, the information added to the messages must include the acknowledge of other messages in the system that were sent before it. The added information is called the interval of the dependency of that message. Thanks these informations, a message will not be delivered until all the messages that it depends upon have been delivered.

Consider the events of type *send* (that represents the send message event) and *del* (the delivery message event) produced by all the processes in groups. These events are structured by the following partial order relation [13] :

i) for any produced event e, e $\rightarrow$ e.

ii) for two events e_1 and e_2 produced by the same process such that : $e_1 \rightarrow e_2$, where $(e_1 \neq e_2)$, then it is not the case that $e_2 \rightarrow e_1$.

iii) let events e_1, e_2, and e_3 such that $e_1 \rightarrow e_2$ and $e_2 \rightarrow e_3$, then $e_1 \rightarrow e_3$.

iv) if there exists an event e that is $send_p(m)$ or $del_p(m)$ and an event e' that is $send_p(m')$ or $del_p(m')$, then e $\rightarrow$ e' or e' $\rightarrow$ e.

v) if there exists an event e that is $send_p(m)$ and an event e' that is $del_q(m)$, where $p \neq q$ then e $\rightarrow$ e'.

Such causal order is not sufficient in real-time multimedia traffic, because it has not take into account that the message can be discarded if it arrives at its destination after the expiration of its deadline. Therefore, the dependency condition of message delivery is applied only in a bound time interval [4].

2.2. Δ *-causality order*

The communication model of Δ *-causality* is mainly used in multimedia application [10] such as teleconference, video-conference, where the real-time traffic is required over networks. To preserve a high QoS of such applications, the audio/video information have a maximum transmission delays, since it can not degrade the QoS [8].

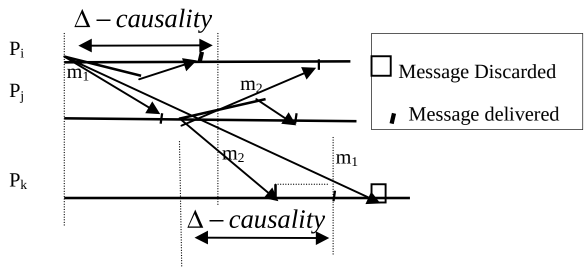

Figure 2.1. Δ *-causal ordering*

The Δ *-causality* ordering ensures that whenever a message arrives within its deadline, and it have not been delivered, it can be delayed until its delivery condition becomes true (figure 2.1). Thus, the core of this new abstraction consists then in defining the delivery condition DC(m) that is associated with each message. In such a way these messages are delivered as soon as possible.
Once all processes have synchronized their local clocks, and basing in the delivery condition presented in [4], each one of them has a variable called *current_time* that is continuously updated by the local clock, and manages the following two variables. The vector *sent* where $sent_i$[j] represents the knowledge of process p_i about sending time of the last message sent by p_i to p_j. The vector *del*, where del_i[j] represents the sending time of the last message sent by p_j to p_i and delivered to p_i.
Each message m piggybacks the following control information : VT_m is a copy of the vector «*sent*» of the sending process at m's sending time; and $send\_time_m$ is the time at which m has been sent (occurrence of the associated send event)
A message m is discarded if ($send\_time_m + \Delta > current\_time$).
By definition, all messages arriving at their destination within their deadline must be delivered as soon as possible. More formally, the delivery condition DC(m) of message m sent to p_i is expressed as :

$$\forall x: (VT_m[x] \leq del_y[x]) \vee (VT_m[x] < current\_time - \Delta)$$

All messages that causally precede m either have been delivered or their deadline have expired.

3. Group communication system

3.1. Group concept

We consider a group communication system G, as a complete graph composing of N processes p_1, .., p_N that communicate by exchanging messages through FIFO channels, and message transfer delay is arbitrary but finite. There is neither common shared memory nor global clock. The communication is made without degradation and without duplication or loss of messages. In other words the underlying network is reliable and asynchronous.
Cooperation between processes is structured with the group concept. Each process has a distinguish identity. The information of identity will be used for electing local servers, master server, and also synchronizing local clocks in the system.
The group communication system is built upon a hierarchical architecture of k-local groups, where k represents the number of groups in the system G. Each group is defined as a set of

processes, that forms a tree structure where the root is a member of that group, and is called the local server. We assume that the group communication system can detect process failures, and report them to the local server. Processes belonging to different local groups can exchange their messages only via their local servers. For that a virtual group is created as a collection of local servers in order to ensure the communication between all groups in the distributed system. Therefore, to guarantee the communication in the hierarchical structure, all data traffic must travel among local servers to be multicasted to all members of the group or be forwarded to the virtual group.

Definition 1. A group g_i is defined as a set of n_i processes $\{p_1, .., p_{ni}\}$ which possess distinct identities.

Definition 2. A group communication system G is a set of k-local groups noted : $G = \bigcup_{i=1}^{k} g_i$.

Definition 3. We define a virtual group g_v as a collection of local servers p_{si},(where $p_{si} \in (g_i \cap g_v)$, g_v: $g_v = \bigcup_{i=1}^{k} P_s$

Definition 4. g_i and g_v (represent respectively a local and a virtual group) are called overlapped groups if there exists a process p_s (playing the role of local server), that belongs to both groups : $g_x \cap g_v = \{P_s\}$.

3.2. Distributed election algorithm

We consider a network with N processes which want to communicate with one other. In direct connection architecture, each process builds connections with other processes in the network, i.e., all processes are fully connected. Since the total number of virtual connections and the architecture given in figure 3.1 can be reduced by appropriately grouping processes. We divide the N processes into k-local groups, such that each process is a member of only one local group, as illustrated in figure 3.2. Our grouping concept puts processes within the same neighbor set called local group to allow local server, of each local group, to buffer less media units without degradation of the QoS. In this way the continuous of data transfer will not be perturbed, and the delay jitter can be compensated.
Once, the local group is known, only one process is chosen by the distributed election algorithm as local server for that group, in order to ensure real-time traffic. All local servers are also members of another group called virtual group (figure 3.2), and one of them is selected as the Master server to be the global root of the group communication system.

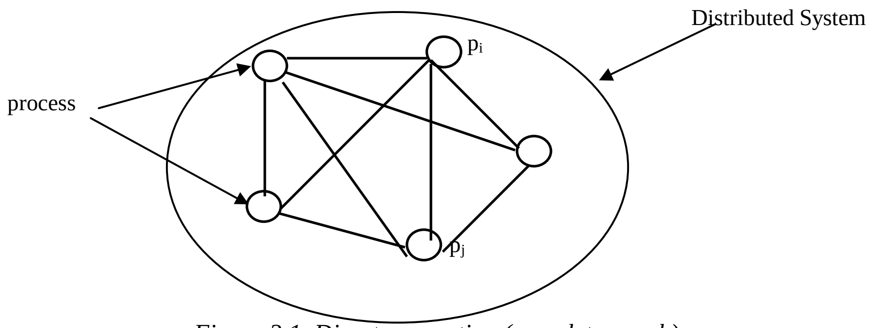

Figure 3.1. Direct connection (complete graph)

To define the so-called « neighborhood » we use some tests [12] in the first phase of distributed election algorithm. Those tests take place during the channel establishment.

i) *deterministic test*: this test consists of verifying that enough processing power is available in the process to accommodate the additional channel without impairing the guarantees given by others.

Since the packet processing time in each process is a constant parameter, the service time is the same for all channels connecting to that process. The service time is independent of the packet size because it is expressed in terms of packets.

Let :

- t_p (sec) represents the packet processing time of the process p, and
- $x_{\min,j}$ (sec) is the minimum inter-arrival time of packets to process p, and is computed on the maximum rate packet $X_{\max,j}$ (packet/sec) supported by the channel j, and is defined as follows :

$$x_{\min,j} = \frac{\mathbf{1}}{X_{\max,j}}$$

Thus, if the condition in the Equation (1) is satisfied, then the channel establishment proceeds. Otherwise, a rejection message is sent to the sender :

$$\sum_{j \text{ in all channels connecting to } p} \left({t_p}/{x_{\min,j}} \right) < 1$$

ii) *Delay bound test* is tested at every process of the channel. The delay bound is tested to ensure that no scheduler is saturated after a new channel is established. This test is performed by each process with their immediate neighbors to form its own group. Since the service time for each channel is the same as the packet processing time of the process, the delay-bound test at process p should satisfy the condition (2). If the condition is not satisfied, the channel is rejected.

$$t_p * m + t_p < \delta_{i,p} \qquad (2)$$

where :

$\delta_{i,p}$: is the delay bound of the channel i to the process p,

m : is the total number of channels already connected to p.

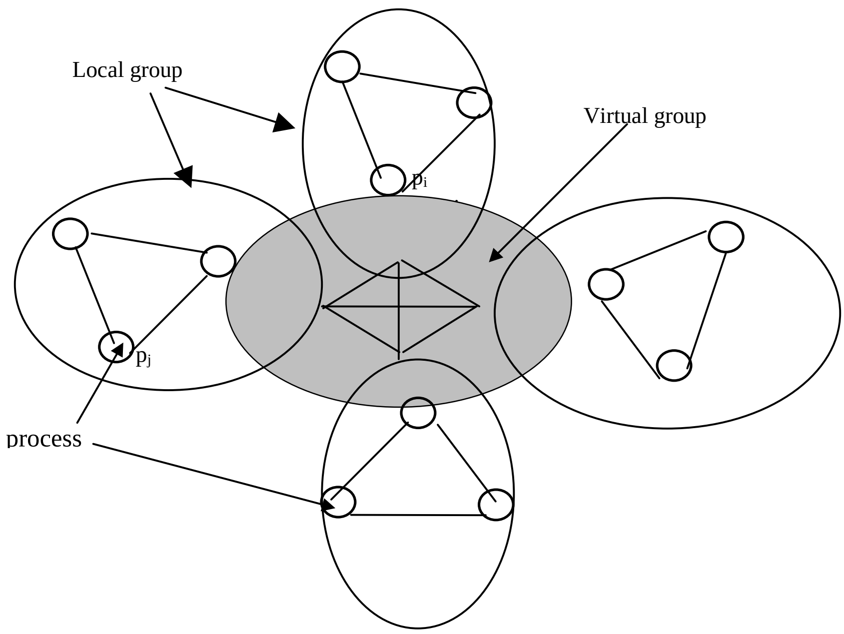

Figure 3.2 .Connection model for k- local groups

In order to ensure real-time data traffic, we synchronize users' local clocks. This synchronization allow us to compensate for local clock drift. We assume that each group has a local server. When each one receives a message from other groups, it broadcasts it to all processes of its group as soon as possible. Our scheme synchronizes the clocks of all users in the groups according to a time reference [9]. This value is used by all processes in the distributed system as their own local clock value. Note that each process adds a synchronization delay to the VMT value in order to have the same clock value, at a given time, with its Master server. The value of time is the value of master server clocks selected by distributed election algorithm. Our algorithm is divided in two phases :

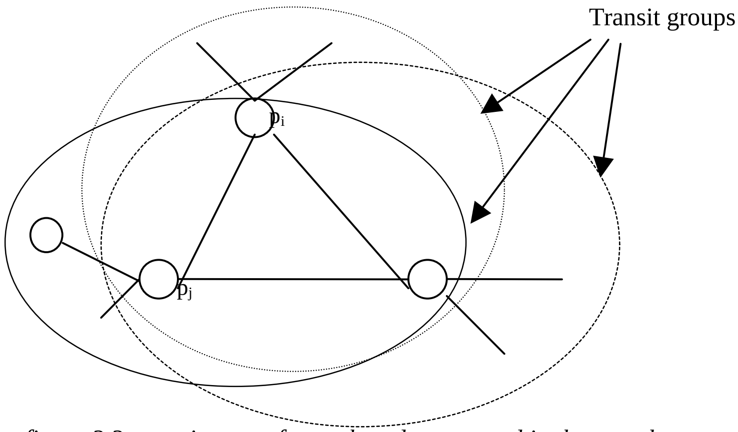

figure 3.3. transit group for each node presented in the complete graph

- In the first phase, all processes in the distributed system know the identities of their immediate neighbors set, and these identities are recorded in their lists in ascending order. In the first phase, each process performs the first and the second test to define its own group called the transit group (figure 3.3).

The following rules are necessary for the validity of the transit group :

- $\forall p_i : |g_{tri}| \geq 2$, *each transit group must be constituted of at last two processes*
- $\forall p_i , \forall p_j : \; p_j \in g_{tri} \Rightarrow p_i \in g_{trj}$

Once the transit groups have been defined, a mechanism must breaks the unnecessary connections to create the local group. Now, all processes have their own lists that are composed of their immediate neighbourhood set, however, many processes can belong to many transit groups. To solve this problem, we proceed as follows, each process p_i sends a Req_set(g_{tri}) message to inform the participants in its transit group of the contents of its own set g_{tri}. After recording all received sets, each process selects the maximum of processes existing simultaneously in all received sets. In the case of overlapping group, the process will belong to the group which has the lowest identity or the successor has the lowest identity if the lowest belongs to the overlapping group. So in this way a local group is specified by breaking the unnecessary links. Thus, the communication group system will be represented by k-local groups, where each process is belonging only to one local group. Once, the local group has been defined, the process generating the lowest identity is selected as local server. Once the local server p_s has been elected, it sends a «Establish(g_s)» message to confirm the connection and to build its hierarchical group model (figure 3.4). The main goal of breaking those unnecessary links is to ensure, efficiently, that the new successor of the local server belongs only to an one local group.

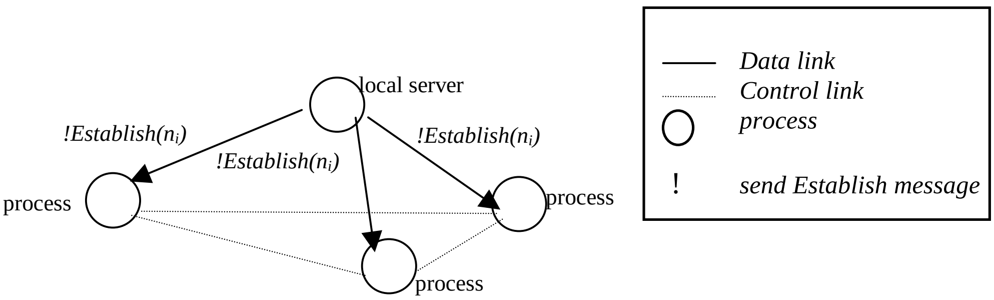

Figure 3.4. Hierarchical architecture in a local group

- In the second phase, when the local servers of the local groups have been selected, they constitute a virtual group. We suppose that the virtual group is a complete graph allowing distributed communication between the local servers. Let g_v this virtual group, then the process having the lowest identity is elected as the master server p_{mast} as illustrated in figure 3.5. Once the Master server has been elected, it sends a «Connect(VMT)» message to all local servers belonging to g_v, where VMT represents the Virtual Master Time value, of the master server p_{master}, that will be used by all server for synchronizing their local clocks. Therefore, the processes in all local groups takes the VMT value via their local servers as their own local clocks. Thus, all participants in groups have the same time value at a given time.
  The role of p_{mast} is to broadcast periodically his virtual local clock VMT to all members of g_v allowing then the local clocks synchronization.

Algorithm of process p_i**:**

let $g_{tr_i} \in G$ ***a transit group of process*** p_i ***such as, constructed by performing tests 1) and 2),***

Let $n_i = |g_{tri}|$ ***,***

$\forall p_j \in g_{tr_i} \rightarrow$ ***send Req_set(***g_{tri}***) to*** p_j***;***

$g_i = \emptyset$***;***

Repeat

<u>When receiving</u> Req_set(g) from p_j

$g_i = g_i \cap g$***;***

$n_i := n_i - 1$ ***;***

Until $n_i - 1 = 0$ ***;***

If $p_i = min(g_i)$ ***then*** $\forall p_j \in g_i \rightarrow$ ***send Establish(***g_i***) to*** p_j ***/**** $p_j \neq p_i$ ****/;***

/* The process which has the lowest identity value in each g is selected as local server for that local group */

<u>When receiving Establish(g) from</u> p_j

Mark p_j ***as my local server and g as my local group;***

Having a virtual hierarchical structure, the links of the network are divided into two types (figure 3.4 and 3.5) :

(i) data links are a part of the hierarchical structure, their role is to transfer multimedia traffic to their destinations ;

(ii) control links are all the rest, their role is to inform the other processes in groups in the case of a process or a link failure by transporting control information like the Master server's group content.

NB. The data transmission is accomplished in distributed manner between the servers of the virtual group. However, it is in hierarchical manner in the local group.

3.3. Local clock drift

Once the virtual group has been constructed and has elected the master server, a mechanism must be provided to compute the Δ-value. This value is the global communication delay in the group system that will represent the lifetime of media unit, after which that media unit can not be useful for the playing back.
We denote by d_{min} the smallest minimum communication delay between one local server and the nearest process belonging to its local group, and d_{max} the largest maximum communication delay between one local server and the farthest site in the system G. These variables represent the transit time, the time spent in their local servers, the collection delay and the delivery delay between each local server and the nearest respectively the farthest process. If τ_{s_i} is the arrival time of messages to a local server s_i, then the earliest and the latest time at which message from s_i is available to be played back by a process p_i is bounded by :

$$T_{min} = \tau_{s_i} + d_{min}.$$

$$T_{max} = \tau_{s_i} + d_{max}.$$

Thus, the lifetime Δ of multimedia data is bounded by : $T_{\min} \le \Delta \le T_{\max}$
Once all local clocks have been synchronized and the lifetime Δ of multimedia data have been defined, we must keep the synchronization among the communication exchange to insure the real-time relationships. If that constraint is relaxed, the clocks are no more than ε time units out of synchronization with each other. To solve that local clock drift without changing to the algorithm, it is suitable to update only the definition of Δ. For that, the selected Master Server sends periodically its VMT value to all members in the system, in order to keep clocks synchronized within 5-10 milliseconds [8].
If $\varepsilon = H_{new}-H_{old}$, Where H_{new} = VMT+ τ_i is the time value of a local clock according to the Master clock (VMT) and τ_i is the synchronization delay of the local clock of process p_i, is non-zero, then Δ is defined to be the time data in a message with the maximum clock drift : $\Delta_{new} = \Delta_{old} + \varepsilon$
This change does not have an influence upon the functionality and the validity of the protocol, because Δ defines only the lifetime of a message.

3.4. Architecture management

Once the hierarchical architecture is built, a mechanism must be provided to handle the entering and leaving of processes and local servers. We introduce a reconfiguration algorithm that solves the problem of processes entering and leaving the architecture. If a user belonging to a local group wants to leave the group, it is removed from the group set identity processes by all other processes in that local group via their local server. However, if a process p_{new} wants to join the group communication model, it multicasts a message request to its immediat neihgboors p_i, requiring the local server address and the group belonging to of each one of them. Since the election algorithm is distributed, each process has full knowledge of the grouping information. Thus, process p_i can give the addresse of its local server to p_{new}. After getting the addresses of the local servers, the new process p_{new}, , sends a connecting message to the local servers. When the local server receives this message, it performs the first, and the second tests, to ensure that it has enough processing power. If this new process p_{new} is accepted by the local server, it receives a acknowledgement. In the case of it receives more than one acknowledgement the new process p_{new}

makes connection with the local server which has the lowest identity. During this phase the new process negotiates the maximum end-to-end delays with each local server separately. Once, the connection with the local server is established all group membrs adds the new process to the group set identity processes.

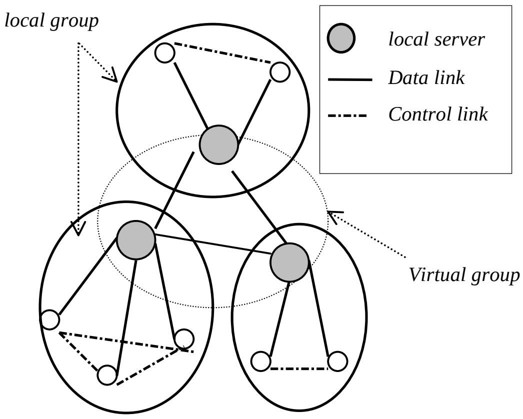

Figure 3.5. Hierarchical architecture in a distributed system

However, if a local server wants to leave the communication, then it informs all local group members, and all local servers. As each member has the same group set contents, each one of them makes a connection with the local server's successor. After the connections are established, processes can get messages through the old local server, or the new local server, but they send their messages to the new local server. In other hand the new local server informs the group members that they can receive its messages at a time stamp T, this time is the time at which the old local server sends all messages stored in its buffer.

4. Correctness

In this section, before proving the liveness and safety properties of the presented protocol, we demonstrate that each local group composing the communication system is represented by only one local server. These servers are known and are reached by all members of the group belonging to. In other hand, we prove that the group communication system K local groups represented by K local servers.

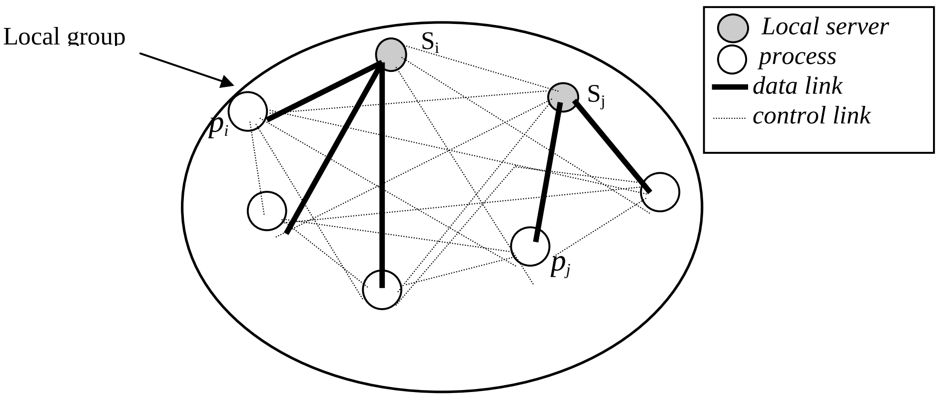

Figure 4.1. Sub-tree structures in a local group

4.1. Correctness of the hierarchical structure

Lemma 1 : *Each local group g_x is represented by one and only one local server, and all locally members know it.*

Proof :
Assume by way of contradiction, that there is two local servers p_{s_i} and p_{s_j} in a local group g_X, belonging to the group communication system G. Once a local server has been selected, it sends periodically its own group set and VMT value for updating the set contents at each process level to maintain the successor of the local server, and in other hand for compensating for local clock drift to their members in their trees. If the system selects two local server in the same local group, as illustrated in figure 4.1, so both servers will build their own tree, and all group members will receive at a given time the server lists contents to manage the successor of the local group. Note that in the local group, the control links between processes belonging to both sub-tree structures are still existing. If both servers send their lists, they will be received by all members in the group g_X, and at each process level, we have the following relationships : $p_{s_i} < p_{s_j}$ indicating that p_{s_i} is the single local server of the group and $p_{s_i} < p_{s_j}$ indicating that p_{s_j} is the single local server of the group. This case is never happen, because all processes in the group are known by the other members and possess distinct identities. So, the selected local server is the single process which has the lowest identity in the group set among the existing processes that compose the local group.
Consequently, the local group will be represented only by one single local server. Hence, a contradiction to the assumption that each local group composing the distributed system G, can be represented by several local servers in the same time.

Theorem 1 : *The communication system architecture is distributed and hierarchical.*

Proof : *Each local group possess one and only one local server by lemma 1. The communication between a local group member and another is accomplished via its local server. So, the communication is hierarchical in a local group. The communication between groups is assumed by local servers by broadcast in distributed manner. So, the system communication architecture is distributed and hierarchical.*

4.2. Safety

We prove that in this architecture the delivery condition DC(m) is invariant.

Lemma 2 : *In a local group, the protocol respects the real-time causal delivery.*

Proof :
Let $p_i, p_j \in g_X$ be two processes belonging to a local server g_x. We assume that $send_i(m) \rightarrow send_i(m')$, and we want prove that both messages are delivered before their deadlines to processes p_j : $del_j(m) \rightarrow del_j(m')$. When the process p_i sends $send_i(m) \rightarrow send_i(m')$, it timestamps the messages by $VT_m[i]$ respectively $VT_{m'}[i]$. It means that the message m is causally delivered before m' by all participants in the group. Thus, in the local server $VT_m[i] < VT_{m'}[i]$ indicating that m' will be multicasted after the message m. Consequently, the receiver will deliver the message m before m' following the Condition Delivery. However, if processes p_j receive m' before m, and its delivery condition becomes true, the receivers consider that the message m is

lost, and deliver m'. The message m will be discarded if p_j receive it after delivering m'. If that is happened, it means that the process p_i has sent the message m' by another link (using direct connection, where the communication delay is very low) than the specified data one in the architecture. This communication type is available only in a complete graph case, where the sender and receiver pair can exchange their messages via a direct connection. But, this is never happen in the present architecture, because all messages travel through the same local server existing in the local group (lemma 1), and that server respects the causal delivery rule. Therefore, the hierarchical structure ensures the real-time causal delivery in a local group. Thus if an event $send_i(m) \rightarrow send_i(m')$ is produced then the result is $del_j(m) \rightarrow del_j(m')$, therefore in each local group the Delivery Condition is invariant.

Theorem 2 : *The presented architecture respects real-time causal delivery.*

Proof :
Let $p_i \in g_x$ and $p_j \in g_y$ be two processes. We assume that $send_i(m) \rightarrow send_j(m')$, and we want prove that both messages are delivered before their deadlines to the other processes $p_k \in g_z$ but $del_k(m) \rightarrow del_k(m')$.
Two cases are possible :

- $g_x = g_y$, this cannot happen because the local server sends message m before m' : $VT_m[i]<VT_{m'}[i]$ (i=j) (lemma 3);
- if $g_x \neq g_y$ we have the following relation :
  $send_i(m) \rightarrow send_j(m')$, it means that message m is causally delivered at the local server p_{s_y} of g_y group, by the time it forwards m' to the other groups. Thus, in each local server $VT_m[i] < VT_{m'}[j]$ indicating that m' follows causally m (lemma 4).
  Let p_{s_z} be the server group of g_z where $p_{s_z} \in (g_z \cap g)$. Since at each local server, m' has a timestamp value that follows m, meaning that in virtual group $\boldsymbol{g_v}$ the message m will be delivered after m'. Therefore, p_{s_z} must deliver m before m' $del_k(m) \rightarrow del_k(m')$. Thus the process $p_k \in g_z$ will delivers the message m before m' : $del_k(m) \rightarrow del_k(m')$. So, the Delivery condition is invariant.

4.3. Liveness

We prove that in spite of the presence of failure, the presented architecture is able to deliver messages by respecting theirs deadlines.

Lemma 4 : *For a local group, all sent messages will be received, and delivered by respecting the Delivery Condition .*

Proof :
When a process sends a message to their participants in the local group. This message is timestamped by VT_m and broadcasted to the members via their local server. Since the hierarchical structure in the local group uses a reliable point-to-point communication, and the same connecting channel to reach the destination via the server (lemma1), all messages must be received and delivered when their delivery conditions become true. However, if a sent message is not received by a destination or is received after that its condition delivery becomes true, then this message is discarded. But, this case is never happen, because the Δ value is computed for ensuring efficiently the delivery of messages in their lifetime, mainly in the local group. In other hand, the communication is reliable, and if a message is not received in its lifetime validity, we

can assume that the process sending this message has used another communication channel than the specified one in the presented topological structure(control link). However, the proposed architecture is a hierarchical one, not a complete graph, where each process can reach the destination by choosing a link among several direct connections to reach the destination. Therefore, all processes send their messages to the server via the same and the specified point-to-point data link to be multicasted. Thus, the presented protocol ensures the delivery condition for each received message in the local group.

Lemma 5 : *For the virtual group the protocol guarantees that all sent message will be received and delivered by respecting the Delivery Condition.*

Proof :
When the local server receives the message from a process to forward it among the virtual group. This message is broadcasted to the other local groups. As the topological structure in the virtual group is defined as a complete graph, then each server can reach another server via the same point to point data link that is used for sending multimedia data. Each group receives the message in its deadline and can delivers it if its delivery condition becomes true. However, if a sent message is not received by a local group or is received after that its condition delivery becomes true, then this message is discarded. This case is never happen, because the communication in the virtual group is reliable point-to-point one, and respect the FIFO order.
Therefore, in the virtual group the delivery condition is respected.

Theorem 3 : *Every sent message, is received by a not crash process before its deadline, is delivered to all its recipients by respecting the Delivery Condition DC(m).*

Proof :
Let g_x and g_y two local groups, and g_v the virtual group. We can distinguish two cases :
- $g_x = g_y$, in this case we are in the same local group, so we can use Lemma 4;
- $g_x \neq g_y$ by applying lemma 5. In both cases we obtain that the topological structure is a hierarchical one, and guarantees the delivery of the received messages in their lifetime.

5. State Space graph

In this section, we present the specification and the validation of the hierarchical architecture protocol. We use the Coloured Petri Nets (CPN) jointly with a functional language CPN-ML (extension of Standard-ML) [13,14]. These High-Level Nets are now in widespread use for many practical purposes. The main reason for the great success of this kind of nets is the fact that it has a graphical representation and well-defined semantics allowing formal analysis of large systems.
CPN combine the power of formal inscriptions(i.e. mathematical expressions) with the clarity of graphical models to provide concise representation of complex systems.

5.1. Description of CPN

We give here the informal introduction of CPN. Readers should refer to [11] for formal definition. A CPN consists of a set of **places**, representing the possible states of a system and indicated by ellipses, each place has an association **type** determining the kind of data that the place may contain. A set of **transitions** representing the actions of a system and indicated by rectangles. A set of directed arrows, called **arcs** connect places and transition in the nets.
Each place must include the following inscription :

- **Colour Set** definitions specify the types of tokens in the model. Each place in CPN has associated colour set which contains the tokens allowed to occupy it. In Design/CPN a colour sets to be used in the net are defined in declaration node.
- **Marking** specify the multi-set of tokens occupying each place, represented as an algebraic sum.

Each transition can include the following inscriptions :

- **Guard** is a boolean expression with colours as terms. It restricts the allowable bindings if the variables occurring in the arc expression.
- **Code region** calculates the output inscription and they can be used for graphical animations.

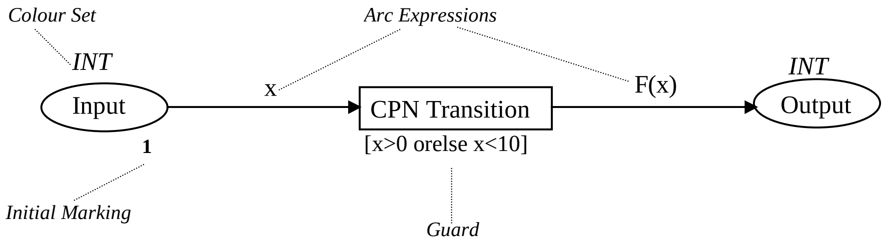

Figure 5.1. Coloured Petri net model

Arcs must have expressions

- **Arc expressions** characterize the multi-set of tokens that will move along an arc when the association transition occurs. Arc expressions contains variables, predefined or user defined functions and boolean selectors , along with the usual operators. For any occurrence of a transition, the variables in the associated arc expressions are bound to particular values. A CPN model can be represented as illustrated in figure 5.1.

In contrast to most specification languages, Petri nets are state and action oriented at the same time - providing an explicit description of both the states and the actions. This means that the modeller can determine freely whether - at a given moment of time - he wants to concentrate on states or on actions

All the CPN inscriptions are declared in CPN-ML. CPN-ML is an extension of a functional programming language called «Standard ML». It allows to declare colour sets that are primitive ML types, such as boolean, integer, real, and string colour sets may also be constructed from non-primitive ML types. From theses types we can construct new colour sets by forming cartesian products, records, union, lists and subtypes. In addition, ML supports abstracts data types allowing the listing of the details of colour sets.

The main feature of the CPN is its hierarchical CP-nets. The basic idea behind hierarchical CP-nets is to allow the modeller to construct a large model by using a number of small CP-nets which are related to each other in a well-defined way. In a hierarchical CP-nets it is possible to relate a transition (and its surrounding arcs and places) to a separate CP-net providing a more precise and detailed description of the activity represented by the transition. Each of the transitions is marked with an HS-tag (hierarchy + substitution) indicating that it is a **substitution transition**. The dashed boxes next to the HS-tags are called **hierarchy inscriptions** and they define the details of the substitutions.

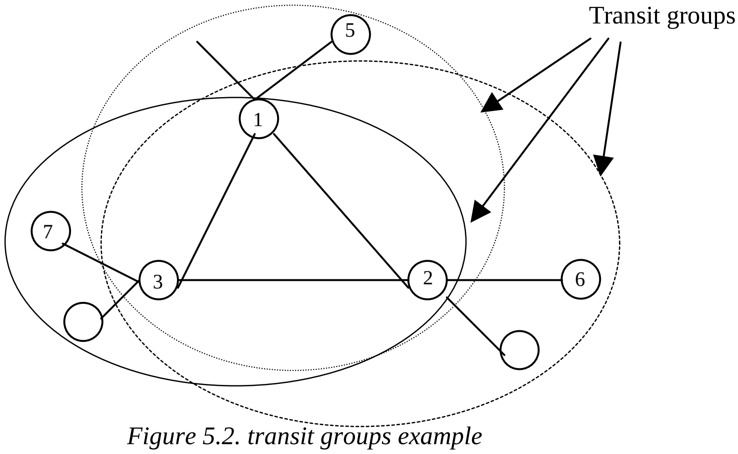

Figure 5.2. transit groups example

5.2. Description of the model

As each process in the distributed system knows the identities of its immediate neighbors set, it performs the first and second test to create its own transit group, as illustrated in figure 5.2. The identities are recorded in the group members sets in ascending order. In our model, the place *Transit_gr* represents a multi-set of transit groups as the result of the both tests. The initial marking in this place is a product of colours INTxLIST (representing the identity of a process, and its group members set composed of their immediate neighbors set) as illustrated in figure 5.3, the rest of places in our CPN model have a colour set LIST. Once the transit groups have been defined, a mechanism must breaks the unnecessary connections to create the local group, and elects a local server for each local group.
The CPN model presented in figure 5.3. is a hierarchical one, the first line of each hierarchy inscription specifies the **subpage,** i.e., the CP-net that contains the detailed description of the activity represented by the corresponding substitution transition. In our example, we see three substitution transition (HS in the figure 5.3) denoted respectively *Local_gr_election*, *Virtual_gr*, and *Master_serv_election.* Also, all of the net inscriptions must be declared in the Declaration nodes as illustrated in figure 5.4.

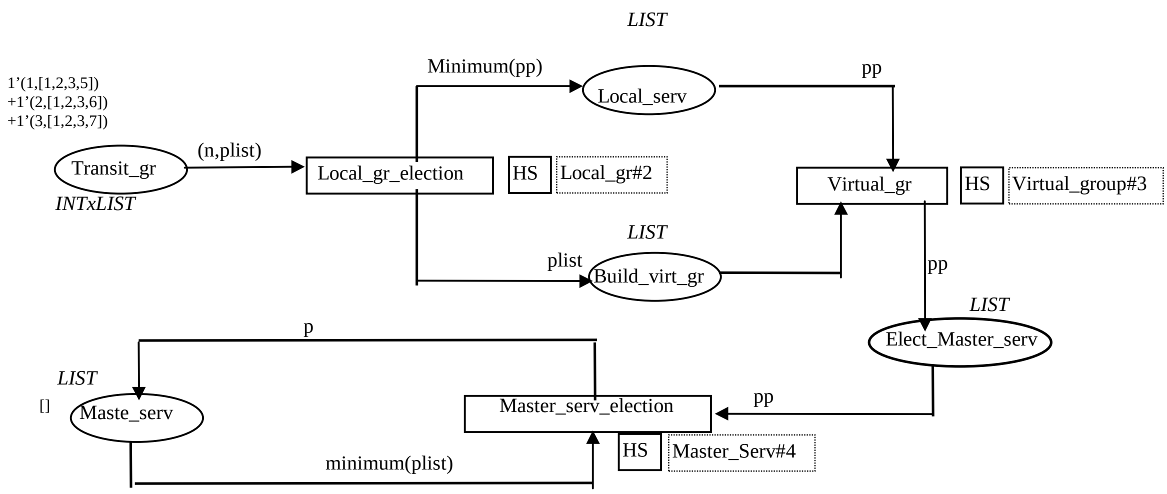

Figure 5.3. Hierarchical CPN for electing a Master server

All processes have their own lists that are composed of their immediate neighborhood set, however, many processes can belong to many transit groups. To solve this problem, each process belonging to each transit group send a message to its group members containing its identity and its group members such as (1,[1,2,3,5]), figure 5.5. As the message transfer delay is finite, each process (in our model is represented by the place *Process_i* in figure 5.5) will receive all transmitted messages.

To create a local group, the processes select the maximum of processes existing simultaneously from the received group sets. In our example, all processes selected a local group representing the following process [1,2,3] in the place *local_gr*. The other processes are sent to the place *others* (figure 5.5). So, in this way a local group is specified. Thus, the communication group system will be represented by k-local groups, where each process is belonging to only one local group.

```
color INT = int ;
color LIST = list INT ;
color INTxLIST = product INT * LIST ; /* represents the group members set */

fun member(x,[]) = false
        | member(x,(y : :l)) = (x=y) orlse member(x,l) ;

fun  nointer([],ys) = []
        | nointer(x : :xs,ys) = if member(x,ys) then noiter(xs,ys) else x : :nointer(xs,ys) ;

fun inter([], ys) = []
        | inter(x : :xs,ys) = if member(x,ys) then x : :inter(xs,ys) else inter(xs,ys) ;

fun nemem(x,xs) = if member(x,xs) then xs else x : :xs ;
fun meme([],ys) = ys
        | meme(x : :xs,ys) = newmem(x,meme(xs,ys)) ;
fun insert(i,[]) =[i]
        | insert(i,y : :l) = if i<y then i : :y : :l else y : :insert(i,l) ;
fun sort [] =[]
        | sort [x] =[x]
        | sort(h : :t) = insert(h,sort t) ;
fun minimumm [] = []
        | minimum [x] = [x]
        | minimum(h : :t) = h : :[] ;

var n,k : INT
var plist, pp, p : LIST ;
```

Figure 5.4. Global Declaration

Once, the local group is defined, the process generating the lowest identity is selected as local server in this model the process 1 is a local server (place local_serv in figure 5.5).

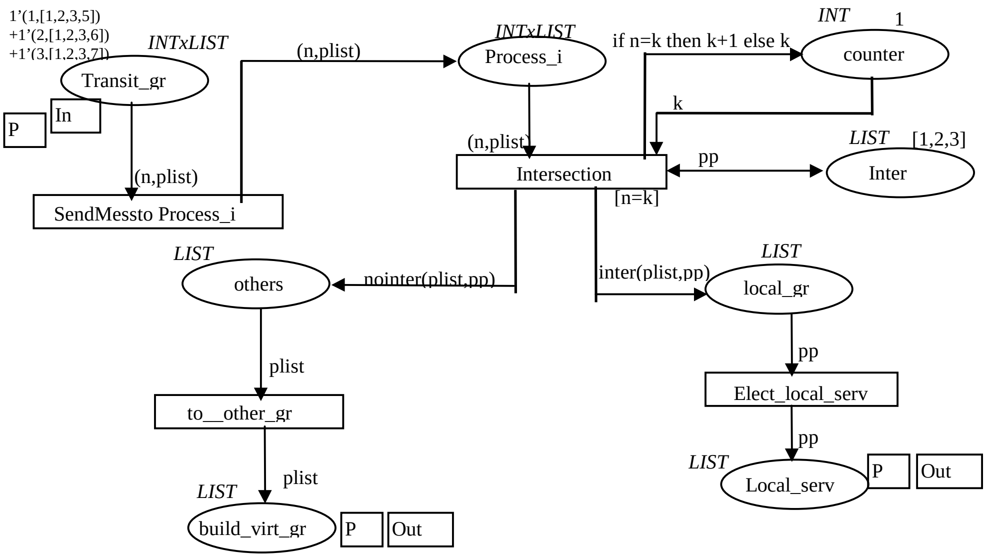

Figure 5.5 Subpage of Local_gr substitution transition

In our model, the place *others* contains a multi-set marking that represents the identities of the following processes 5, 6, and 7. We assume that each one of them is, respectively, a local server in an other local group.

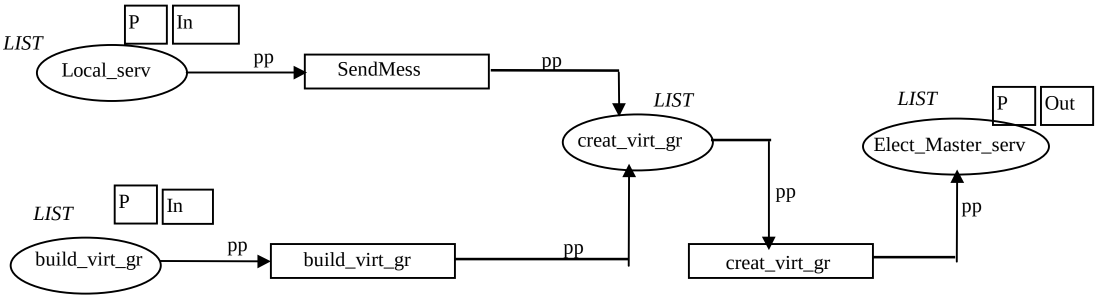

Figure 5.6 Subpage of Virtual_group substitution transition

Once the local servers in each local group have been selected, they create a virtual group, where they can exchange their messages (place *creat_vir_gr* in figure 5.6). In the virtual group, each local server sorts, in the place *Elect_Master* in figure 5.7, the received identity values in ascending order. The server generating the lowest value is selected as Master server of that group communication system in our case the place *Master_srv* possesses the process 1 as Master server (figure 5.7).

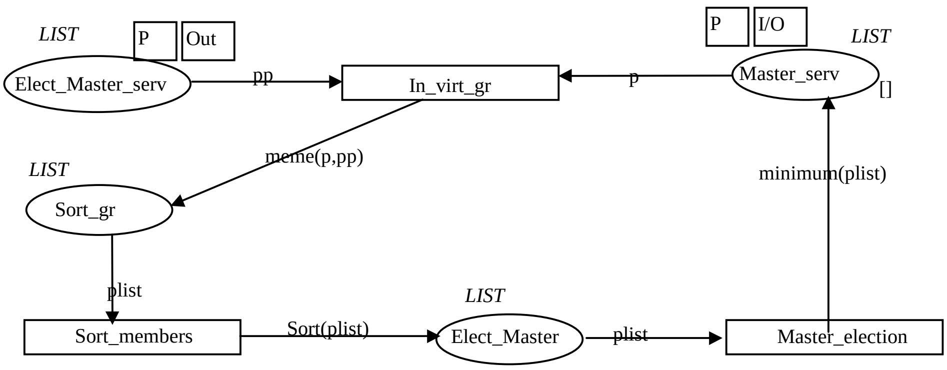

Figure 5.7. Subpage of Master_serv_election substitution transition

5.3. State Space Analysis

we simulate this protocol to obtain the state space graph. The basic idea behind a state space is to construct a graph which has a node for each reachable marking and arc for each occurring binding element. State spaces are also called occurrence graphs or reachability trees. The first of these names reflects the fact a state space contains all the possible occurrence sequences, while the two latter names reflect that the state space contains all reachable markings.
For investigating the state space we ask the state space tool to make a state space report providing some key information about the behaviour of the CP-net. This report is produced totally automatic. It contains a lot of highly useful information about the behaviour of the CPN model. The state space report can be presented in three parts.

Statistics	
Occurrence Graph	
	Nodes: 1786
	Arcs: 5145
	Secs: 5
	Status: Full
Scc Graph	
	Nodes: 1786
	Arcs: 5145
	Secs: 3

Figure 5.8 Size of occurrence graph and SCC- graph

The first part looks as shown in figure 5.8. It contains **statistical** information about the size of the state space. We see that the state space has 1786 nodes and 5145 arcs. We have calculated the *full* state space, and this took only 5 seconds. The statistical part also contains information about the **strongly connected components** SCC-graph of the space. The SCC is the maximal sub-graph in which it is possible to find a path from any node to any other node. Strongly connected components are very useful to determine certain kinds of behavioural properties, and they can be calculated by standard algorithms (which are linear in time and space). From figure 5.8, we see that there are 1786 strongly connected components and 5145 arcs that start in one component and end in another.
The second part of the state space report contains information about the boundedness properties. The figure 5.9 shows that the upper and lower integer bounds, i.e. the maximal and

Boundedness Properties :

Best Integers Bounds	***Upper***	***Lower***
Distributed_System'Elect_Master_srv	4	0
Distributed_System'Master_srv	1	0
Distributed_System'Transit_gr	3	0
Distributed_System'biuld_virt_gr	1	0
Distributed_System'local_srv		1
Local_group'Inter	1	0
Local_group'Process_i	3	0
Local_group'Transit_gr	3	0
Local_group'biuld_virt_gr	3	1
Local_group'counter	1	0
Local_group'local_gr	3	0
Local_group'local_srv	1	0
Local_group'others	3	0
Master_Server'Elect_Master_srv	4	0
Master_Server'Elect_Mster	1	0
Master_Server'Master_srv	1	0
Master_Server'Sort_gr	1	
Virtual_group'Elect_Master_srv	4	0
Virtual_group'biuld_virt_gr	3	0
Virtual_group'creat_vir_gr	4	0
Virtual_group'local_srv	1	0

Figure 5.9 Upper and lower Integer bounds

minimal number of tokens which the individual places may have. We see that many places has between zero and three tokens. We also see that other places always have exactly one or zero token. This reassures us that the system is working as expected.

Node 1786 :

Local_group'Inter :	1'[1,2,3]
Local_group'Process_i :	empty
Local_group'Transit_gr :	empty
Local_group'biuld_virt_gr :	empty
Local_group'counter :	1'4
Local_group'local_gr :	empty
Local_group'local_srv :	empty
Local_group'others :	empty
Distributed_System'Elect_Master_srv :	1'[1]
Distributed_System'Master_srv :	empty
Distributed_System'Transit_gr :	empty
Distributed_System'biuld_virt_gr :	empty
Distributed_System'local_srv :	empty
Master_Server'Elect_Master_srv :	1'[1]
Master_Server'Elect_Mster :	empty
Master_Server'Master_srv :	empty
Master_Server'Sort_gr :	empty
Virtual_group'Elect_Master_srv :	empty
Virtual_group'biuld_virt_gr :	empty
Virtual_group'creat_vir_gr :	empty
Virtual_group'local_srv :	empty

Figure 5.10. Home and Dead marking

The third part of the state space report is shown in figure 5.11. it provides information about home and liveness properties. A home marking is a marking which is reachable from all reachable markings, i.e., a marking which always can be reached independently of what has happened up to know. We see that the protocol has a single home marking M_{1786} figure 5.10. A dead marking is a marking with no enabled transitions. We see that the protocol has a single dead marking, and that the dead marking is identical to the home marking. Hence, let us take a closer look at this marking. To do this we ask the state space tool to draw the node 1786 (see figure 5.10). From this we see that the marking M_{1786} corresponds to the state where all four request messages have been successfully received and each process has performed the lists intersection to select its own local group memberships. The fact that M_{1786} is a home marking tells us that the protocol has the nice property that it never can reach a state from which it is impossible to terminate with the correct result.

From liveness properties (figure 5.11), we also see that there are no dead transitions. This means that each transition is enabled in at least one reachable marking. We also see that there are no live transitions. A live transition is a transition that always, no matter what happens, can become enabled once more. When there are dead markings, there cannot be any live transitions. At first glance, one might think that the existence of dead markings would prevent the existence of home markings. However, by definition, a marking is always reachable from itself, and hence a dead marking may be a home marking. This is the case for the protocol system.

Liveness Properties

Home Markings:	[1786]
Liveness Properties	
Dead Markings:	[1786]
Dead Transitions Instances:	None
Live Transitions Instances:	None

Figure 5.11. Home properties

6. Conclusion

In this paper we developed a hierarchical architecture composed of a dynamic grouping processes in k-local groups. This architecture supports Δ-causal ordering. The hierarchical architecture is established with a distributed election algorithm. The choice of the hierarchical architecture allow us to minimise the amount of exchanging messages for synchronizing easily the local clocks, and reduce the amount of control messages for maintaining the temporal relationships in order to reduce the local clocks drift.

Since the number of processes may change within the lifetime of the communication group, we introduce an architecture management that allows processes to join or leave the communication group.

We validate this model by using High-level Petri nets. The simulation result shows that there is no blocking message, nor unspecified reception message. Therefore, all studied properties are guaranteed.

Our perspective is to develop a new multimedia synchronization implemented on this dynamic hierarchical architecture and evaluate the performance by comparing the buffer size with that of a direct connection architecture.

References

[1] A. Abouaissa, A. Benslimane, and M. Naïmi, "A Group Communication Model for Distributed Real-time Causal Delivery", Proc. of the 7th IEEE Inte. conf. On Computer Communications and Networks pp :918-925, October1998.

[2] F. Adelstein, M. Singhal, "Real-Time Causal message ordering in multimedia sysytems", Proc. of the 15th IEEE Int. conf. On Distributed Computing Systems, pp : 36-43, 1995.

[3] L. Amanton, M. Naïmi, "Total and causal Ordering in Distributed Computing", Journal of Computing and Information, Vol. 2, N° 1, pp. 308-321, 1996.

[4] R. Baldoni, A. Mostefaoui and M. Raynal, "Causal Delivery of messages With Real-Time Data in Unreliable Network", Real-Time Systems vol.10, No.3, pp : 1-18, 1996.

[5] R. Baldoni, A. Friedman and R.V. Renesse, "The Hierarchical Daisy Architecture for Causal Delivery", September 1996.

[6] K. Birman and T. Joseph, "Reliable Communication in Presence of Failure", ACM Trans. on Computer Systems 5(1) :47-76, February 1987.

[7] K. Birman, A. Schiper, and P. Stephenson, "Lightweight Causal and Atomic Group Multicast", ACM Trans. on Computer Systems, 9(3) :272-314, August 1991.

[8] D. Ferrari, "Clients requirements for Real-Time Communication Services" IEEE Comm. Magazine, pp : 65-72, 1994.

[9] R. Gusella, S. Zatti, "The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkley UNIX 4-3 BSD", IEEE Trans. on Software Engineering, 15(7) :847-853, july 1989.

[10] S. Greonenberg, and D. Marwood, "Real-Time Groupware as a Distributed Systems : Concurrency Control and its Effect on the Interface", In Proc. of ACM Conf. on Cooperative Support for Cooperative Work, ACM press pp : 207-217, 1994.

[11] K. Jensen, "Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use", Vol.1 : Basic concepts, 1992. Vol.2 : Analysis Methods, 1994. Vol.3 : Practical Use, 1997.

[12] S. Kadur, F. Golshani and B. Millard, "Delay-jitter control in multimedia applications", Multimedia systems, Springer-Verlag, vol. 4, N° 1, pp.30-39, 1996.

[13] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed Systems", Communication of the ACM, vol. 21, No. 7, pp : 558-565, July 1978.

[14] D. Mills, "Internet Time Synchronization : The Network Time Protocol", IEEE Trans. on Comm. 39(10) :1482-1493, October 1991.

[15] R. Milner, R. Harper, M. Tofte, 'The definition od Standard ML", MIT Press, 1990.

[16] R. Milner, M. Tofte, "Commentary on standard ML" MIT Press, 1991.

[17] S. Mullender, "Distributed Systems", ADDISON-WESLEY 1995.

[18] R. Prakash, M. Raynal, and A. Singhal, "A Efficient Causal Ordering Algorithm for mobile Computing Environnement", Proc. of the 16th IEEE Int. Conf. on Distributed Computing Systems, Hong-Kong, june 1996.

[19] R. Rodrigues and P. Verissimo, "Causal Separators and Topological Timestamping in Large scale systems", Proc. of the 15th International Conference on Distributed Systems May 1995
[20] P. V. Rangan, H. M. Vin and S. Ramanthan, "Communication Architectures and Algorithms for Mixing in Multimedia Conferences" IEEE/ACM Transactions on Networking, vol. 1, No. 1, pp. 20-30, Feb. 1993.
[21] R. Schwarz and F. Mattern, "Detecting Causal Relations in Distributed Computing : in Search of the Holy Grail", Distributed Computing, 7(3) :149-174, 1994.
[22] R. Yavatkar : "A Protocol for Coordination and Temporal Synchronization in Multimedia Collaborative Applications", Proc. of the 12th IEEE Inter. Conf. on Distributed Computing Systems, IEEE press, 606-613.
[23] W. Yen and I.F. Akyildiz, "The hierarchical architecture for buffer managing in integrated services networks", Multimedia Systems, vol. 4, N°. 3, pp :131-139, 1996.

Authors' biographies :

Abderrahim BENSLIMANE is Associate Professor of computer science and Engineering at the Technical University of Belfort, France. He received his PhD in Computer Science from the Franche-Comte University of Besançon in 1993.
His research and teaching interests are in Specification and Verification of Communication Protocols, Computer and High speed Networks as ATM, Distributed Multimedia and languages Programming.

Abdelhafid ABOUAISSA received his B.S. degree from Technical university of Wroclaw, Poland, in 1995, and M.S. degree from Franche-Comte University of Besançon, France, in 1996. He is currently a Ph.D Student at Technical University of Belfort, France, where his interests include Multimedia Synchronization, group communication systems, ATM, QoS management.