# Order notions and atomic multicast in distributed systems : a short survey

Michel Raynal

INRIA

UNITE DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.(1)39635511

# Rapports de Recherche

**N° 1197**

***Programme 3***
*Réseaux et Systèmes Répartis*

# ORDER NOTIONS AND ATOMIC MULTICAST IN DISTRIBUTED SYSTEMS : A SHORT SURVEY

**Michel RAYNAL**

**Mars 1990**

★RR-1197★

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

# Order Notions and Atomic Multicast in distributed Systems : A short survey

Michel RAYNAL
IRISA
Campus de Beaulieu
F-35042 Rennes Cédex

8 mars 1990
Publication Interne n° 524 - 18 Pages

**Abstract**

Understanding distributed systems requires a fine understanding of order relations on the events their executions produce. The first part presents two mechanisms that capture two kinds of relevant orders (Lamport's clocks and Fidge-Mattern's clocks). The second part shows how this kind of mechanisms can be used to implement different multicasts in distributed system. this report is essentially a short survey.

## Un aperçu des notions d'ordre et de diffusions dans les systèmes répartis

**Résumé**

La compréhension de l'ordre sur les événements issus de l'exécution d'un système réparti est un élément essentiel pour maitriser ces sytèmes. Après avoir rappelé les 2 types d'ordres (total et partiel) qu'il est possible de capter lors d'une exécution et montré des mécanismes qui en rendent compte, on étudie leurs liaisons avec les protocoles de diffusion. Cet article est essentiellement un survey.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

# Abstract

Executions of distributed applications or distributed systems can be characterized as sets of events structured by a partial order. In such a partial order, events local to a site are totally ordered ; and for each message its sending event precedes its receiving event. This order, put forward by Lamport in 1978 and called precedence order, is a fundamental notion to design or to analyze many distributed control algorithms and consequently is a masterpiece in the design of distributed systems architectures or of distributed kernels.

The first part of the paper examines in depth different logical clocks mechanisms allowing to capture this order in part or in whole. So if Lamport's logical clocks allow to build a total order consistent with this partial order they cannot afford to decide if two events are logically independent. Such an independence knowledge can be essential in order to realize some distributed debugging or distributed observation. Clock vectors, proposed simultaneously by Fidge and by Mattern in 1988, allow to characterize dependence and independence of events. A study of their properties come with the presentation of these clock mechanisms.

The second part of the paper gives a survey of some high level communication primitives based on this order (causal ordering, multicast ordering, group ordering, etc). We show their usefulness in control algorithms and give implementation details ; they rest on clock mechanisms (or variations) presented in the first part. These primitives can be classified into two categories : first ones ask for one exchange phase only and others ones ask for at least two exchange phases (these are some kind of 2PL protocols). Some of these primitives have been proposed and realized in some systems (ISIS for example); some have been defined and are being implemented. In some sense they constitue a communication kernel for distributed systems, allowing to design applications without taking into account irrelevant aspect not mastered by low level communication primitives ; actually they hide some part of the undesired non- determinism inherent within distributed systems.

The understanding of precedence relations between events and the definition of high level communication primitives are two main trumps to master architecture as well as software engineering of distributed systems and distributed kernels. This paper is essentially a survey of these two points.

# 1 Introduction

The design of distributed systems kernels is of the highest importance in order to master the architecture and the operation of these systems. Though prototypes of distributed systems do exist a question has not been truly answered : what is a kernel for a distributed system and what are its primitives ?

Answering these questions requires firstly a better understanding of what is the execution of distributed applications and secondly what are the communication and synchronization tools they need.

This paper tries to give some elements to answer these questions ; it constitutes a short survey of these two points. The first part (§2) studies differents mechanisms allowing to timestamp events produced by distributed executions ; their relative properties are analyzed in depth and some of their possible uses are exhibited. The second part (§3) is interested in communication primitives which could be offered by a distributed kernel ; a list and an assessment of such primitives are given. As one can see this paper is essentially a survey of timestamping mechanisms and of communication primitives whose aim is to build an abstract distributed machine easier to use than the underlying one.

# 2 Get some order in a distributed system

## 2.1 Run and order

A distributed system is made of n sites $S_1, \ldots, S_n$ which communicate by exchanging messages along channels. Without loss of generality we suppose there is one and only one process per site, and we use the two terms equivalently. We are interested here in asynchronous systems that is to say in systems where sites run at their own speed, and where transfer delays of messages are a priori not bounded. Any execution of an application on such a system is characterized by 2 fundamental sequential constraints :

i) events occuring on some site are totally ordered. (An event is the sending of a message, the reception of a message or an internal event not implying a channel)

ii) if we consider some message $m$ : the sending event precedes its receiving event. Consequently all the events preceding the sending precede all the ones following its reception.

These two constraints, put forward by Lamport [L78], structure the set of events generated by some run in a partial order "$\rightarrow$" called causal precedence order. Now we present two mechanisms which allow to take into account this order in part or in whole, and consequently to exploit it to solve particular distributed control problems.

## 2.2 Obtaining a total order

In a famous paper [L78] Lamport proposed a simple mechanism which associates a timestamp to each event with the following property : the timestamps are totally ordered and this total order is consistent with the partial order on events. A logical clock $h_i$ initalized to 0, is associated to each site $S_i$. Between two events local to $S_i$, $h_i$ in incremented by 1. Every message $m$ piggybacks its logical sending time $m, h$. Upon receiving a message $m, h$ the receiving site executes the following updating protocol :

$$h_i := max(h_i, h) + 1$$

and considers this new clock value as the message receiving time.

If we consider two events $a$ and $b$, timestamped respectively $h$ and $k$ and linked by one of the sequentiality constraints i) or ii) in the order $a \rightarrow b$ for example, then we have $h < k$. If we have no information concerning the precedence order between the 2 events $a$ and $b$ and if $h < k$ we can conclude either $a \rightarrow b$ or $a$ and $b$ are independents (one does not belong to the causes of the other). In others words there is no means to conclude if these two events are or not dependent. But it is very easy to build a total order $\Rightarrow$ consistent with the partial order $\rightarrow$. To this end each event $a$ is timestamped with a pair $(h, i)$ where $h$ is the logical clock value of the event and $i$ is the identity of the site where $a$ occurred. All the sites have distinct and totally ordered identities. So if $a$ and $b$ are timestamped respectively $(h, i)$ and $(k, j)$ the total order $\Rightarrow$ is defined by :

$$(a \Rightarrow b) \Longleftrightarrow (h < k \text{ or } (h = k \text{ and } i \neq j))$$

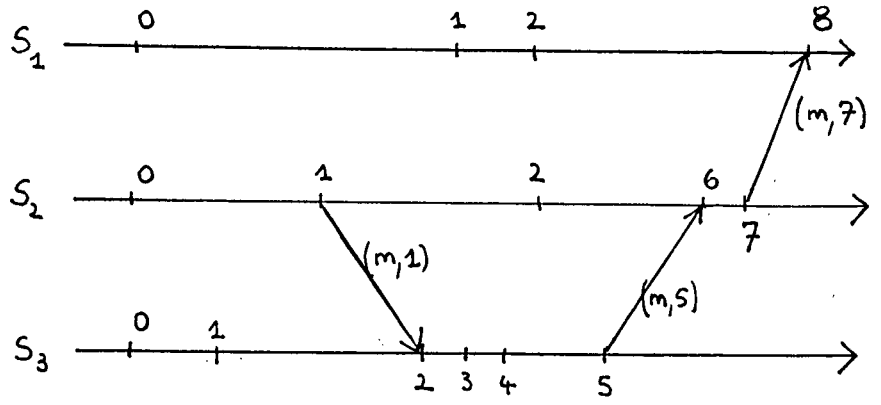


Figure 1 : Lamport's clocks

This timestamping technique and associated protocols are very useful as soon as one is interested in a total order consistent with causality precedence. This is the case, for example, when we want to give a meaning to the word "fifo" in a distributed context [L77].

**Remark**

Successive clock values used by each site $S_i$ are monotonically increasing but are not necessarily the sequence of all the successive integers : the updating protocol associated to each message reception can create an increase of $h_i$ greater than one. But the updating protocol ensures the following nice property : if $h$ is the clock value associated to an event $a$, then exactly $h$ events causally precede the event $a$ on the longest causal path ending at $a$ ; in other words thas is the minimum number of events that must occur before $a$ in the execution [F89b].(cf. figure 1).

## 2.3 Get the partial order

Solutions to distributed problems need to know if two events are or not causally related. As we have just seen, Lamport's timestamping mechanism is unable to detect causal independence. A mechanism allowing to detect causal dependence and causal independence has been proposed in 1988 (ten years after Lamport's proposal) simultaneously by C.J. Fidge [F88, F89a] and F. Mattern [M88]. This new mechanism has been extensively used by [CB89] to define a concurrency measure dedicated to distributed computations run on distributed systems.

Every site $S_i$ is endowed with a logical clock vector $vh_i[1 \ldots n]$, initialized to 0. When a local event occurs (sending, receiving or internal) the site $S_i$

executes $vh_i[i] := vh_i[i] + 1$. Every message is timestamped with the clock vector of its sender site, at sending time. Upon receiving a message *(m,vk)*, the receiving site $S_i$, executes, with the incrementation of $vh_i[i]$, the following clock vector updating :

$$j \in 1..n \ : vh_i[j] := \max(vh_i[j], vk[j])$$

F. Mattern exhibited the noteworthy property of this timestamping mechanism in [M88] : it expresses exactly the partial order of the causal precedence relationship.

Definition. $vh$ and $vk$ are two $n$ dimension clock vectors.

$$\begin{aligned} vh \leq vk &\iff \forall j : vh[j] \leq vk[j] \\ vh \geq vk &\iff \forall j : vh[j] \geq vk[j] \\ vh \parallel vk &\iff \textbf{not } (vh \leq vk) \textbf{ and not } (vk \leq vh) \end{aligned}$$

let $a$ and $b$ be two events timestamped respectively with the vectors $vh$ and $vk$ ; we have :

$$\begin{aligned} &a \rightarrow b \iff vh < vk \\ &a, b : \textit{independents} \iff vh \parallel vk \end{aligned}$$

**Remark 1** When an event occurs if we store the identity $i$ of the occurrence site $S_i$ and if we consider the pair *(clock vector, i)* as the timestamp, then the independence test can be simplified. Let $(vh, i)$ and $(vk, j)$ the timestamps of the events $a$ and $b$ ; we have now :

$$\textbf{not } (a \rightarrow b) \textbf{and not } (b \rightarrow a) \iff (vh[i] > vk[i]) \textbf{ and } (vh[j] < vk[j])$$

**Remark 2** Let an event $a$ be timestamped $vh$. We have the following relationships [S88] :

$vh[j]$ = number of events preceding causally $a$, which occured on the site $S_j$

$\sum_j vh[j]$ = total number of events causally preceding $a$.

This remark gives its meaning to each component of the clock vector ; see the remark 1 for the analogous meaning in the case of Lamport's clock.

This timestamping mechanism is very useful when causal independence between events has to be cast. This is the case in the definition of concurrency measures of a distributed application ; this is also the case in distributed debugging or in recovering after failures : in these cases one has to know if some events are or not causally related.

# 3 High level communication primitives

## 3.1 Motivation

As we have seen a distributed computation is structured as a partial order on the set of events it has produced. Two captures of this order have been done : an approximate but consistent one (Lamport) and an exact one (Fidge-Mattern). These are basic mechanisms to control and observe distributed executions. In lots of applications, the correctness criterium refers to a consistency predicate and as noticed by Birman and Marzullo [BM89] :

*Any system concerned with maintaining consistency must impose and respect ordering when operations conflict.*

Consequently it is very important to define communication primitives whose semantics is richer than the ones of the send and receive low-level primitives. Such high-level primitives must be in a position to constitute an interface for upcoming distributed kernels. We review some of them in the sequel. We dont review here high level synchronization primitives such as the multi-rendez-vous ; the interested reader will report to [C85, B89, R87, R88, LD87].

## 3.2 Causal ordering

The causal ordering notion extends to messages reception events the causal precedence relations existing on their sendings. Consider sendings of two messages $e_1 = send(m_1)$ and $e_2 = send(m_2)$ such that on the one hand $e_1 \rightarrow e_2$ and on the other hand $m_1$ and $m_2$ have the same destination site. Causal ordering ensures that :

$receive(m_1) \rightarrow receive(m_2)$

On the figure 2, causal ordering guarantees $m_1$ is delivered to $S_3$ before $m_2$.

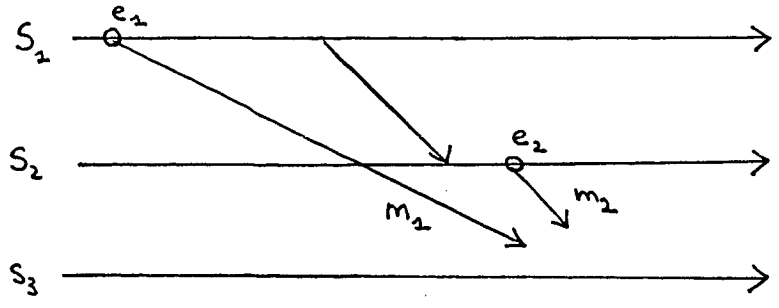


Figure 2 : Causal ordering

**Interest**

The semantics of causal ordering allows to eliminate a priori the undesirable part of the nondeterminism generated by arbitrary communication delays when one has to solve some problems. This property creates at the global level a good extension of the fifo property associated to channels. Besides the consistent updating of multiple copies of some data [JB86], it simplifies the implementation of a global observer ; the observer $S_3$, in figure 2, can consume immediatly the messages delivered to it : its sequential perception is consistent with the distributed execution on $S_1$ and $S_2$ as the order of reception at $S_3$ cannot be misleading concerning sending of these messages [RS89]. It is also possible with causal ordering to implement trivially immediate ordered service in distributed systems [KK89].

That communication primitive has been implemented in the ISIS system [BJ87] ; this realization lies on a forwarding technique, and resist crash failures. A very simple implementation has been proposed by [RS89] ; it relies on a technique counting sent and received messages as do some termination detection distributed algorithms [M87, HJPR87, R88] or some protocols using sequence number [S76, R88].

## 3.3 Multicast primitives

We consider now the notion of multicast group. Such a group is a collection of sites that are the destinations of the same sequence of messages. If the group is the set of all the sites, we speak of broadcast. Multicast primitives provide some guarantees regarding the order in which message are delivered to destination sites [Cr86, GS89] (here sending order is no relevant).

i) Single source ordering

If message $m_1$ and $m_2$ are sent by the same site and are addressed to the same multicast group then all destination sites get them in the same order.

ii) Multiple source ordering

In this case the source does not matter. If $m_1$ and $m_2$ are addressed to the same multicast group then all destination sites get them in the same order (whatever the senders are).

iii ) Multiple group ordering

In this case if messages $m_1$ and $m_2$ are delivered to several sites then these sites get them in the same relative order (messages can come from different sources and can be addressed to different but overlapping groups).

As noticed by [GS89] : there are of course applications that do not require all of these (or even any of these) properties. But there are applications where the receipt of messages in different orders will lead inconsistency or deadlock problems. In other words these properties facilitate the implementation of partial correctness (safety) and of total correctness (liveness).

As one can see causal ordering and multicast primitives are orthogonal mechanisms : none of them include the other. Now we look at some implementation proposals.

## 3.4 Sketch of implementation proposals

### 3.4.1 Causal ordering

The proposal of [RS89] is very simple. It uses two data structures per site $S_i$, piggybacking of control information by messages and update of local data when a site receives a message.

- local data :

$$rec_i : array\ [1..n] \text{ of integer init } 0$$
$$sent_i : array\ [1..n, 1..n] \text{ of integer init } 0$$

- sending of $m$ from $S_i$ to $S_j$

$$sent_i[i,j] := sent_i[i,j] + 1$$
$$send(m, sent_i) \ to \ S_j$$

- reception by $S_i$ of *(m,stm)* from $S_j$ :

$wait(rec_i[j] + 1 = stm[j,i])$ and (for any $k \neq j : rec_i[k] \geq stm[k,i]$) ;
*delivery of* $m$;
$rec_i[j] := rec_i[j] + 1$;
$sent_i[k,l] := \max(\ \text{sent}_i[k,l], stm[k,l]) :$ for any $(k,l)$

A formal proof of correctness is given in [RS89]. If we look at the wait condition, intuitively it tells us a message $m$ sent by $S_j$ can be delivered to $S_i$ : firstly if all messages sent by $S_j$ before $m$ have already been received (first part of the condition) and secondly if all other messages that causally precede $m$, have been delivered (second part of the condition).

It is possible to obtain a similar algorithm using Fidge-Mattern clock-vectors as an observation device allowing consistent delivery of messages [SSE89].

### 3.4.2 Multiple group ordering

Traditionnal solutions assign unique timestamps to messages (à la Lamport timestamps) and then delivers messages in the timestamp order which is total and unique all over the system. Before the delivery of a message to a site $S_i$ can occur, $S_i$ has to be sure no other message with a lower timestamp is present in the system and still not delivered. A protocol is therefore needed to ensure this property ; consequently all the protocols implementing some kind of group ordering in asynchronous distributed systems have (at least) two phases of control message exchanges before a delivery can occur. The more interesting of these protocols are the Lamport's one [L78], and the Skeen's one [BJ87] (implemented in the ISIS distributed system). Other protocols have been proposed ; [CM84] uses a dual approach : messages are first ordered and then sent to their destination site : here there are also two phases of exchange and a coordinator site is necessary (associated to a group, it receives all the messages to be received by this group, orders them and then forwards them

to the destination sites using a network traversal scheme [R88]). This solution has been extended by [GS89] in order to solve the multiple group ordering ; they use a message propagation graph extending the coordinator approach of [CM84] ; this graph is some union of the destination groups structured in a diffusion tree.

# 4 Conclusion

As we have seen executions of distributed applications or distributed systems can be characterized as sets of events structured by a partial order. In this partial order, events local to a site are totally ordered ; and for each message its sending event precedes its receiving event. This order, put forward by Lamport in 1978 and called precedence order, is a fundamental notion to design or to analyze lots of distributed control algorithms and consequently is a masterpiece in the design of systems architectures or distributed kernels.

The first part of the paper has examined in depth different logical clocks mechanisms allowing to capture this order in part or in whole. So if Lamport's logical clocks allow to build a total order consistent with this partial order they cannot afford to decide if two events are logically independent. Such an independence knowledge can be essential in order to realize some distributed debugging or distributed observation. Clock vectors, proposed simultaneously by Fidge and by Mattern in 1988, allow to characterize dependence and independence of events. A study of their properties has been done together with a presentation of these clock mechanisms.

The second part of the paper has given a survey of some high level communication primitives based on this order (causal ordering, multicast ordering, group ordering, etc). We have shown their usefulness in control algorithms and given some implementation details ; they rest on clock mechanisms (or variations) presented in the first part. These primitives can be classified into two categories : some asking for one exchange phase only, and others asking for at least two exchange phases (these are some kind of 2PL protocols). Some of these primitives have been proposed and realized in some systems (ISIS for example) ; some have been defined and are being implemented. In some sense they constitute a communication kernel for distributed systems, allowing to design applications without taking into account irrelevant aspect not mastered by low level communication primitives ; actually they hide some part

of the undesirable non- determinism inherent within distributed systems.

The understanding of precedence relations between events and the definition of high level communication primitives are two main trumps to master architecture, and software engineering of distributed systems and distributed kernels. The aim of this paper was to study these 2 points from this point of view.

# 5 References


[B 89] BAGRODIA R. *Process sychronization : design and performance Evaluation of distributed algorithms.* IEEE Trans. on SE, vol 15,9,(Sept 1989),pp 1053-1065

[BJ 87] BIRMAN K.P., JOSEPH T.A. *Reliable communication in the presence of failures.* ACM Tocs, vol 5,1,(Feb 1987), pp 47-76

[BJ 89] BIRMAN K.P., MARZULLO K. *The role of order in distributed programs.* Tech. Report,TR 89-1001, Cornell Univ.,(May 1989), 22 p

[CM 84] CHANG J., MAXEMCHUK N.F. *Reliable Broadcast Protocols.* ACM Tocs, vol 2,3,(August 1984), pp 251-273

[C 85] CHARLESWORTH A. *The multiway rendez-vous.* ACM TOPLAS, 9,2, (July 1987), pp 350-366

[CB 89] CHARRON-BOST B. *Measure of parallelism of distributed computations.* Proc STACS 89,LNCS 349,(1989), pp 434- 445

[Cr 86] CRISTIAN F. et al. *Atomic broadcast : from simple message diffusion to byzantine agreement.* Tech. Report,RJ 5244(54244), IBM Almaden Research Center, (July 1986), 30 p

[F 88] FIDGE C.J. *Timestamps in message-passing systems that preserves the partial ordering.* Proc. 11th Australian Comp. Conf., (February 1988)

[F 89a] FIDGE C.J. *Dynamic Analysis of Event Orderings in Message-passing Systems.* Ph. D. Thesis, Australian Nat. University, (May 1989), 229 p

[F 89b] FIDGE C.J. *A simple run-time concurrency measure.* Research report, queensland University, (1989), 12 p

[GS 89] GARCIA-MOLINA H., SPAUSTER A.M. *Message ordering in a multi-cast environment.* Proc. IEEE Int. Conf. on DCS, Newport Beach, CA, (June 1989), pp 354-361

[HJPR 87] HELARY J.M., JARD Cl, PLOUZEAU N., RAYNAL M. *Detection of stable properties in distributed applications.* Proc. 6th ACM Symp. on PODC, (August 1987), pp 125-136

[JB 86] JOSEPH T., BIRMAN K. *Low cost management of Replicated Data in Fault Tolerant Distributed Systems.* ACM TOCS, 4,1, (February 1986), pp 54-70

[KK 89] KEARNS Ph, KOODALATTUPURAM B. *Immediate ordered service in Distributed systems.* Proc. IEEE 9th IC DCS, (1989), pp 611-618

[L 78] LAMPORT L. *Time, Clocks and the Ordering of Events in a Distributed System.* Comm. ACM, vol 21,7,(July 1978), pp 558-565

[LD 87] LEE I., DAVIDSON S. *Adding Time to Synchronous Process Communications.* IEEE Trans. on Computers, 36,8,(August 1987),pp 941-948

[L 77] LE LANN G. *Distributed systems : towards a formal approach.* IFIP Congress, Toronto, (1977), pp 155-160

[M 88] MATTERN F. *Virtual Time and global states in distributed systems.* Int. Conf. Parallel on Distributed Algorithm, North-Holland,(1988), pp 215-226

[M 87] MATTERN F. *Algorithms for Distributed Termination Detection.* Distributed Computing, 2,3, (1987), pp 161-175

[R 87] RAMESH S. *A new and efficient Implementation of Multiproccss synchronization.* Parle Conf., LNCS 259,()1987), pp 387-401

[R 88] RAYNAL M. *Networks and distributed computation : concepts, tools and algorithms.* The MIT Press, (1988), 166 p

[RS 89] RAYNAL M., SCHIPER A., TOUEG S. *The Causal Ordering abstraction and a simple way to implement it.* Research Report INRIA, 1132, (December 1989), 8 p

[SSE 89] SCHIPER A., SANDOZ A., EGGLI J. *A new algorithm to implement Causal Ordering.* 3rd Workshop on Distributed Algorithms, LNCS 392,(1989), pp 219-232

[S 76] STENNING W. *A data transfer protocol.* Computer Network, 1, (1976), pp 99-110

[S 88] SCHMUCK F. *The use of efficient broadcast protocols in asynchronous distributed systems.* Ph. Thesis, Cornell University, TR 88-928,(1988), 124 p

## Liste des dernières publications internes parues à l'IRISA

**PI 516** **COMMENT INTRODUIRE LA CONTIGUITE EN ANALYSE DES CORRESPONDANCES ? Application en segmentation d'image.**
Brigitte ESCOFIER, Habib BENALI, Kaddour BACHAR
Février 1990, 26 Pages.

**PI 517** **MACHINE MODELING AND LOOP OPTIMIZATION FOR HORIZONTAL MICROCODED MACHINES**
François BODIN, François CHAROT
Février 1990, 24 Pages.

**PI 518** **MULTISCALE SYSTEM THEORY**
Albert BENVENISTE, Ramine Nikoukhah, Alan S. Willsky.
Février 1990, 30 Pages.

**PI 519** **PANDORE : A SYSTEM TO MANAGE DATA DISTRIBUTION**
Françoise ANDRE, Jean-Louis PAZAT, Henry THOMAS
Février 1990, 14 Pages.

**PI 520** **SCHEDULING AFFINE PARAMETERIZED RECURRENCES BY MEANS OF VARIABLE DEPENDENT TIMING FUNCTIONS**
Christophe MAURAS, Patrice QUINTON
Sanjay RAJOPADHYE, Yannick SAOUTER
Février 1990, 14 Pages.

**PI 521** **COMPUTABILITY OF RECURRENCE EQUATIONS**
Yannick SAOUTER, Patrice QUINTON
Février 1990, 28 Pages.

**PI 522** **PROGRAMMING BY MULTISET TRANSFORMATION**
Jean-Pierre BANATRE, Daniel LE METAYER
Mars 1990, 26 Pages.

**PI 523** **GOTHIC MEMORY MANAGEMENT : A MULTIPROCESSOR SHARED SINGLE LEVEL STORE**
Béatrice MICHEL
Mars 1990, 20 Pages.

**PI 524** **ORDER NOTIONS AND ATOMIC MULTICAST IN DISTRIBUTED SYSTEMS : A SHORT SURVEY**
Michel RAYNAL
Mars 1990, 18 Pages.



Imprimé en France
par
. l'Institut National de Recherche en Informatique et en Automatique .