# Specification and Properties of a Cache Coherence Protocol Model

C.Chatelain C.Girault S.Haddad
Universite Paris VI -C.N.R.S. (M.A.S.I. and C$^3$/Algorithmes répartis)
4 Place Jussieu, 75252 Paris Cedex 05

**Keywords**: Petri nets, protocols, specificatiorl, verification, cache coherence, memory hierarchy, multlprocessors

## 1 Introduction

W*e* describe a cache-primary memory protocol for an architecture composed of several processors, each with their own local cache, connected via a switching structure to a shared main memory, Caches are used to speed up: the memory hierarchy the caches and the primary memory are respectively partitioned into small blocks and lines of fixed size that are automatically managed by hardware just about "page on demand" feature by operating systems, However the access time ratio between primary memory and cache is 10 versus over 10000 in the secondary-primary memory case and data transfers may be speeded up by organlzlng the memory into interleaved banks, This compensates for the cache small size inducing frequent line defaults and replacements that must be very quickly managed. Cache organizatlons and performances are extensively covered in [Smith 82, 85]. Some projects [Gadjski 83, Gottlieb 83] envision several hundreds of processors. Because of access conflicts, the sharing a unique common cache becomes quickly inefficient for targe numbers of processors

The introduction of local caches in a multiprocessor gives rise to the **"cache coherence problem**". Several caches can contain a copy of a particular main memory tine but it is required that all programs continue to run exactly like for a unique cache without having to introduce extra synchronizations or data reorganizations. Processes are only synchronized for sharing of logical entities without taking care of thelr implementatlon into the memory lines. Thus it is necessary to automatically prevent processors from simultaneously modifying their respective copies of a line as well as from reading a line while another is writing it. Moreover a processor requiring a copy of a memory location must always receive the most up-to-date version, otherwise inconsistencies would arise. These requirements are the same as for the classical "readers and writers" problem but here the sequences of accesses may be interrupted by requests from other caches and block replacements must be managed.

For the "**write-through** strategy" the primary memory is updated each time a processor performs a write access, thus any cache may obtain up-to-date information either directly from the memory or by continuously listening to bus exchanges (snooping caches). This elementary strategy does not apply for large processor numbers. The more efficient "**write-back** strategy" lightens the bus congestlon by avoiding these systematic updates: instead a complete block update is only required when a modified block must be replaced or accessed by another cache [Censier 78, Archibald 84, Katz 95, Yen 85].

The bus and memory bottlenecks may be avoided by splitting the memory into parallel modules managed by parallel controllers and by using multibusses or networks allowing parallel exchanges [Papamarcos84, Archibald 85] These needed parallelisms complicate the coherence problem. Hypotheses on the network and careful serializations must be introduced to deal with the asynchronism of cache and controllers.

Formal models for these protocols are needed to insure their correctness and to establish bounds for the hardware resources required both at the cache and controller levels. Our first purpose is to give a top down description of such a model based on Predicate/Transition nets [Genrich 79, 80, Jensen 81, Brams 82]. This modeling has been a valuable way to understand and to formally describe a complex synchronization protocol, to experiment with variants and to increase the parallelism inside the proposed architectural design [Baer 85b] The size of the model, its complex tokens and predicates, the use of arrays of FIFO queues forbid to study it by unfolding the colored net into an uncolored one for using after classical validation tools. Moreover fairness and temporal properties are also very complex. Hence our second purpose is to underline some key behavioral points and explain how colored invariants are a basis for further behavioral studies. They have been directly constructed together with the net itself, but some of them may also be obtained by recent methods and tools [Vautherin 84, Haddad 86a, 86b].

This paper is organized as follows. In the next section we describe the hardware components and their physical and logical interconnections. Then in section 3 we describe in details a particular coherence protocol for which we give in section 4 a Predicate/Transition model. Section 5 gives the basic ideas for validation and finally Section 6 concludes on possible extensions and other applications of our model.

## 2 Multiprocessor architecture

The rnultiprocessor architecture consists of.

- Processor-cache pairs Pk-Ck: **N**=[0..N-1] is the set of their indices,
- Primary memory modules Mx and associated controllers Kx: **M**=[0..M-1] is the set of their indices,
- An interconnection network.

From a cache point of view and for a protocol specification, it is sufficient to consider a processor as interpreting instructions to produce elementary **access orders** for the cache and consuming the delivered results. FIFO buffers may allow the processor to prepare several access orders in advance but here this pipelining will not be considered. Each processor Pk is provided with its own cache memory element Ck and communicates only with it by a private interface

A **cache** manages a set of **blocks** which may contain copies of some primary memory **lines**. For that it classically contains:

- An associative **directory** to recognize if a given line a is present into sorrle block b of the cache. One **state bit s[b]** per block allows to know whether the block b contain is **valid**, i.e. contains an up to date copy of the line a, or is **invalid**. A **modified bit m[b]** indicates whether the cache may **write** and modify the block b or may only **read** it.
- A set of data values corresponding to the contents of the respective blocks,
- FIFO queues for incoming and outcoming messages

On the one hand a cache has a **local part** which responds to the orders of the corresponding processor and a **global part** which responds to the messages from all the controllers. On the other hand a cache is composed of two separate units: a **control unit** and a **memory unit**. They share common data structures needing partial exclusion. They may be organized into pipelined stages: buffer management, associative search, replacement, order and message analyses, data transfers.

The primary memory is divided into independent **modules**. Each module Mx stores a fixed subset of the **lines** that may be copied into cache blocks. All the lines of a module Mx are managed by its own **controller** Kx: the number x of the controller managing a given line â is obtained from the higher bits a of the line address, while the remaining lower bits ã give the line number within Mx and we note a=(â,ã). The memory may thus be simply upgraded by adding new modules. To speed up block transfers each module usually consists of interleaved banks but this hardware detail has not to appear in the model,

A controller contains:

- a bit map to encode the **global state** of each line: given grants and information about owner caches,
- a set of data values that are the contents of the respective lines,
- FIFO queues for incomlrlg and outcomlng messages.

A controller is divided into a **control unit** and a **memory unit**. The first one manages informations on all the module lines and exchanges command messages with the caches. The memory unit receives and sends the data via the network. Moreover auxiliary stages manage in parallel the FIFO queues and the network interface.

The global state plays a key role: various solutions have been proposed according to the amount of information distributed among caches and controllers [Censier 78, Archibald 84, Yen 85]. N+ 1 bits per line may allow a controller to exactly know the state and all the owners of each line but the amount of Informatlon for all the lines becomes costly when the number N of processors increases. This paper deals with a solution of J.Archibald and J-L.Baer, using only a fixed amount of two bits per line, that allows for easy expandability in the number of processor-cache pairs, This solution has been already modeled to study the architectural design [Baer 85a. 85b]. Here we present a simplified variant: it deletes some states and messages useful for performance considerations but is more suitable to study the protocol properties.

For each line the global state allows to distinguish three cases: "**Absent**" (the line is in no cache), "**PresentW**" (only one cache has been granted to write the line that is considered as modified), and "**PresentR**" (any number of caches have been granted to only read the line that remains not modified). Since the information given by these states does not include the location of the owning caches, the memory controller might need to broadcast queries to all caches (e.g., for writing-back or invalidating a given line).

Any cache may be connected with any controller, hence protocols imply the exchanges of control and data messages between all the caches and all the controllers; this simultaneous transmission of messages between several partners is the task of the **interconnection network** [Auguin 84, Baer 85b. Chi Yuan 84, Siegel 85]. To avoid conflicts two independent networks are used the first one from caches towards modules and the second one from modules towards caches. For verification purposes the model must be independent of the network design, hence we assume as little as possible about the message tralfers.

The network may transfer different types of messages of various length distinguished by appropriate headers and cut into consecutive packets. Packets may contain any combination of orders, addresses, status bits and data. Here one header flag (F= L or S) distinguishes short messages for only control information or long ones that include also data for line contents. Some emitter and receiver units will recognize and manage these packets.

Messages between the same partners must be delivered in the emission order; entries in the input-output queues of the controllers and buffering must avoid any loss of messages and so all the ordered transfers are assumed to be performed. Also the message broadcasting is left in charge to the network layer. All broadcast messages are assumed to be synchronously delivered.

## 3 Coherence protocol

Four cases are considered, depending on a read or write order, and on a **hit** (valid copy of the desired line already present into the cache) or a miss. In addition messages are implied by line replacements.

- ***Read hit***: In case of immediate access, no special action is needed. As this is the case of most of the accesses, its detection must be efficient.
- ***Write hit***: An immediate access is also possible if the line has been already modified. Otherwise Ck sends a REQUEST for modifying the line to the controller Kx monitoring it. This REQUEST might entail a broadcast QUERY to invalidate the line. If it is present in some caches then the GRANT will be given by Kx.
- ***Replacement***: When Ck needs to make room for a new line, it must replace an old one. If this **old line** is valid and has been modified a write-back is needed: the controller must be informed (for a state change) by a RETURN command and moreover data must be written back. Otherwise the Ck may use freely the block containing the old line.
- ***Read miss***: Once a possible replacement has been taken care of, a REQUEST for a load on read is made to the controller. If the line is not modified in some cache, Kx can immediately give the GRANT. Otherwise Kx sends before a QUERY to force the owning cache to update the line by a write-back. Then the GRANT and the data are sent to the requiring cache.
- ***Write miss***: Kx must broadcast a QUERY to **invalidate** the line if it is owned by some caches only for reading or to enforce a **purge** that is both a write-back and an invalidation if the line is owned for writing in one cache. The GRANT and the data are finally sent to the requiring cache.

Let us describe now the messages sent by the caches (*) and by the controllers (#).

* For any access such that a cache owns a valid copy of a line and does not need a higher grant (case read hit or case write hit for a modified line), the cache Ck may immediately access its line copy, But if Ck owns a line a only for reading while Pk orders to modify it (case write hit for an unmodified line) or if it does not own such a valid copy (case read miss or case write miss) it must send a command REQUEST(k,a,t) to the controller x=â where t=r or t=w indicates the desired GRANT for this line.
* Before getting a line a, Ck may also need to make room for it by replacing another old line o. If the copy of this old line is still valid and has been modified, the cache warns the controller Ky (with y=ô) by sending a command RETURN(k,a,**C**) with the line content **C**, in order to PURGE (i.e. write-back and invalidate) the line. If the copy is invalid or unmodified, Ck has only to FREE the associated block without warning Ky.

# When a line requested by a cache Ck, is owned for writing by some other cache (state "presentW"), the controller Kx must broadcast a QUERY message to obtain that the unknown owner writes back the line. Hence Kx waits for one RETURN command with the line content before sending a GRANT to the requesting cache. In case of a READ REQUEST the owned line must be updated but may remain valid and shared. In case of a WRITE REQUEST the line must be purged by the owning cache.
When the requested line is absent or only owned for reading the controller does not need to obtain a RETURN for writing-back and thus may immediately send a GRANT(k,â,ã,g,**C**) to Ck, where g= r or w precises if writing is allowed and **C** is the line content. In case of a WRITE REQUEST and if Kx suspects that there is at least one owner (states "PresentR"), it broadcasts a QUERY forcing the invalidation by all owners. To avoid traffic overload Kx does not wait any acknowledgement from the caches. So the coherence protocol needs to be sure that all invalidations will be made before the granted access. In case of a READ REQUEST or if Kx is sure that no cache owns the line (state "Absent"), no broadcast is needed.
A message QUERY (h,â,ã,u) is broadcast to the set of all caches excepting the requesting one (h belongs to the set (**N**-k) of the destination cache numbers). The parameter u precises the new line-use i.e. whether the line a=â,ã must remain valid or not. The caches directories allow to determine their behavior.
* Whenever a cache Ck receives a QUERY about a line which it does not own or which is invalid, it has nothing to do. A valid queried line which is unmodified has only to be invalidated without sending a RETURN whereas if it is modified the cache must purge or update the line according to u and send a RETURN.
# Each time a line is returned by a cache, the controller has to change its state and to copy the line content. Any return for writing-back a requested line is called an **awaited** return because and allow the controller Kx send a GRANT with the new line content to the requesting cache. Conversely the other returns are called **unwaited** returns.

## 4. Petri net model

The structure of our model reflects the underlying hardware structure. It consists of four nets working in parallel and interconnected by interface places modeling common information structures or buffers. These nets are:

- the P-net for processor actions,
- the B-interface for the private bus of a processor-cache pair,
- the L-net for local cache-processor interaction,
- the C-interface for the cache common data structures shared by Land G,
- the G-net for the global cache actions upon receiving messages from the controller,
- the N-interface for the network communications between L, G and K,
- the K-net for the controller flow of actions

Two main sets **N**=[0..N-1] and **M**=[0..M-1] of colored tokens are associated with each cache and each controller. Let h, k, l $\in$ **N** and x $\in$ **M**.
**A**=[0..A-1] = [0..M-1, 0..A/M-1] is the set of line addresses where A/M is the number of lines in each memory module. Let a, o $\in$ **A**
B=[0..B-1] J is the set of block addresses into a cache. Let b $\in$ B.

Other sets of colors are used for the values of control variables and message parameters.

$t \in (r,w)$ is an access type: either read or write
$m \in (r,w)$ is a block modified bit: either read, or write
$s \in (v,i)$ is a block state bit: either valid or invalid.
$u \in (v,i)$ is new block use queried by a controller: either valid or invalid.
$g \in (z,r,w)$ is a line global state: respectively absent, presentR or presentW.
$F \in (S, L)$ is an header flag to distinguish either short messages for requests and queries or long ones for returns and grants.

### 4.1 Processors and bus interface

A sequential **processor net** is sketched: when it is ready (k token in Pr) it may submit a new access order to its cache (transition Pinst) and the processor becomes waiting (k token in Pw) for the access being done. After the instruction execution (transition Pexec) it becomes again ready.

At the initialization the place Pr contains a complete set of k tokens for all k in **N**.

The **bus interface** is composed of 3 places:

- The processor transmits the desired line address a and access type t thereby a (k,a,t) token into the place BinstR: the cache analyses only these informations to perform all needed coherence actions before allowing the access.
- The place Binstp may contain a (k,f,d,c) token used later to perform the access itself: f precises the access function, d is the deplacement within the line a and c is the new value of the content of the word (a,d) in case of a write or is a dummy value in case of a read.
- The place BlnstD may contain a (k,c') token modeling the result delivered to the processor, this new value c' is the preceding content of the word (a,d) in case of a read or a dummy value in case of a write.

Functions such that atomic exchanges or incrementations are thus allowed.

These places as well as the places Pr and Pw may be FIFO ones, allowing (in a further model) the processor to give several access orders in advance.

### 4.2. Network interface

All the cache subnets communicate with all the controller subnets through 2 sets of arrays of FIFO places that model the streams of the common interconnection networks. In fact two unidirectional networks, used either for control messages or data transfers, are the simplest solution for implementation. For sake of place we do not detail here how supplementary network interface units would allow to deal with packets multiplexing and demultiplexing [Baer 85b].

- The places Creq and Cret are used for the commands sent by the caches to the controllers. Here FIFO places are not needed for serialization but useful for fairness. Some freedom is left to give priority to the transfer of requests
- The places Kin is used for the commands received by the controllers from the caches. The tokens may be dispatched into the places Kreq and Kret to be managed by the parallel request unit and return unit of each controller.
- The place Kout is used for the messages sent by the controllers to the caches. FIFO ordering of queries (F=S) and grants (F=L) is essential from each controller towards each cache.
- The place Cin is used for the messages received by the caches from controllers. FIFO ordering of queries or grants concerning each line is essential within each cache.

Each cache FIFO-array place is indexed by the cache numbers modeling a private independent queue for each cache. Each corresponding controller FIFO-array place is indexed by the controller numbers modeling a private independent queue for each controller. Network transitions between the two FIFO-array places model the message transfer between any of the first queues to any of the second ones according to the emitter and receiver parameters.

- The transition NSCK removes a (k,a,t) token from Creq to transfer a (S,k,a,t) token into Kin.
- The transition NLCK removes a (k,a,**C**) token from Cret to transfer a (L,k,a,**C**) one into Kin.
- The transition NSKC moves a (S,k,a,u) token from Kout to Cin for a short query transfer. Doing for each cache implements the broadcast.
- The transition NLKC moves an (L,k,a,g,**C**) token from Kout to Cin for a long grant transfer including the line content **C**: thus the order of grants and queries is kept.

## 4.3 Caches

A cache is composed of a loca! part managing its processor orders and a global part managing the messages coming from all the controllers. These two parts share common data structures containing informations on all the blocks owned by this cache.

### 4.3.1 Common data structure interface

- The place Cdirsm models the cache directory: it always contains a set of (k,b,a,s,m) tokens for all possible values of k and b, allowing the associative search of the block b that may contain a given line a. At the initialization the a components of all these tokens are meaningless. To achieve high speed requirements this search is very simplified in the hardware: a given line may be only placed in a very small subset of blocks, but this has no effect on the model.

  The s component gives the state s of the line copy contained into the block b: it indicates whether the content of this block b is valid or not. At the initialization all the s components must be invalid.

  The m component gives the mode m (r=read or w=write) of the bloc b: it indicates what accesses are allowed for this block and thus whether this block b has been modified or not. At the initialization all the m components must be set to r.

Each cache is split into a control unit and a memory unit that perform either local or global work and need to communicate.

- A k token into the place CidleC indicates that the control unit of Ck is idle. It insures a mutual exclusion between instruction analysis in the local part and query or grant analyses in the global part.
- A k token into the place CidleM indicates that the memory unit of Ck is idle. It insures a mutual exclusion between word access in the local part and block transfers in the global part.
- The places CgetR and CputR may contain the tokens denoting the blocks that are to be loaded or written-back by the cache memory unit. The place CputR is a FIFO one, allowing some transfers to be waiting. The place CgetR is 1-bounded if the processor orders are not pipe-lined and must be a FIFO one otherwise.
- The place CaccR indicates that the access wanted by the processor may and must be performed.

### 4.3.2 Cache local part

A cache has to supervise the ordered accesses and to manage the block contents.

**Instruction acceptance**

By the transition LAinst the cache accepts a new access order (k,a,t) produced by its processor. It must be idle and waiting for a new order. More over to avoid an access when an invalidation, an update or a purge are required an inhibitor arc from Cin forbids this acceptance as soon as a query for the line a is received. The transition moves the token from BinstR to L instA.

**Presence analysis**

The cache checks if a copy of the line a is present in some block b. This is known by the presence or the absence of a (k,b,a,s,m) token, into the associative directory Cdirsm. The state of a copy present into a block b is precised by the s component with either s=v (valid copy) or s=i (invalid copy).

- transition LvalhitA: The line a is present in the block b of Ck and still valid (s=v), thus no replacement nor line getting are needed. Moreover if (t=r) or (m=w) no request is needed and the local access may be immediately allowed. Therefore the (k,a,t) token in LinstA becomes a (k,b) token in CaccR while the directory is left unchanged and the cache control unit becomes idle again (k token into CidleC).
- transition LvalhitM: If the line a is present and valid (s=v), but if (t=w) and (m=r) a request to the controller is needed because the processor wants to write an unmodified line. Therefore the (k,a,t) token in LinstA is moved to Creq while the directory is unchanged and the control unit becomes idle.
- transition Linvhit: The line a has been present into the block b but is now invalid because either a purge or an invalidation has occurred. As the block b is free no replacement is needed. Therefore the (k,a,t) token in LinstA is moved to Creq denoting a READ or a WRITE REQUEST.
- transition Lmiss: The line a does not appear in any cache block. This condition is modeled by an inhibitor arc from Cdirvm requiring the absence of all tokens (k,b,a,s,m) for b belonging to the set **B**. The token from LinstA is moved to Lrep for the replacement algorithm.

**Replacement algorithm**

The three next transitions choose a block b for the line a, move the token from LinstA to Creq for a READ or WRITE REQUEST and check the old line o that was previously in b. The (k,b,o,s,m) token allows to distinguish three cases.

- transition LinvrepN: If the line o is invalid (s=i) in the block b nothing is needed
- transitiorl LvalrepF: If the line o is valid but unmodified (s=v and m=r) it may be freed without any warning because no writing-back is needed and the controller does not keep trace of the owners a given line.
- transition LvalrepP: If the line o is valid and modified a purge is needed therefore a (k,b,o) token is placed into CputR ordering the cache memory unit to write-back the block content into the memory module.

The token in Cdirvm is changed into a (k,b,a,i,r) token denoting that from now on the block b is associated with the new line a but remains invalid and by convention unmodified until the line a will be gotten.

**Cache access**

The access may either immediately occur in case of a valid hit without request or on the contrary after a grant and a line transfer. In any case the access is allowed as soon as the cache memory is available that is when a (k,b) token appears in CaccR.

The transition Lacc performs this access using the parameters (k,f,d,c) found into Binstp: d is the deplacement within the line a, f precises the access function (load, store or exchange) and c is the new value of the content of the word (a,d) in case of a write or exchange or is a dummy value in case of a read. It delivers a (k,c') token to the processor, via the place BinstD where c' is the preceding content of the word (a,d) or is dummy in case of a write.

An inhibitor arc, issuing from CaccR, has a very important role in the cache global part: in case of a concurrent query for the same line it forbids any writing-back of the block b (and thus any modification by another cache) before performing the wanted local access on the same version of the block as when it has been allowed. This also insures the fairness of accesses for the processor because the block content may not be purged or updated before the access. A parallel invalidation is possible but it does not matter: only a read is allowed for Pk, another processor Ph will perform a write and, as no logical synchronization has occurred between Ph and Pk, the accesses may concern independent data in the same line. However the invalidation will immediately forbids further accesses: if, after a synchronization with Ph, Pk tries to read the line it will be forced to request the new line version.

### 4.3.3 Cache global part

The cache global part has to analyze the queries broadcast by the controllers, to wait for the grants and to transfer the block contents.

Queries and grants are kept in order in a unique FIFO place Cin, and are distinguished by the flag either F=S for queries or F=L for grants that also convey the line content. This ordering is essential: when conflicts occur between two caches, it happens that the FIFO contains for the same line an invalidation query, followed by a grant because in parallel the cache has requested to write, followed by a purge because an other cache has also request to write. Of course independent messages for different lines may be merged in the file and the FIFO is also useful for fairness.

The controllers communicate only in terms of line addresses within the primary memory, thus when receiving a query or a grant for a line a, an associative search in the directory Cdirvm is needed to know if a copy of this line a is associated with some block b, and in this case whether this copy is valid or not and whether it has been modified or not.

#### Query acceptance

By the transition GAquer the cache Ck accepts a message QUERY(S,k,a,u) broadcast by a controller where u is the new use of the line queried by the controller (u=i for an invalidation or a purge and u=v for an update). Of course the exclusion for the cache control unit is needed. A(k,a,u) token is put into GquerA.

#### Query analysis

- transition GvalhitPU: If a valid modified copy of the line a is present in some block b then a write-back is needed. The token in GquerA is completed by b to give a (k,b,a,u) token in the FIFO place CputR ordering a RETURN to the memory unit. The directory Cdirvm is immediately updated according to the u component of the query: if u=v the cache is still allowed to read the block.
- transition GvalhitI: If a valid unmodified copy of the line a is in b no write-back is needed but only an invalidation. Thus a (k,b,a,i,r) token is put in Cdirvm where a becomes meaningless.
- transition Ginvhit: If a copy of the line a has been present in some block b but has become invalid the cache control unit is not concerned by this query.

- transition Gmiss: If there does not exist any block b containing the line a the cache control unit is again not concerned.

**Grant acceptance**
The grants inform the cache about the transfers of line contents performed at the initiative of the controllers: the global part of the cache has to get the transferred data into the associated blocks.
The transition GAgrant accepts the message GRANT(L,k,a,g,**C**) sent by a controller where g is the new type of granted accesses. The controller sets g=r for a READ GRANT and g=w for a WRITE or MODIFY GRANT.
Here to simplify the line content **C** is always transferred into the message. In case of a MODIFY GRANT this may be superfluous. This is quite complicate to manage because a line may become invalidate after a cache has sent a MODIFY REQUEST and the controllers do not know what are the cache owning a line [Baer 85a, 85b].

**Grant analysis**
The transition GRWM updates the directory Cdirvm: the (k,b,a,v,g) token precises the new access rights to the block b. A (k,b,a,**C**) token is put into CgetR to notify the cache memory unit that it must copy the transferred data into the block b before performing the access itself. It would easily be possible to avoid the associative search by storing the b-a association at the request time.

**Data transfers**
The cache memory unit has either to get into a block the copy of a line transferred from a module or to transfer a block content that will be put into the associated line of a module. Sufficient output buffers are needed for these block contents and modeled by the use of the FIFO place Cret. The memory operations are not atomic, so we explicit their beginnings and ends between which the cache memory unit remains busy.
A lot of serialization conditions must be insured before beginning the data transfers notified by the tokens in the places CputR and CgetR.

- transition GgetB: A (k,b,a,**C**) token in CgetR indicates that the content **C** of the line a must be gotten into the block b. But if a replacement is needed for b, it must be done before or, at least the old block content must already be put into the output buffer Cret: thus there is an inhibitor arc applying to any (k,b,o) token in the file CputR concerning this block b of this cache k provided that it concerns an old line o distinct from the new line a.
- transition GoutB: A (k,b,a) token in CputR indicates that the content of the block b must be extracted from the cache memory, put into the FIFO output buffer Cret and transferred. However because of the request of another cache a query for a purge or an update of the line a may be received soon after this line has been granted but the corresponding return must only occur after the line getting and also after the granted access has been performed possibly modifying the block content. Thus an inhibitor arc must delay this return while there is a (k,b,a,**C**) token in CgetR for the same line a. The second inhibitor arc forbids the return while there is a (k,b) token in CaccR for any line, it might have been restricted to the same line a but it is better to give a complete priority to all accesses. It is possible to have at the same time a (k,b,o) token into CPutR with o≠a, a (k,b,a,**C**) token into CgetR and a (k,b,a) token into CputR but no deadlock will occur: the replacement, the block getting the access and the write-back will occur in this order.

It is possible to avoid all these inhibitor arcs by using a unique FIFO place instead of CgetR and CputR: the presented solution allows to speed up the getting of the requested line and thus the processor access.

The last two transitions finish the put or get operations and release the cache memory unit.

- transition GgetE: A (k,b,**C**) token remains into Gget while the line content **C** is being gotten into the block b. After the access must be performed by the cache local part, thus a (k,b) token is placed into CaccR.
- transition GputE: A (k,b,a) remains into Gput while the block b content is extracted. Then a (k,a,C) token where C is the whole block content is placed into the FIFO place Cput that models the cache output buffers. It will then be transferred and put into the corresponding primary memory module.

## 4.4 Memory controllers

A controller Kx deals with all the REQUESTS and RETURNS from all caches but concerning only the lines contained in the module Mx. A request from one cache may need the controller to broadcast a set of QUERIES to all other caches. When a write-back either after a query or for replacement a RETURN and the line content are received from the owner cache. When a controller has the most recent version of a requested line it sends a GRANT with the line content to the requesting cache.

The cache and network must only deal with line addresses. Thus each line address a is also expressed as a=(â,ã) where the higher bits â determine the controller Kx such that x=a and lower bits ã give the relative line address inside Kx. Conversely a line address a=(â,ã) may be constructed from â =x and ã.

A controller is composed of two separate units: the control unit and the memory unit. We have split each control unit into a request analysis unit and a return analysis unit. These two control subunits may work in parallel but share informations on all the lines owned by this module.

- The place Kgs keeps the global states of all the module lines and insures a mutual exclusion, separately for each line, between request and return analyses. Kgs always contains a set of (x,ã,g) tokens for all the values of x and all the line numbers ã within each module. At the initialization all the lines are "Absent" from all caches (g=z).
- The place KreqW remembers what are the already accepted requests, this serves to accept only one request for a given line a at a time.
- Whenever a request is delayed until the end of a writing-back from the cache holding the line, the request unit places an (k,x,ã) token into the place KretW to remember the request parameters until the reception of the waited return.
- The FIFO-array KputR contains the (k,x,ã) tokens placed by the return unit to warn the memory unit for the write-backs.
- The FIFO-array KgetR contains the get orders indicated by (k,x,ã) tokens placed by the request unit if no write-back is needed or otherwise by the return unit.
- An x token is present respectively into KidleQ, KidleR and KidleM when the request, return and memory units are available.

As soon as a command from a cache is received into Kin, it is dispatched into one of the FIFO-array place Kreq or Kret according to its flag and the address is decomposed into upper and lower bits. The transition KSin removes a short request (S,k,a,t) from Kin and puts a (k,â,ã,t) into Kreq. The transition KLin removes a long return (L,l,o,C) deletes the now useless returning cache number l and puts an (ô,õ,**C**) token into Kret.

**Request acceptance**

When the request arlalysis unit is idle the transition KAreq may accept a command REQUEST(k,a,t) where k is the requesting cache number, a=â,ã is the line address, t is the desired access type. The requests received by the controller Kx are modeled by (k,x,ã,t) tokens into the FIFO-array place Kreq where distinct orders apply for each controller.

A memory controller can process concurrently several requests for different lines but two requests for the same line must be sequentially served. Hence as soon as the request unit accepts a new request it indicates by an (x,ã) token into the place KreqW that no other request for a may be accepted until the line transfer. The inhibitor arc from KreqW to KAreq allows to extract from Kreq the first (k,x,ã,t) token such that there is no (x,ã) token into KreqW and to move it towards the place KreqA.

**Request analysis**

Five cases are distinguished according to the request parameters and the line grant state given by the (x,ã,g) token in Kgs.

We firstly consider the two cases where g=w. The line a has been modified so the owner cache Cl must write-back the corresponding block. As this owner Cl is unknown, the controller broadcasts a query to all caches Ch for h belonging to the set **N**-k. This "broadquery" is denoted by a set (S,**N**-k,â,ã,u) of tokens into the FIFO place Kout where S indicates a short query message, u is the future valid bit imposed by Kx u=i for a purge or u=v for an update. The owner will know by inspecting the modified bit of its directory that it must send its copy of the line a. The controller will send a grant to Ck only after receiving the write-back. Hence a (k,x,a) token is put into KretW to wait for the corresponding return.

- transition KPquerW: If the REQUEST is a WRITE (t=w) the requesting cache must obtain the line a for it alone. Thus Kx set u=i. The new grant state in Kgs becomes "Absent" (g=z) until the purge return will be received.
- transition KuquerR: If the REQUEST is an UPDATE (t=r) the requesting cache may only obtain the line a in a sharing mode. Thus Kx sets u=v allowing the owner to keep the line. The new grant state becomes "PresentR" (g=r).

The three remaining cases concern a line not used for writing (g≠w) and thus no write-back is needed: an updated copy of the line a still remains into the module. Thus a GRANT may be immediately sent to Ck and a (k,x,ã,g) token in KgetR notifies the memory unit to promptly send the GRANT with the line a to Ck. The controller must check whether an invalidation is needed or not.

- **t**ransition KIquerW: In case of g=r and a REQUEST with t=w from Ck, the controller broadcasts, perhaps uselessly, a QUERY for invalidation to all the caches of the set **N**-k. This query is denoted by a set (S,**N**-k,x,ã,i) of tokens into Kquer. All the owner caches will find a modified bit set to r and will only reset the associated valid bit to i without sending any return. This avoids the network congestion by useless answers. The new grant state Kgs becomes g=w for "PresentW" since Ck will be the unique owner of the line a.

  When g=r the grant may be in fact overvaluated (perhaps the line a is not used at all and so g=z would be better): because of the two-bits encoding of Kgs the controller is unable to exactly count the number of owners of a line but this may cost only some superfluous invalidation requests.

  After an invalidation Kx does not wait for an acknowledgement: the grant may be sent immediately but remains serialized after the invalidation into the FIFO-array places Kout and Cin.

The broadcast will keep the whole interconnection network busy, thus any grant towards any cache as well for this request or for a logical synchronization (implying a write) will be postponed.

The controller may avoid to broadcast a QUERY for invalidation in the two important cases of a READ REQUEST or of a line certainly absent.

- <u>transition KNauerW:</u> When g=z the line a is absent even for a WRITE request (t=w) the controller has no query to broadcast, The new grant state (x,â,g) in Kgs becomes "PresentW".
- <u>transition KnauerR:</u> In case of a READ REQUEST (t=r) whatever g=z or g=r, there is again no query to broadcast. Ck will become a new reader so the global state (x,ã,g) is always set to "PresentR" (g=r).

**Return acceptance**

When the return analysis unit of Kx is idle a command RETURN (ô,õ,**C**) may be accepted: as usual x=ô, õ is the address of the line, **C** is the line content and the number of the sending cache does not need to be transferred since the line content **C** itself is inside the command.

Some returns are **awaited** answers to previously broadcast queries in which case there is a (k,x,ã) token in KreqW precisely for the same line ã=õ. Most of these awaited returns have been sent by the cache global parts because of an update or purge query but some, originated by the cache local parts for replacements, must be considered as an anticipated answer to a query sent in parallel. This unification of returns whatever their cause is very important to avoid deadlocks and in any cases the cache directory has been coherently set. Others returns for replacements remain unwaited.

The return modeled by the first token (ô,õ,**C**) of the FIFO-array place Kret is accepted by the <u>transition KAret.</u> The token is moved to KretA.

**Return analysis**

This analysis distinguishes three cases according to the global state (x,õ,g) of the returned line x,õ and places an (x,õ,**C**) token into KputR for ordering the memory unit to store the line content.

For an awaited return there is no matter if a replacement purge occurs when only an only update was awaited because the cache owner count is not managed. The (k,x,ã) token into KretW with ã=õ remembers the requesting cache. A (k,x,ã,g) token may be put into KgetR for ordering the memory unit to send the GRANT and the line content to the requesting cache. This unit will correctly send the grant only after the writing-back.

- <u>transition KretR:</u> If g=r, the return is a purge or update awaited after a READ request. As the requesting cache becomes either the unique or a supplementary owner the global state always becomes "PresentR" (g=r).
- <u>transition KretW:</u> If g=z, this is an awaited purge return after a WRITE request. The requesting cache Ck becomes the unique owner of the line o: the state becomes "PresentW" (g=w),
- <u>transition Kunwret:</u> If g=w this may only be an unwaited purge return for a replacement at the initiative of Cl while an update was waited by Kx as an answer to its query. There is no need to check by an inhibitor arc that the return is unwaited because g=w only in this case. As the sending cache was the unique owner of the line the state is now "Absent" (g=z).

**Data transfers**

The last two steps concern the supervision of transfers by the **controller memory unit** either to **put** into the module a line content written-back by some cache or to send to a cache a line copy that it will **get** into the associated block. These operations are not atomic thus we have explicitly shown their beginings and ends between which the controller memory unit becomes busy (an x token is removed from KidleM).

- transition KgetB: A (k,x,ã,g) token in KgetR orders to send the content of the line a=x,ã to the requesting cache Ck, however the transfer must not begin if there remains a write-back to do for the same line. Such a write-back is indicated by a (x,õ,**C**) token anywhere into the FIFO place KputR: so this condition must be checked by an inhibitor arc issuing from KputR with ã=õ. This inhibition also applies when, because of the memory load, a return waits into KputR and after the request unit orders directly a grarlt in KgetR.
- transition KDutB: A returned line o=ô,õ is modeled by a (ô,õ,**C**) token in KputR where **C** denotes the line content to put into the module. This operation always possible will allow further line sendings to caches avoiding deadlocks.

It would be possible to avoid the inhibitor arc by merging the places KputR and KgetR but this would remain less possibilities to speed up the grants.

The last two transitions terminate the data management.

- transition KgetE: A (k,x,ã,g) token remains into Kget while the line content is extracted. Then the GRANT, the line address and its content **C** must be transferred by the network to the requesting cache Ck, thus a (L,k,x,ã,**C**) token for a long message is placed into the FIFO ouput buffer Kout.
- transition KputE: An (x,õ,**C**) token remains into Kput while the content of the line x,õ is written-back into the memory module.

## 5 Validation

The Predicate/Transition net has been designed in order to keep elementary invarlants corresponding to the work of each unit, to the correct structuration of each block and line state and the management of each request.

To simply express complex invariants we need concise notations. If P is a place of color domain D and c is a set of colors belonging to this domain D, let P\c denote the number of tokens of color c in P. This may be extended if D is a product of component dornains and c is a set of compound colors.

For sake of simplicity we assume that all elementary domains may be distinguished by adequate renamings. If D is a product of distinct component domains and D' is a subdomain of D i.e. a product of only some of components of D, may be in a different order, let proj(P/D') denote the bag of tokens of P, restricted to the D' components and reordered with respect to D'. If c is a set of colors in D', let P\c = proj(P/D')\c. We note (P + Q)\c for P\c + Q\c. These notations allow to follow partial token information diversely combined in various places.

The following invarlants take into account the conventional initial markings without processor pipe-lining.

For each processor color k ∈ **N:**

- (Pr + Pw) \k = 1

  There is one unique token of color k either in Pr or in Pw.
- (Pr + Binstp + BinstD) \k = 1

  There is one token of color k in Pr or one (k,*,*,*) token in Binstp or one (k,*) token in BinsD.

Thus the place Pw, useful to explain that the processor is waiting for its result, is in fact redundant.

- LinstW\k = (BinstD + Pr + BinstR)\k

  This indicates the place LinstW is redundant. With pipe-lining it serves to bound the number of orders accepted by a cache independently of the look ahead possibilities of the processor.

For each cache color k ∈ **N:**

- (CidleC + LinstA + Lrep + GquerA + GgrantA) \k = 1

There is one token of color k in CidleC or one compound token with a k component in on of the places of the cache control unit.

- (CidleM + Gput + Gget) \k =1

There is one k token or component in the places of the cache memory unit.

For each cache k and each block b:

- Cdirvm \ (k,b) = 1

There is one (k,b,*,*) token for each block of each cache.

For each controller color x ∈ **M:**

- (KidleQ + KreqA) \x = 1
- (KidleR + KretA) \x = 1
- (KidleM + Kput + Kget)\x =1

For each controller color x ∈ **M** and each line color ã ∈ **A/M:**

- Kgs\ (x,a) =1

There is one (x,a,*) token for each line of each module.

More interesting invariants describe the request and return treatments involving communications between caches and controllers. We have shown their supports on the last figures. For instance we obtain for the requests:

For any cache and processor k:

- (Pr + BinstR + LinstA + Lrep + Creq +
  Kreq + KreqA + KretW + KgetR + Kget + Kgrant +
  Cgrant + GgrantA + CgetR + Gget + CaccR + BinstD)\k = 1

There is either one k token in Pr or one (k,*,*) in BinstR, LinstA, Lrep, Creq or one (k,*,*,*) in Kreq, KreqA, KgetR, Kget or one (k,*,*) in KretW or one (k,*,*,*,*) in Kgrant or one (k,*,*,*) in CGrant, CgrantA, CgetR or one (k,*,*) in Gget or one (k,*) in CaccR, BinstD.

This means that the processor is idle or its order is managed by its own cache or by some controller.

- Pr\k + (BinstR + L instA + Lrep + Creq)\(k,a) +
  (Kreq + KreqA + KretW + KgetR + Kget + Kgrant)\(k,â,ã) +
  (Cgrant + GgrantA + CgetR)\(k,a) + (Gget + CatcR + BInstD)\k = 1

This more precise invariant expresses the tact that when the processor k is not idle a request for the same a is being managed. Moreover the transition LvalhitA keeps the invariant by removing a (k,a,*) token in LinstA and putting a (k,b) one in CaccR but checks that there is a (k,b,a,*,*) token in Cdirvm. The transitions LvalhitM, LinvhitRW, LinvrepN, LvalrepF, LvalrepP move a (k,a,*) token towards Creq but insure also that there is a (k,b,a,*,*) token in Cdirvm and the transition GRWM does not change this token but put a (k,b,*,*) token in CgetR that is moved in Gget and then CaccR without any change of Cdirvm
A more precise study would allow to check that the access type is correctly transmitted. The crucial point concerns the transitions KPquerW and KUquerR that remove a (k,â,ã,t) token in KreqA for putting an incomplete (k,â,ã) token in KretW and an (â,ã,g) token in Kgs with respectively g=z or r for t=w or r; but the transitions KretW and KretR restore the matching. Thus the g component into KgetR, Kget, Kgrant, Cgrant, GgrantA and then Cdirvm always matches with the ordered t.

- KreqW\(x,a) = (KreqA + KretW + KgetR)\(x,a)
  This shows that the place KreqW is redundant, however its removal would introduce several inhibitor arcs.

To study returns it will be necessary to distinguish invalidations from other queries as well as awaited returns from unwaited ones. Such properties need too much behavioral study thus we are choosing another way that is to introduce redundant service places and tokens in order to take more advantage of the invariants. We are also applying with success colored net transformations to reduce the net size [Berthelot 85].

## 6 Conclusion

The Petri net modeling of the protocols is very valuable in understanding the difficulties that can arise in the synchronization and exchanges of messages and to obtain behavioral properties. The model could be used as an entry point for a simulation program, a Stochastic Petri net [Florin 85, Marsan 85] or a TImed Petri net [Chretienne 84].

Slight modifications of the underlying Petri net model may allow to study a spectrum of solutions, each corresponding to a pair (Memory Module Controller/Cache global part). The current model assumes that messages always arrive at their destination but it could be expanded to include acknowledgements and time-out mechanisms in order to extend the protocol to distributed multiprocessors.

## References

Archibald,J. and Baer,J-L. "An economical solution to the cache coherence problem." Proc of 11 th Int. Symp. on Computer Architecture, IEEE, 1984,pp. 355-362,

Archibald,J. and Baer,J-L, "An evaluation of cache coherence solutions in shared-bus multiprocessors." Technical report 85-10-05, University of Washington, Seattle, October 1985,

Auguin,M, and Boeri,F. "Etude comparative de reseaux d'interconnection dans une archltecture MIMO". Congres sur les nouvelles architectures pour les communications, Paris (sept 1984)

Baer,J-L, and Girault,C. "A Petri net solution for the cache coherence problem." Proc of 1 rst Int. Conf. on Supercomputing Systems, St Petersbourg Florida (December 1985), IEEE 85CH2216-0, pp680-689.

Baer,J-L. and Girault,C. "Design of a parallel architecture for a solution to the cache coherence problem," Parallel computing, Berlin (September 1985, to be published by North Holland).

Berthelot,G. "Analyse de processus parallèles par transformation de réseaux de Petri"AFCET, TSI ,vol 4 n° 1, Janvier 1985, pp 73-82,

Berthelot,G. and Terrat,R, "Petri nets for the correctness of protocols." IEEE trans on Communications 30,n.2 (Dec. 1982).

Brams,G.W, "Reseaux de Petri: theorie et pratique." Masson ed. vol 1 and 2, Paris 1982 and 1983.

Carlier,J, Chretienne,Ph, and Girault,C. "Modeling scheduling problems with timed Petri nets" 4th. European workshop on application and theory of Petri nets, Toulouse (September 1983),

Censier,L,M. and Feautrier,P, "A new solution to coherence problems in multicache systems." IEEE TCC-27,12 (Dec 1978),pp. 1112-1118,

Chi Yuan Chin and Kai Hwang, "Connection principles for multipat packet switching networks" Proc of 11th Int. Symp. on Computer Architecture, IEEE, June1984, pp, 99-108.

Chretienne,P, "Executions controlées des réseaux de Petri temporisés" AFCET TSI, vol 3 n° 1, Janvier 1984, pp 23-31.

Diaz,M "Petri net based models for the specification and validation of protocols", 5th, European workshop on application and theory of Petri nets, Aarhus (June 1984).

Finkel,A. and Memml,G. "FIFO nets: a new model of parallel computation", 6th G.I. conference on theoretical computing, Dortmund (January 1983).

Florin,G, and Natkin,S, "Les réseaux de Petri stochastiques." AFCET, TSI ,vol 4 n° 1, Janvier 1985, pp 143-160.

Gajski,D, Kuck,D., Lawrie,D, and Sameh,A, "CEDAR' a large multiprocessor." Computer Architecture News 11,1 (March 1983),pp. 7-11.

Genrich,H.J. and Lautenbach.K. "The analysis of distributed systems by means of predicate/transition nets" semantics of concurrent computation, Lecture Notes in Computer Science n° 70, Springer Verlag 1979.

Genrich,H.J., Lautenbach,K. and Thiagarajan.P.S "Elements of general net theory" in "Net theory and applications", Brauer,W. ed., Lecture Notes in Computer Science n° 84, Springer Verlag 1980.

Gottlieb,A., Grishman,R., Kruskal,C.P., Mc Auliffe,K.P., Rudolph,L. and Snir,M. "The NYU ultra computer: Designing an MIMD shared memory parallel computer." IEEE TCC- 32,2 (Feb. 1983),pp, 175-189, Haddad,S. and Bernard,J.M. "ARP a software for specification and validation of protocols and distributed applications", 3rd Conference-Exhibition on Software engineering, AFCET, Versailles, May 1986.

Haddad,S and Girault.C. "Algebraic structure of flows of a regular Colored Petri Nets". 7th European workshop on appl ication and theory of Petri nets, Oxford (June 1986).

Huber,P, Jensen,AM., Jensen,L.O. and Jensen,K. "Towards Reachabillty Tree for high-level Petri Nets". 5th. European workshop on application and theory of Petri nets, Aarhus (June 1984)

Jensen,K. "Coloured petri nets and the invariant method" T.C.S. 14,n° 3, North Holland pub.,(June 1981).

Katz,R.H., Eggers,S.J., Wood,D.A., Perkins,C.L. and Sheldon,R.G. "Implementing a cache consistency protocol" Proc of 12th Int. Symp. on Computer Architecture, IEEE, Boston, June 1985,pp. 276-283.

Kujansuu,R. and Lindqvist,M "Efficient algorithms for computing S-invariants for predicate/transition nets" 5th. European workshop on application and theory of Petri nets, Aarhus (June 1984).

Kludge,W.E. and Lautenbach,K. "The orderly resolution of memory access conflicts among competing channels" IEEE trans on Computers 31 ,n° 3 (March 1982).

Marsan,M.A., Chiola,G. and Conte,G. " Generalized stochastic Petri net models of multiprocessors with cache memories" Proc of First Int. Conf. on Supercomputing Systems, St Petersbourg Florida (December 1985), IEEE 85CH2216-0, pp690-696.

Memmi,G. "Méthodes d'analyse de réseaux de Petri,réseaux à files et applications aux systemes en temps réel." ,Thèse d'état, Université Paris 6, June 1983.

Papamarcos,M. and Patel,J. "A Low Overhead Coherence Solution for multiprocessors with Private Cache Memories" Proc of 11th Int. Symp. on Computer Architecture, IEEE, June 1984,pp. 348-354.

Peterson,J.L. "Petri net theory and the modeling of systems", Prentice Hall, 1981.

Rudolf,L. and Segall,Z. "Dynamic Decentralized Cache Schemes for MIMD Parallel Processors". Proc of 11th Int. Symp. on Computer Architecture, IEEE, 1984,pp. 340-347

Siegel,H.J. "Interconnection networks for large-scale parallel processing." Lexington Books, 1985.

Smith,A.J. "Cache memories." Computing Surveys 143 (Sept. 1982),pp. 473-530.

Smith,A.J. "Cache evaluation and the impact of workload choice" Proc of 12th Int. Symp. on Computer Architecture, IEEE, Boston, June 1985,pp. 276-283.

Vautherin,J. and Memmi,G. "Computation of flows for unary predicate/transition nets" 5th. European workshop on application and theory of Petri nets, Aarhus (June 1984).

Vautherin,J. "Non linear Invariants for safe coloured Petri nets and applicatlon to the proof of parallel programs." 6th. European workshop on application and theory of Petri nets, Espoo, Finland (June 1985).

Yen,W.C., Yen,D.W.L. and King-Sun Fu "Data Coherence Problem in a Multlcache System". IEEE TCC-34;1 (Jan 1985).

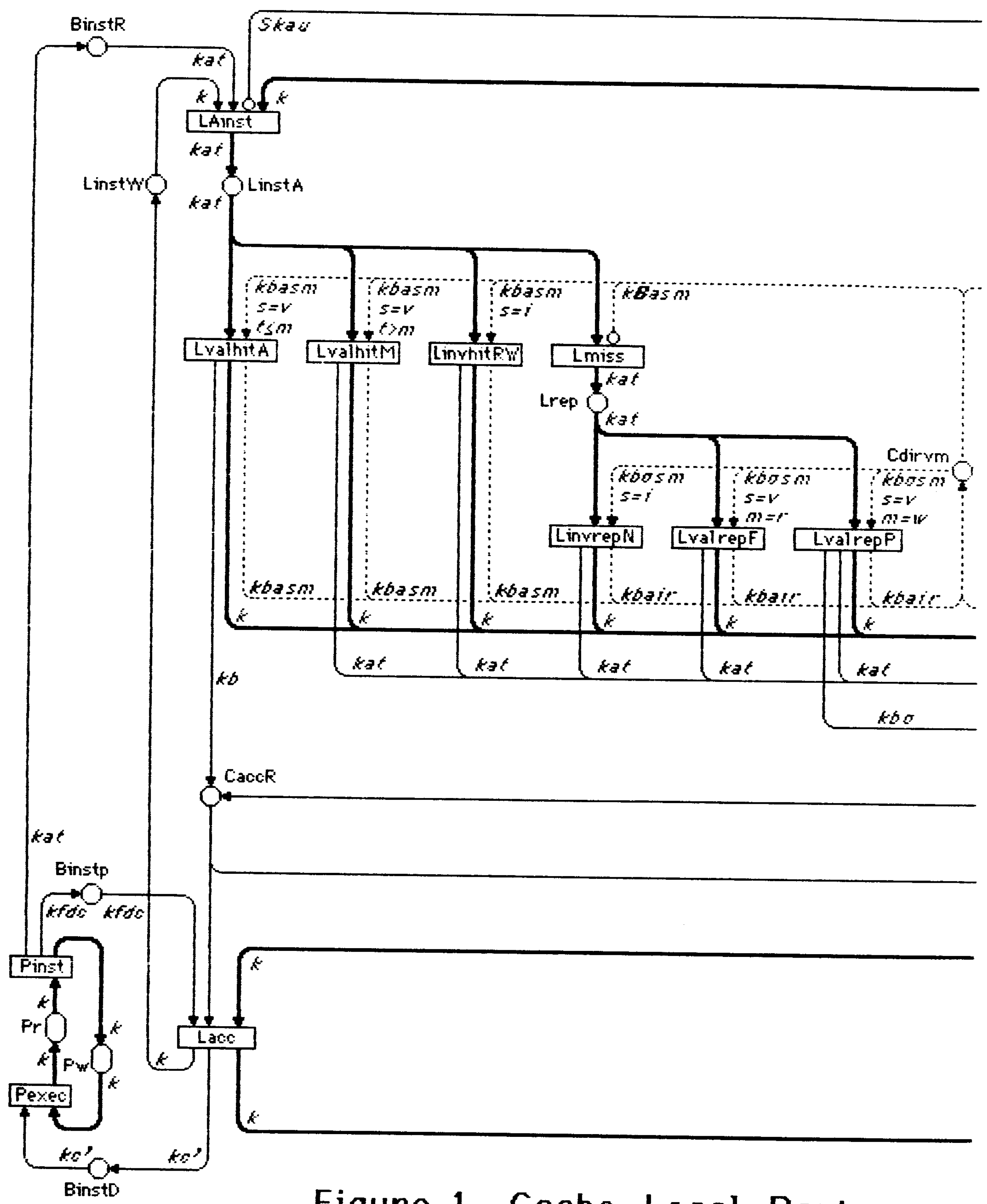

Figure 1. Cache Local Part

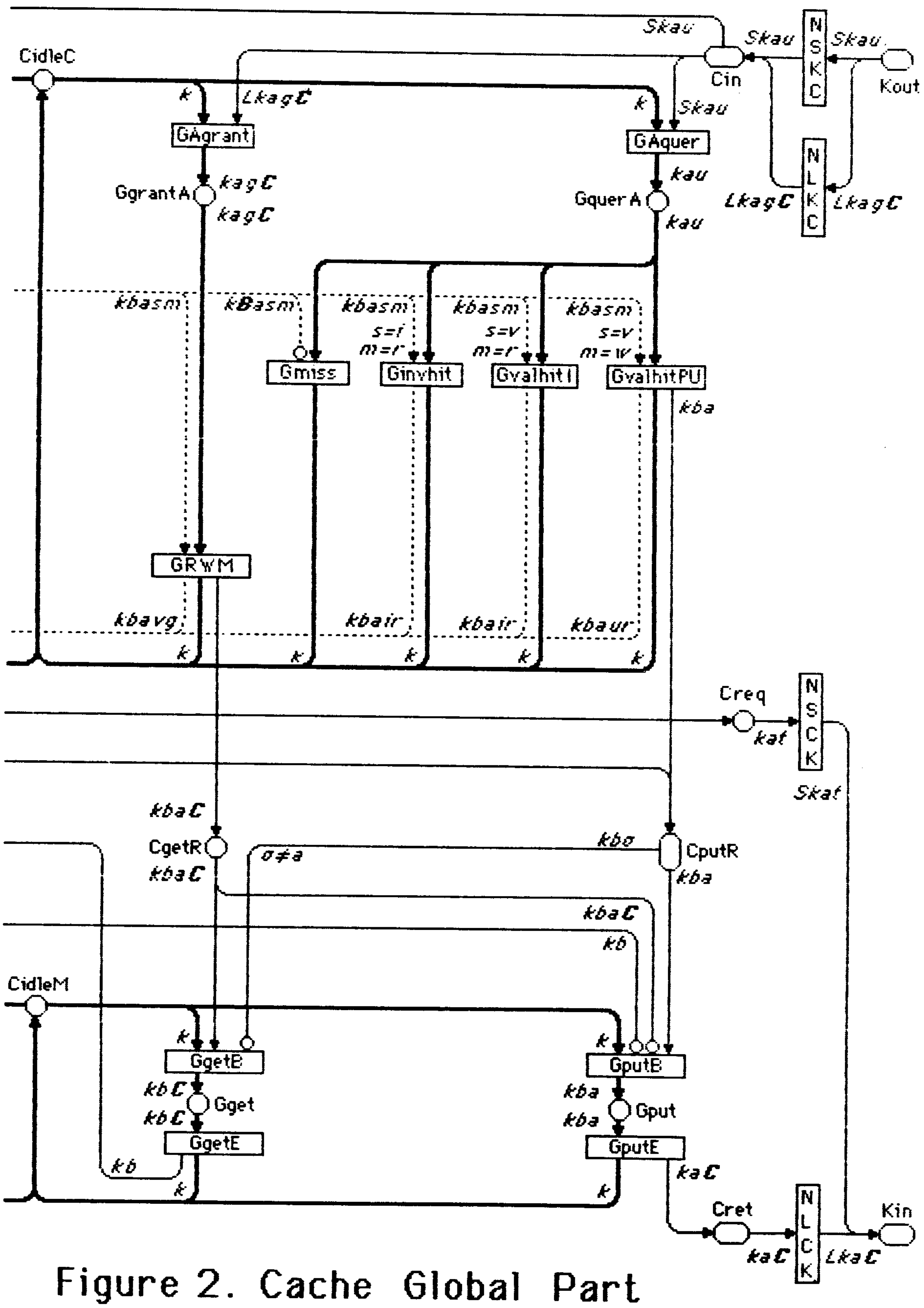

Figure 2. Cache Global Part

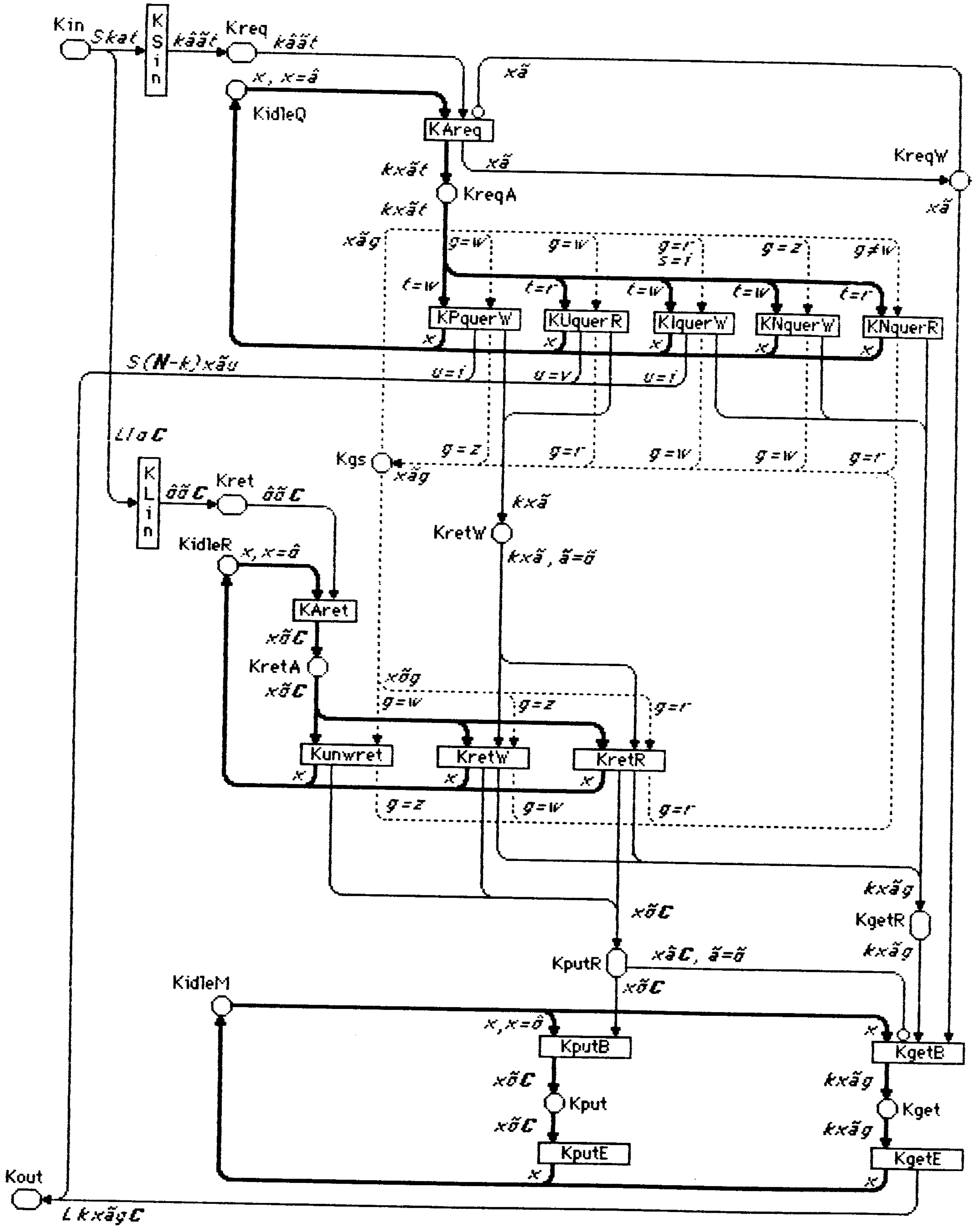

Figure 3. Memory Controller