Decoupled Interconnection of Distributed Memory Models*

Ernesto Jiménez[1], Antonio Fernández[2], and Vicente Cholvi[3]

[1] Universidad Politécnica de Madrid, 28031 Madrid, Spain
ernes@eui.upm.es
[2] Universidad Rey Juan Carlos, 28933 Móstoles, Spain
afernandez@acm.org
[3] Universitat Jaume I, 12071 Castellón, Spain
vcholvi@inf.uji.es

Abstract. In this paper we present a framework to formally describe and study the interconnection of distributed shared memory systems. In our models we minimize the dependencies between the original systems and the interconnection system (that is, they are decoupled) and consider systems implemented with invalidation and propagation.
We first show that only fast memory models can be interconnected. We then show that causal and pRAM systems can be interconnected if they fulfill some restrictions, and for these cases, we present protocols to interconnect them. Finally, we present a protocol to interconnect cache systems.

1 Introduction

Distributed shared memory is an abstraction used for process communication. In this abstraction, processes read and write variables of a shared memory, which is usually implemented with distributed memory and message passing. Depending on the semantics of the shared memory a number of consistency models have been proposed in the literature [1,9]. Some of the most popular models are the sequential [16], causal [3], pRAM [18], and cache [12]. Informally, the sequential model requires that the read and write (memory) operations obtained in an execution of a distributed system could have been obtained if they had been executed in a single processor. Therefore, in this model there must be a total order (a *view*) of the operations such that they seem to have been executed in that sequential order. This sequential view must be the same for every process. The causal model relaxes the memory semantics because it allows several views (one for each process), where only causally related memory operations must be ordered. Therefore, two processes could have different causal views if there are operations that are not causally related. Similarly, the pRAM model is a

* This work has been partially supported by the Spanish MCyT under grants TIC2001-1586-C03-01 and TIC2001-1586-C03-02, the Comunidad de Madrid under grant 07T/0022/2003, and the Universidad Rey Juan Carlos under grant PPR-2003-37.

M. Papatriantafilou and P. Hunel (Eds.): OPODIS 2003, LNCS 3144, pp. 235–246, 2004.

relaxation of the causal model, because it only requires that in each process all write operations of another process seem to have been executed in the same order as they were issued. Finally, the cache model is like the sequential model but applied on each variable independently. That is to say, there is a sequential view formed by all operations on variable x (for every variable x of the shared memory). Many protocols have been proposed in distributed shared memory systems that implement these consistency models.

In this work we study the interconnection of distributed shared memory systems. By this we mean the adding of an *interconnection system* to several existing distributed shared memory systems that implement a given consistency model to obtain a single distributed shared memory system that implements the same consistency model. This line of work was started in [10], where the interconnection of causal propagation-based systems was studied. In this work we use much weaker assumptions on the systems to be interconnected, consider also invalidation-based systems, and explore other models as well.

Our Contributions. In this work we first define a framework for the interconnection of systems. We formalize several classes of interactions, both for propagation and invalidation-based protocols, between the existing systems and the interconnection system. All these classes decouple the systems from the interconnection system, unlike the previous model [10].

Then, we show that systems that implement *non-fast* consistency models cannot be interconnected in these classes. A fast consistency model is one that allows implementations of read and write operations that return control after only local computations. After that, we study the interconnection of pRAM and causal systems. We show that they can not be interconnected in general, but can under certain restrictions. We give sufficient conditions and the corresponding protocols to do so. Finally, we also show that systems that implement cache consistency can always be interconnected.

Note that this is the first work that studies the interconnection of pRAM and cache systems, that considers both propagation and invalidation, and that shows that certain interconnections are in fact impossible. Our protocols need not be very useful nor efficient in practice, since all we try to do with this work is to define the bounds of the possibilities of interconnection.

The rest of the paper is organized as follows. In Section 2 we introduce our framework for the interconnection of systems. In Section 3 we show the impossibility result for non-fast consistency models. In Section 4 we study the interconnection of pRAM systems, in Section 5 the interconnection of causal systems, and in Section 5 we show how to interconnect cache systems. Finally, in Section 7 we present concluding remarks.

2 Definitions and Notation

We consider *distributed shared memory systems* (or *systems* for short) formed by a collection of *application processes* that interact via a *shared memory* formed

by a set of *variables*. All the interactions between the application processes and the memory are done through read and write operations (*memory operations*) on variables of the memory.

Each memory operation is applied on a named variable and has an associated value. A write operation of the value v in the variable x, denoted $w(x)v$, stores v in the variable x. A read operation of the value v from the variable x, denoted $r(x)v$, reports to the issuing application process that the variable x holds the value v. To simplify the analysis, we assume that a given value is written at most once in any given variable and that the initial values of the variables are set by using fictitious write operations.

Consistency Model of a System. An *execution* α of a system S is the set of read and write operations observed in some run R of system S.

Definition 1 (Process Order). *Let p be a process of S and $op, op' \in \alpha$. Then op precedes op' in p's* process order, *denoted $op \prec_p op'$, if op and op' are operations issued by p, and op is issued before op'.*

Definition 2 (Execution Order). *Let $op, op' \in \alpha$. Then op precedes op' in the* execution order, *denoted $op \prec op'$, if:*
1. op and op' are operations from the same process p and $op \prec_p op'$, or
2. $op = w(x)v$ and $op' = r(x)v$, or
3. $\exists op'' \in \alpha : op \prec op'' \prec op'$

We denote by α_p the subset of operations obtained by removing from α all read operations issued by processes other than p. We also denote by $\alpha(x)$ the subset of operations obtained by removing from α all the operations on variables other than x.

Definition 3 (View). *Let $\prec^o$ be an order on execution α, and let $\alpha' \subseteq \alpha$. A* view *β of α' preserving $\prec^o$ is a sequence formed by all operations of α' such that this sequence preserves the order $\prec^o$.*

Note that if $\prec^o$ applied on α' is not a total order, there can be several views of α'. We use $op \xrightarrow{\beta} op'$ to denote that op precedes op' in a sequence of operations β. We will omit the name of the sequence when it is clear from the context. We will also use $\beta_1 \rightarrow \beta_2$, where β_1 and β_2 are sequences of operations, to denote that all the operations in β_1 precede all the operations in β_2.

Definition 4 (Legal View). *Let $\prec^o$ be an order on execution α, and let $\alpha' \subseteq \alpha$. A view β of α' preserving $\prec^o$ is* legal *if $\forall r(x)v \in \alpha'$:*
a) $\exists\, w(x)v \in \alpha' : w(x)v \xrightarrow{\beta} r(x)v$, and
b) $\nexists\, w(x)u \in \alpha' : w(x)v \xrightarrow{\beta} w(x)u \xrightarrow{\beta} r(x)v$.

By using this definition of legal view, we can define systems satisfying the causal, pRAM, and cache consistency models.

Definition 5 (Causal, pRAM, or Cache System). *A system S is causal if for every execution α and every process p there is a legal view β_p of α_p preserving $\prec$ on α. A system S is pRAM if for every execution α and every process p there is a legal view β_p of α_p, preserving $\prec_q$ on α_p, for all q. A system S is cache if for every execution α and every variable x there is a legal view β_x of $\alpha(x)$ preserving $\prec$ on $\alpha(x)$.*

System Architecture. From a physical point of view, we consider distributed systems formed by a set of *nodes* and *a network* that provides communication among them. The essence of this model has been taken from [6]. The application processes of the system are actually executed in the nodes of the distributed system. We assume that the shared memory abstraction is implemented by a *memory consistency system* (*MCS*). The MCS is composed of *MCS-processes* that use the local memory at the various nodes and cooperate following a distributed algorithm, or *MCS-protocol*, to provide the application processes with the impression of having a shared memory. The MCS-processes are executed at the nodes of the distributed system and exchange information as specified by the MCS-protocol. They use the communication network to interact if they are in different nodes. Each MCS-process can serve several application processes, but an application process is assigned to only one local MCS-process. For each application process p we use $mcs(p)$ to denote its MCS-process.

An application process sequentially issues read or write operations on the shared variables by sending (read or write) *calls* to its MCS-process. After sending a call, the application process blocks until it receives the corresponding *response* from its MCS-process, which ends the operation.

We consider MCSs implemented with *propagation* and *invalidation*. For simplicity, in both cases we consider that each MCS-process has a copy (replica) of the whole shared memory. In an MCS with invalidation, some of the copies of a variable x can be "invalid" or outdated. If an MCS-process' copy of a variable x is invalid and one of its application processes tries to read x, the MCS-process has to obtain the current value of x from some other MCS-process (following the MCS-protocol). When an application process issues a write operation $w(x)v$, the local copy of x in its MCS-process is updated with the current value v, and the valid copies of x in the rest of MCS-processes are marked as invalid. In an MCS with propagation no copy is ever invalid and holds the current value (as seen by the MCS-process). This value is returned to an application process that issues a read operation. New written values are propagated among MCS-processes to maintain the copies up to date.

The Interconnection System. This paper deals with the interconnection of systems. This means that, after the interconnection, the set of original systems will behave as one single system. Using the terminology defined above, interconnecting systems is, in fact, interconnecting MCSs. In our model, the load of such an interconnection will fall on an *interconnection system* (IS). An IS is a set of processes (*IS-processes*) that execute some distributed algorithm or protocol

(*IS-protocol*). For simplicity in the IS design, we consider one IS-process for each MCS to be interconnected. The IS-process of each system is at the same level as an application process and has its own MCS-process. The IS-process uses the MCS-process to read and write on the shared memory of the local system. In particular, the only way a value written by an application process in some system can be read by an application process in another system is if the IS-process of the latter system writes it. IS-processes exchange information among them (as specified by the IS-protocol) by using a communication network. Note that, after the interconnection, the overall system has a *global MCS* formed by the MCSs of the original systems plus the IS that interconnects them.

For system interconnections we extend the interface between the MCS and the IS beyond read and write operations issued by IS-processes. We assume that MCS-processes are connected with its corresponding IS-process through a reliable FIFO channel and send messages to it with the changes on the local memory replicas by using these FIFO channels. We consider the following classes of interfaces between the MCS and the IS.

(a) **Weak decoupled class with propagation (WDP).** The MCS-process of the IS-process sends a message to the IS-process every time a variable copy is updated. Each of these messages, denoted by $msg(p, x, u)$, carries the process p that issued the corresponding write operation, the variable x, and the new value u.

(b) **Strong decoupled class with propagation (SDP).** Every MCS-process in the system sends a message to its corresponding IS-process every time a variable copy is updated. Each of these messages, denoted by $msg(p, m, x, u)$, carries the application process p that issued the corresponding write operation, the MCS-process m that updated, the variable x, and the new value u. Trivially, in the SDP class, the IS-process receives at least as much information as in the WDP class. Thus, in this sense, SDP is stronger than WDP.

(c) **Strong decoupled class with invalidation (SDI).** Every MCS-process in the system sends a message to its corresponding IS-process every time a variable copy is invalidated or updated (by a write operation issued by one of its application processes). Each of these messages, denoted by $msg(p, q, x, u)$, carries the process p that issued the corresponding write operation, the MCS-process q that updated or invalidated this replica, the variable x, and the new value u (if it is an update). For each write operation $w(x)u$ issued by some process p, the IS-process will receive an update message $msg(p, mcs(p), x, u)$ from p's MCS-process, and an invalidation message $msg(p, m, x)$ from each MCS-process m that had a valid copy of x, and has invalidated it.

Model and Notation. In this paper we assume an asynchronous model. This means that there is no bound on the time instructions and message transmissions take. We do not assume synchronized clocks among processes. We also assume that no system component (processes, nodes, and networks) fails.

In the rest of the paper we will use N to denote the number of systems to be interconnected. The systems to be inteconnected will be denoted by

$S^0, \cdots, S^{N-1}$, and the resulting interconnected system by S^T. The IS-process for each system S^k (where $k \in \{0, \cdots, N-1\}$) is denoted by isp^k. It is worth to remark that isp^k is part of the system S^k. For that reason, the MCS-process $mcs(isp^k)$ has a local replica of each variable of the shared memory, and those replicas are updated or invalidated (depending on the method used to maintain the coherence of these replicas) following the MCS-protocol implemented in the MCS of S^k. We also assume that the IS-processes are interconnected among them through reliable FIFO communication channels which will be used to propagate write operations from one system to the other. We consider that the set of processes of S^T includes all the processes in $S^0, \cdots, S^{N-1}$ except for $isp^0, \cdots, isp^{N-1}$ (they are only used to interconnect the systems $S^0, \cdots, S^{N-1}$).

Regarding executions, we will use the next notation. α^T will denote an execution of S^T. Similarly, α^k (where $k \in \{0, \cdots, N-1\}$) will denote the execution of S^k obtained in the same run. Note that α^k and α^T have in common all the operations issued by processes in S^k. We also extend the notation used with read and write operations. We denote by $w_p^k(x)v$ the write operation $w(x)v$ issued by process p of system S^k. Similarly, we denote by $r_p^k(x)v$ the read operation $r(x)v$ issued by process p of system S^k. A write operation $w_q^l(x)v$ in α^T issued by some processes q in S^l appears in α^k, $k \neq l$, as the write operation $w_{isp^k}^k(x)v$ issued by the process isp^k in S^k. This is so because *every* write operation issued by isp^k in α^k is, from the IS-protocol, just the propagation of a write operation issued by a process of another system S^m, $m \neq k$. We denote by $orig(op)$ the original write operation propagated as operation op in α_p^k by process isp^k. Similarly, given a write operation op issued in S^l, $l \neq k$, we denote by $prop(op)$ the write operation issued by isp^k as a result of propagating op to S^k as defined by the IS-protocol.

We will say that a consistency model can be interconnected if there is an IS-protocol that interconnects systems implementing this consistency model. The IS-protocol can specify the number of systems it interconnects. (However, it cannot restrict how applications processes are mapped to nodes.)

3 Non-fast Consistency Models

In this section we show that systems implementing "non-fast" consistency models can not be interconnected in any of the classes defined in the previous section. We say that a consistency model is *fast* if there is an MCS–protocol that implements it, such that memory operations only require local computations before returning control, even in systems with several nodes. There is a number of consistency models (e.g., causal or pipelined RAM) that are fast, while there are stronger memory models (e.g., the sequential or atomic) that are not. This implies that the property of being fast classifies the set of memory models in a non trivial way.

The proof of the following theorem is omitted due to space limitations.

Theorem 1. *There is no IS that guarantees the interconection of systems implementing some non-fast memory model.*

As a consequence of this theorem, we can derive that a number of popular memory models can not be interconnected. In [6] it is shown that the sequential consistency model is not fast. Hence it cannot be interconnected and neither can the atomic consistency model, nor its derivations, safe and regular [17]. Similarly, Attiya and Friedman [4] have shown that the processor consistency models PCG and PCD [12,2] are not fast and hence cannot be interconnected. Finally, in [4] Attiya and Friedman also proved that any algorithm for the mutual exclusion problem using fast operations must be cooperative. This implies that any synchronization operation that guarantees mutual exclusion must be non–fast. Therefore, any synchronized memory model that provides exclusive access cannot be interconnected. As a result, we have that memory models such as the *eager release* [11], the *lazy release* [15], the *entry* [7] or the *scope* [13] can not be interconnected.

4 pRAM Consistency Model

In this section we study the interconnection of pRAM systems. We first show that in general the interconnection is not possible. Then present IS-protocols for the different classes defined under different restrictions.

Impossibility for Interconnecting pRAM Systems. In this subsection we show that we can not guarantee the interconnection, through some IS in any class, of every pair of pRAM systems. The proof of the following theorem is based on the fact that, when some process p in S^k, $k \in \{0, 1\}$, issues several write operations, it may update the corresponding variables in its local memory in a different order from $p's$ process order.

The proof of the following theorem is omitted due to space limitations.

Theorem 2. *There is no IS in SDP that guarantees pRAM interconnection for every pair of pRAM systems.*

We know, by definition, that SDP is stronger than WDP. Hence, we can also apply this impossibility result to ISs interconnecting pairs of pRAM systems in WDP. Note also that the above proof can be easily adapted to the SDI class. Hence, we have that there is no IS in SDI that guarantees pRAM interconnection for every pair of pRAM systems.

IS-protocols for pRAM Systems. In this section we show how to interconnect systems implementing the pRAM [18] consistency model in SDP, WDP, and SDI, as long as these systems satisfy certain restrictions. First, we present an IS-protocol in SDP, for MCSs that satisfy the following property, which is fulfilled by all pRAM MCSs we have found.

Property 1. In any computation α^k of system S^k ($k \in \{0, \cdots, N-1\}$), for each process p in S^k, there is an MCS-process $s(p)$, known by isp^k, such that if p issues two write operations $w_p^k(x)v \prec_p w_p^k(y)u$, then $s(p)$ updates its local replica of x with the value v before updating the variable y with the value u.

Property 1 guarantees that each process isp^k knows MCS-processes in system S^k that locally apply the write operations in the same order as they were issued in S^k by the application processes. Furthermore, isp^k knows at least one such MCS-process for each application process p. With this knowledge, isp^k will be able to propagate the write operations to other systems preserving the process order $\prec_p$ (see Definition 1). Then, in our algorithm, each IS-process isp^k, $k \in \{0, \cdots, N-1\}$, contains two concurrent tasks, $Propagate^k_{out}$ and $Propagate^k_{in}$. $Propagate^k_{out}$ deals with transferring write operations issued in S^k to every S^l, $l \neq k$, while $Propagate^k_{in}$ deals with applying within S^k the write operations transfered from the systems S^l, $l \neq k$. The two tasks that form the pRAM IS-protocol in SDP operate as follows (see Fig. 1):

- Task $Propagate^k_{out}$ is activated after a message $msg(p, s(p), x, v)$, for some process p, is received by isp^k. Then, $Propagate^k_{out}$ sends the pair $\langle x, v\rangle$ to every isp^l, $l \neq k$. From the above Property 1, the sending of the pairs generated from the write operations issued by p follows p's process order. We avoid to re-propagate write operations received from other systems, by checking that the write operation was not issued in S^k by isp^k.
- Task $Propagate^k_{in}$ is activated whenever a pair $\langle x, v\rangle$ is received from some process isp^l, $l \neq k$. As a result, it performs a write operation $w^k_{isp^k}(x)v$, thus propagating the value v to all the replicas of variable x within S^k.

```
1  Task Propagate^k_out :: upon
     reception of msg(p, s(p), x, v)
2  begin
3    if p ≠ isp^k then
4      send ⟨x, v⟩ to every isp^l, l ≠ k
5  end
```

```
1  Task Propagate^k_in :: upon
     reception of ⟨x, v⟩ from isp^l, l ≠ k
2  begin
3    w^k_{isp^k}(x)v
4  end
```

Fig. 1. The pRAM IS-protocol in isp^k in SDP.

The correctness of the IS-protocol of Fig. 1 is omitted due to space limitations. There, we show that the system S^T, obtained by connecting N systems $S^0, \cdots, S^{N-1}$ using this pRAM IS-protocol in SDP, is pRAM.

We now consider an IS-protocol in WDP such that this IS only interconnects MCSs that fulfill the following Property 2.

Property 2. In any computation α^k of system S^k ($k \in \{0, \cdots, N-1\}$), for each process p in S^k, if p issues two write operations $w^k_p(x)v \prec_p w^k_p(y)u$, then $mcs(isp^k)$ updates its local replica of x with the value v before updating its local replica of y with the value u.

Property 2 guarantees that the MCS-process of isp^k applies the write operations in the same order as they are issued in S^k. We can observe that this

Property 2 is a particular case of Property 1 where process $s(p)$ is now $mcs(isp^k)$, for all p. Hence, we can use the same IS-protocol of Figure 1.

Finally, to end this section, we consider an IS-protocol in SDI such that this IS only interconnects MCSs that fulfill the following Property 3.

Property 3. In any computation α^k of system S^k ($k \in \{0, \cdots, N-1\}$), for each process p in S^k, if p issues two write operations $w_p^k(x)v \prec_p w_p^k(y)u$, then $mcs(p)$ updates its local replica of x with the value v before updating its local replica of y with the value u.

Now Property 3 guarantees that every MCS-process of system S^k applies the write operations in the same order as they are issued in S^k. Again, this property is a particular case of Property 1 where process $s(p)$ is now $mcs(p)$. Hence, we can use the same IS-protocol of Figure 1 to interconnect pRAM systems in SDI.

5 Causal Consistency Model

In this section we study the interconnection of causal systems. Note that the pRAM model is strictly weaker than causal model [3,8]. Therefore, the results of impossibility of Section 4 are also applicable to causal systems.

In Section 4 we present an IS–protocol in SDP for interconnecting pRAM systems satisfying Property 1. We can show that there is no IS in SDP that interconnects every pair of causal systems satisfying Property 1. The proof is omitted due to space limitations. This result can be easily extended to WDP with Property 2 and SDI with Property 3.

We now propose an IS–protocol in SDP for causal systems. We also show in this subsection that the resulting system of this interconnection is causal. For our IS, we will only consider MCSs that fulfill the following Property 4 (which is in fact satisfied by all the causal protocols we have found).

Property 4. Consider any execution α^k of the causal system S^k (where $k \in \{0, \cdots, N-1\}$). For each two write operations $w_p^k(x)v \prec w_q^k(y)u$ on α^k, each MCS–process of system S^k updates its local replica of x with the value v before updating its local replica of y with the value u.

Property 4 guarantees that every MCS-process of system S^k applies the write operations preserving the execution order $\prec$ (see Definition 2). In Figure 2 we present the causal IS-protocol we propose. This protocol requires that the communication among IS-processes is totally ordered. There are well-known message-passing protocols (e.g., [5, pp. 177-179]) to provide total ordering of messages.

It can be observed that the IS-protocol is composed by two task, like the IS-protocol of Fig. 1. In fact, the $Propagate_{in}^k$ task is the same. However, the key difference is found in task $Propagate_{out}$. In this task a pair $\langle x, v \rangle$ is not sent to the other systems until all the MCS replicas of x have been updated. Note that, from Property 4, write operations are propagated to the rest of systems following

```
1  Task Propagate^k_out :: upon reception
     of msg(p, q, x, v), from every
     MCS-process q
2  begin
3    if p ≠ isp^k then
4      send ⟨x, v⟩ to every isp^l, l ≠ k
5  end
```

```
1  Task Propagate^k_in :: upon reception
     of ⟨x, v⟩ from isp^l, l ≠ k
2  begin
3    w^k_{isp^k}(x)v
4  end
```

Fig. 2. The causal IS-protocol in isp^k in SDP.

the causal order in system S^k. We need the communication among IS-processes to be totally ordered to enforce that two write operations from different systems and casually ordered, are applied in the rest of systems in the causal order.

Let us now show that the system S^T, obtained by connecting N systems $S^0, \cdots, S^{N-1}$ using the causal IS–protocol of Fig. 2 in SDP, is causal.

Let p be some process in system S^k, $k \in \{0, \cdots, N-1\}$, and let $mcs(p)$ be its MCS-process. Recall that α_p^k (resp. α_p^T) is the set obtained by removing from α^k (resp. α^T) all read operations except those from process p. We define β_p^k as a sequence with the same operations as α_p^k that preserves the order in which all operations of α_p^k are issued by process p, and the order in which every write operation is applied in $mcs(p)$. Formally,

Definition 6. *Let β_p^k a sequence of the operations in α_p^k. Let op and op' in α_p^k. Then $op \rightarrow op'$ in β_p^k, if any of the following happens:*
1. op and op' are operations from the same process p of S^k and $op \prec_p op'$.
2. $op = w_q^k(x)u$, $op' = w_s^k(y)v$, and in $mcs(p)$ the local copy of x is updated with u before updating y with v.
3. $op = w_q^k(x)u$, $op' = r_p^k(y)v$, and in $mcs(p)$ the local copy of x is updated with u before p issues op'.

Note that, like in α_p^k, every write operation of process isp^k in β_p^k is the propagation of a write operation issued by a process of S^l, $l \neq k$. We define β_p^T as the sequence obtained by replacing in β_p^k every write operation op from isp^k by the write operation $orig(op)$. The proof of the following results is omitted due to space limitations.

Lemma 1. *β_p^T is formed by all operations of α_p^T, preserves the execution order $\prec$ on α^T, and is legal.*

Theorem 3. *The system S^T is causal.*

6 Cache Consistency Model

In this section we study the interconnection of cache systems. We show that, unlike the previous models, the interconnection of cache systems is always possible, independently of how they are implemented. The interconnection only uses

read and write operations, without any other consideration about the interface between the MCS and the IS. Hence, we can use the same IS-protocol for SDP, SDI and WDP classes.

The IS-protocol we propose only works for the interconnection of two systems. However, it can be repeatedly used to interconnect as many systems as desired. Each isp^k (in this case $k \in \{0, 1\}$) has one task for each variable of the shared memory, presented in Figure 3. Note that each IS-process maintains a copy of

```
1   Task Propagate^k(x) :: upon reception of ⟨x, v⟩ from isp^{1−k}
2   begin
3       if v ≠ "NoData" then
4           w^k_{isp^k}(x)v
5           last(x) = v
6       r^k_{isp^k}(x)u
7       if u = last(x) then
8           u = "NoData"
9       send ⟨x, u⟩ to isp^{1−k}
10  end
```

Fig. 3. The cache IS-protocol in isp^k for variable x.

the latest value propagated from the other system in $last(x)$ for each variable x. That copy must be initialized with a special value (e.g., "$NoData$"). Note also that initially one of the IS-processes (for instance isp^0) must send to the other a message with $\langle x, NoData \rangle$ for each variable x to start the interconnection. The proof that this cache IS-protocol interconnects two cache systems is omitted due to space limitations.

7 Conclusions

In this paper we have formalized and studied the interconnection of distributed shared memory systems. We have shown that non-fast, pRAM, and causal systems cannot be interconnected in general, while cache systems can. Then, we have given sufficient conditions to interconnect pRAM and causal systems. Then, with the results presented in this paper we can determine, for example, whether several systems, that implement certain memory models, can be interconnected by merely looking at the properties that these systems satisfy, and independently of what specific protocols they use.

Limitations of space have made impossible to include all the proofs. A complete version of this paper can be found in [14].

References

1. S.V. Adve. *Designing Memory Consistency Models for Shared-Memory Multiprocessors.* PhD thesis, University of Wisconsin-Madison, 1993.

2. M. Ahamad, R. Bazzi, R. John, P. Kohli, and G. Neiger. The power of processor consistency. In *Proceedings of the 5th ACM Symposium on Parallel Algorithms and Architectures*, pages 251–260, 1993.
3. M. Ahamad, G. Neiger, J.E. Burns, P. Kohli, and P.W. Hutto. Causal memory: Definitions, implementation and programming. *Distributed Computing*, 9(1):37–49, August 1995.
4. H. Attiya and R. Friedman. Limitations of fast consistency conditions for distributed shared memories. *Information Processing Letters*, 57:243–248, 1996.
5. H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* McGraw-Hill, 1998.
6. H. Attiya and J.L. Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems*, 12(2):91–122, 1994.
7. B.N. Bershad, M.J. Zekauskas, and W.A. Sawdon. The Midway distributed shared memory system. In *COMPCON*, 1993.
8. V. Cholvi. *Formalizing Memory Models.* PhD thesis, Department of Computer Science, Polytechnic University of Valencia, December 1994.
9. V. Cholvi. Specification of the behavior of memory operations in distributed systems. *Parallel Processing Letters*, 8(4):589–598, December 1998.
10. A. Fernández, E. Jiménez, and V. Cholvi. On the interconnection of causal memory systems. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing.* ACM, July 2000.
11. K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 15–26. ACM, May 1990.
12. J.R. Goodman. Cache consistency and sequential consistency. Technical Report 61, IEEE Scalable Coherence Interface Working Group, March 1989.
13. L. Iftode, J. Singh, and K. Li. Scope consistency: A bridge between release consistency and entry consistency. In *Proc. of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1996.
14. Ernesto Jiménez, Antonio Fernández, and Vicente Cholvi. Decoupled Interconnection of Distributed Memory Models. Technical Report TR-GSYC-2003-2, Universidad Rey Juan Carlos, October 2003, http://gsyc.escet.urjc.es/publicaciones/tr.
15. P. Keleher. *Distributed Shared Memory Using Lazy Consistency.* PhD thesis, Rice University, 1994.
16. L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 28(9):690–691, September 1979.
17. L. Lamport. On interprocess communication: Parts I and II. *Distributed Computing*, 1(2):77–101, 1986.
18. R.J. Lipton and J.S. Sandberg. PRAM: A scalable shared memory. Technical Report CS-TR-180-88, Princeton University, Department of Computer Science, September 1988.