INFORMATICA, 1995, Vol. 6, No. 1, 35–60

A DEADLOCK RESOLUTION ALGORITHM*

José Ramón GONZÁLEZ DE MENDÍVIL
Carlos F. ALASTRUEY

Department of Electricity and Electronics
University of the Basque Country
Faculty of Sciences-Leioa, P.O.Box 644, 48080 Bilbao, Spain

José BERNABEU

Department of Information Systems and Computation
Polytechnic University of Valencia
Camino de Vera sn, P.O.Box 22012, 46020 Valencia, Spain

Akim DEMAILE

Telecom Paris
46 rue Barrault, 75013 Paris, France

Abstract. In this paper we consider the problem of the distributed deadlock resolution. Starting from a high level specification of the problem and the resolution algorithm for a system with single request model, we provide successive levels of decreasing abstraction of the initial specification in order to achieve a solution in a complete distributed system. The successive refinements and the final distributed deadlock resolution algorithm are formaly described and proved by using the Input-Output Automata Model. The proposed solution is a modification of the algorithms in Mitchell and Merritt (1984) and González de Mendívil *et al.* (1993) and preserves a similar message traffic to resolve a deadlock.

Key words: distributed systems, distributed deadlocks, deadlock resolution, input-output automata model, single request model.

1. Introduction. The problem of deadlock is crucial in the design of concurrent systems. This problem is well understood in shared-memory systems but it represents a very difficult problem in distributed systems (Singhal, 1989). In an intuitive way, the deadlock situation is achieved in a system when a group of processes are indefinitely blocked because they request access to resources

* This work has been supported by the research grant no. UPV 224.310-EA061/93 of the University of the Basque Country and by CICYT grant TIC93-0304.

which are allocated to processes in the same group. The natural form for modelling the wait for relations between processes is by using a Wait-For-Graph (WFG). The nodes in the graph represent the actor (processes, resources) in the system and the arcs represent the situations: "actor p waits for actor q". The WFG is used for representing different models of recource allocation strategies and it is a comprehensive way to compare different solutions for handling deadlocks (Knapp, 1987).

For the particular case of the *single request model*, the necessary and sufficient condition for the existence of a deadlock is that of a directed cycle in the WFG (Knapp, 1987). The traditional form of deadlock algorithms for distributed systems comprises two steps:

(a) detection step, to find out cycles in the WFG, and

(b) resolution step, to resolve the deadlock by selecting a unique victim in the cycle.

Many of the algorithms in the distributed deadlock detection-resolution (DDD/R) topic for the single request model are probe-based algorithms (Knapp, 1987). A first distinction can be done between those that use priority (solution that uses fewer messages to detect the deadlocks), and those which do not (they need no extra round to resolve the deadlocks). A second distinction can be done in the former class, depending whether the algorithm stores the probes or not. On one hand, storing algorithms, such as in Chady and Misra (1987); Mitchell and Merritt (1984), have a smaller cost, in message trafic, during the detection phase, but need cleaning rounds for the resolution; on the other hand, non-storing algorithms (González de Mendívil, *et al.* 1992; González de Mendívil *et al.*, 1993) require a slightly higher trafic for the detection, but need no additional phase. Our algorithm is priority-based-without-storing.

In general, the proof of the correctness of DDD/R algorithms is difficult because of the highly complex operations of such algorithms. This is why most proposals for distributed deadlock algorithms have either (Kshemkalyani and Singhal, 1991) (a) ignored the correctness proof (Sinha and Natarajan, 1985), (b) used simulation techniques to "show" correctness (Choundhary *et al.* 1989), or (c) given informal or intuitive arguments of correctness (Obermarck, 1982).

Only recently attention has been paid to rigorous correctness proofs of the algorithms (Kshemkalyani and Singhal, 1991; González de Mendí vil *et al.*, 1992; Elmagarmid, 1988). However, due to the lack of an underlying theory for DDD/R and the highly complex operations of the algorithms, most of these correctness proofs have used *ad hoc* methods (González de Mendívil *et al.*, 1992; González de Mendívil *et al.*, 1993; Elmagarmid, 1988). Other correctness proofs are based on formal methods, such as the use of *invariant techniques* (Kshemkalyani and Singhal, 1991), but at a low level of abstraction.

We propose the use of the *Input-Output Automata Model* (IOA model) as the base of our formalism. This model was designed as a tool for modelling concurrent and distributed systems. We refer the reader to (Lynch and Tuttle, 1989; Tuttle, 1989) for a complete development of the model, motivation and examples.

In the particular topic of *deadlock resolution*, a great number of papers examine separately deadlock detection and deadlock resolution (Singhal, 1989; Kshemkalyani and Singhal, 1991). These algorithms take into account correctness criteria just based on deadlock detection:

(a) detection of every existing deadlock;

(b) no false deadlocks detected.

As a consequence, algorithms which are correct under those criteria, may be incorrect after the resolution phase. The criteria of correctness based strictly on the resolution of deadlock has been recently proposed in González de Mendívil *et al.* (1994) by using the IOA formalism.

In González de Mendívil *et al.* (1993), the authors have introduced a first refinement of the automaton specification in order to obtain an abstract solution for a distributed deadlock resolution algorithm. The algorithm is priory-based-without-storing, and is based on a modification of the algorithm proposed in Mitchell and Merritt (1984) and (González de Mendívil *et al.*, 1992). Such a solution verifies the criteria of correctness:

(a) liveness condition: *the resolution algorithm resolves every deadlock*;

(b) safety condition: *the resolution algorithm does not resolve false deadlocks.*

Its performance (González de Mendívil *et al.* (1994). is similar to the existing algorithms: a worst case $O(N^2)$ in messages for resolving a deadlock is achieved, being N the number of nodes involved in the deadlock.

In this paper, we introduce a new refinement of the automaton which represents the abstract distributed solution in González de Mendívil *et al.* (1994). The proposed automaton is designed in order to obtain a completely *distributed* algorithm. This means that in every site there is a running instance of the algorithm, and the whole of them, cooperating via the communication network, forms the complete deadlock resolution mechanism. We also prove that the distributed solution verifies the criteria of correctness given above.

The rest of this paper is organized as follows. In Section 2 is briefly presented the specification of the problem of deadlock resolution and the abstract solution proposed in González de Mendívil *et al.* (1994). Section 3 is devoted to introduce the deadlock resolution algorithm for a distributed system. In Section 4, we provide the formal proof of correctness. Finally, Section 5 is devoted to concluding remarks and future perspectives.

2. Abstract solution of a deadlock resolution algorithm. Deadlock resolution algorithms are primary intended to perform some activity in order to solve a deadlock situation. In absence of such activity, deadlocks can indefinitely stand in the system as a stable property. A deadlock resolution algorithm can be considered correct if the following criteria of correctness hold.

Liveness Condition. If there exists a deadlock in the system then it will be eventually resolved by the resolution algorithm.

Safety Condition. The resolution algorithm does not resolve false deadlocks.

Where in condition (2), resolution of a false deadlock is considered to be the resolution of a non-existing deadlock.

In this paper, as the algorithm is presented using the IOA formalism, the mathematical objects used will be in the first parts quite general, and by the end, will become more restrictive. As an illustration, all the content of this section is devoted to abstract graphs. Their nodes can be interpreted as the actors defined preiously (this is why we keep on writing WFG), however they represent a situation much more general, only based on some properties of those graphs. Note that the generality will be used in the following sections, where we introduce a new kind of nodes. We will use standard definitions on graphs; $pred_A(i)$ is the set of predecessors of a node i in a given set of arcs A; $succ_A(i)$ is the set of its successors; $incom_A(i)$ is the set of incoming arcs; and $d_A^+(i)$ is its outdegree (number of successors).

In order to characterize the deadlock situation, it is convenient to use the WFG (Knapp, 1987). WFG is a pair (N, A) where N is the (countable) set of nodes, and $A \subseteq N \times N$. Their elements will be denoted, respectively, i, j, k, r, l

or n; and $\langle i,j\rangle, \langle l,r\rangle \ldots$. We assume nodes in the system are distinguishable and ordered: $(N,<)$ is a totally ordered set ($i<j$ iff j has higher priority than i).

An arc$\langle i,j\rangle \in A$ means that i *waits* for j. Since we restrain our work to the single request model, a deadlock corresponds with the existence of a directed cycle $\mathbf{C} = \{\langle i,(i+1) \bmod k\rangle \in A \mid 0 \leqslant i < k-1\}$ in the WFG. For sake of clearance, we shall now simply write $\mathbf{C} = \{\langle i, i+1\rangle \in A\}$. Let $\mathcal{N}(\mathbf{C})$ be the set of nodes in the arcs of $\mathbf{C}$. If there exists a cycle $\mathbf{C}$, a node $n \in \mathcal{N}(\mathbf{C})$ must be selected as victim in order to resolve the deadlock. The deadlock is resolved by aborting the victim.

The system under consideration comprises the following components (Fig. 1):

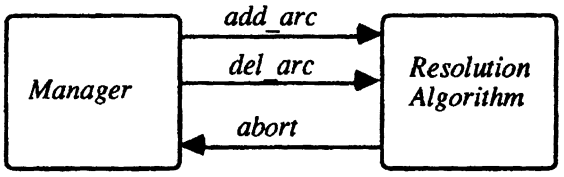

Fig. 1. System components and actions.

(a) a *Manager* of the active elements of the system, which provides the actions *add_arc*(.) and *del_arc*(.) respectively to create a new wait-for relation or to delete it;

(b) a *Resolution Algorithm* which provides to the Manager the action *abort*(.) when it resolves a cycle. We assume that the effective resolution of a deadlock for the Manager is properly made.

In the following subsection, we give an automaton, denoted R_0, which represents the resolution algorithm, and a set of well formed histories which represents the fair behaviours of the automaton of the manager, M_0. We do not develop M_0 in order to simplify the discussion.

2.1. Specification of a deadlock resolution algorithm. The single request model is characterized by the conditions:

(a) a node cannot directly wait for itself;

(b) a node cannot be waiting for more than one node;

(c) spontaneous abortions of nodes are not allowed: all the abortions are caused by the deadlock resolution algorithm;

(d) an aborted node does not take part in the system operations, the previous dependencies of the victim with another nodes will be cancelled by the manager.

A manager satisfying the previous assumptions provides to the resolution algorithm a set of well formed histories, denoted H_0, as it formalized in the following Definition 2.1.

In the following definition, we introduce the abuses of notation $\pi_{k0} \dots \pi_{k1}$ as the set $\{\pi_k \mid k_0 \leqslant k \leqslant k_1\}$; $\pi_{k0} \dots$ as $\pi_{k0} \dots \pi_\infty$. Finally, the set $B = \{\langle i,j\rangle \mid i \neq j\} \subseteq N \times N$, is a fixed set indicating the possible arcs to be performed by the Manager.

DEFINITION 2.1. Let Σ be the signature of actions:
$\Sigma = (in(\Sigma), \varnothing, out(\Sigma))$:
$in(\Sigma) = \{add\_arc(\langle i,j\rangle, del\_arc(\langle i,j\rangle) \mid \forall \langle i,j\rangle \in B\}$,
$out(\Sigma) = \{abort(i) \mid \forall i \in N\}$.

Let β be a sequence of actions in Σ. The behaviour $\beta = \pi_1\pi_2 \dots \pi_k \dots$ is a well formed history, $\beta \in H_0$, iff the following conditions hold:

(i) $\pi_k = add\_arc(\langle i,r\rangle) \wedge \pi_{k'} =$ // No node has more
$add\_arc(\langle i,j\rangle) \wedge k < k' \Rightarrow$ // than one outgoing arc
$\pi_k \dots \pi_{k'} \cap \{del\_arc(\langle i,r\rangle), abort(r)\} \neq \varnothing$;

(iia) $\pi_{k'} = del\_arc(\langle i,j\rangle) \Rightarrow$ //An arc can be deleted only when
$\exists k < k'$: $\pi_k = add\_arc(\langle i,j\rangle)$ // it actually exists
$\wedge \pi_k \dots \pi_{k'} \cap \{del\_arc(\langle i,j\rangle), abort(j)\} = \varnothing$;

(iib) $\pi_{k'} = del\_arc(\langle i,j\rangle) \Rightarrow$ // and when it has no
$(\exists k < k'$: $\pi_k = abort(i))$ // successor or its origin
$\vee(\exists k < k'$: $\pi_k = add\_arc(\langle j,r\rangle) \Rightarrow$ // is a victim
$\pi_l \dots \pi_{k'} \cap \{del\_arc(\langle j,r\rangle), abort(r)\} \neq \varnothing)$;

(iii) $\pi_k = abort(i) \Rightarrow$ // Once a node has been aborted,
$\pi_k \dots \cap$ // it no longer participates
$\{add\_arc(\langle i,r\rangle), add\_arc(\langle r,i\rangle)\} = \varnothing$.

In the following, we introduce the automaton of the resolution algorithm, R_0, at highest level of abstraction.

– States of R_0. A state of R_0 is defined by a snapshot of the set of arcs A of the WFG and the set of *victims* in the system. A particular state $s \in states(R_0)$ is defined as a record $(s.A, s.victims)$.

– Start States of R_0. The start states of $R_0, start(R_0) \subseteq states(R_0)$, are the possible initial configurations for the system: $s_0 \in start(R_0)$ if and only if $s_0.A = \emptyset$ and $s_0.victims = \emptyset$.

– Actions signature of R_0. Its external signature is Σ (of Definition 2.1), and it has no internal action.

– Steps of R_0. The set of steps for the automaton R_0, $steps(R_0)$, is defined by the transition graph $steps(R_0) \subseteq states(R_0) \times acts(R_0) \times states(R_0)$. The steps of R_0, $(s, \pi, s') \in steps(R_0)$, are specified in Fig. 2 with the preconditions/effects formalism. An action π is enabled from any state s satisfying its preconditions, and π takes s to the state s' if s' can be obtained by modifying s as indicated by the effects π (the components of the state that are not modified are simply not mentionned). Since by definition (Lynch and Tuttle, 1989), an IOA has to be input enabled, no input action has precondition.

INPUT ACTIONS	OUTPUT ACTIONS
add_arc$(\langle i,j\rangle)$	**abort** (i)
eff $s'.A \leftarrow s.A + \{\langle i,j\rangle\}$	pre $i \notin victims$, $\exists \mathbf{C} \subseteq A$, $i \in \mathcal{N}(\mathbf{C})$
del_arc$(\langle i,j\rangle)$	eff $s'.A \leftarrow s.A - incom_{s.A}(i)$
eff $s'.A \leftarrow s.A - \{\langle i,j\rangle\}$	$s'.victims \leftarrow s.victims + \{i\}$

Fig. 2. The automaton R_0.

The input actions of the automaton R_0 are *add_arc*(.), and *del_arc*(.) and their effects are obvious. Its output actions are *abort*(.). An *abort*(.) has as precondition the existence of a non-(being)-resolved cycle $\mathbf{C}$ in the system. The effect is the choise of a node $i \in \mathcal{N}(\mathbf{C})$ as victim and the break of the cycle.

– Partition of R_0. Finally, a partition of the locally controlled actions (the actions under control of the automaton, $local(R_0) = int(R_0) \cup out(R_0)$), must be defined in order to complete the description of R_0. Classes of the partition can be interpreted as the different tasks locally controlled by the automaton. The IOA model suppose a weak fairness notion among the tasks, that is, fair turns are provided for each class in order to perform an action if it is enabled. Guided by the previous interpretation, we define the partition as $part(R_0) = \{C_0^i \mid i \in N\}$, being $C_0^i = \{abort(i)\}$ to ensure that every enabled $abort(.)$ will be eventually executed.

The automaton R_0 preserves the well formedness of H_0 and also captures the desired resource allocation model:

$\forall \alpha \in execs(R_0)$: beh$(\alpha) \in H_0$, $\forall i \in N$, $\forall s \in states(R_0)$
 in α, $0 \leqslant d^+_{s.A}(i) \leqslant 1$;
$\forall \alpha \in execs(R_0)$: beh$(\alpha) \in H_0$: if there are two different
 cycles $\mathbf{C}_1, \mathbf{C}_2$ in a state s of α then $\mathbf{C}_1$ and $\mathbf{C}_2$ are disjoints –
 $\mathcal{N}(\mathbf{C}_1) \cap \mathcal{N}(\mathbf{C}_2) = \varnothing$.

In order to define in our system the concept of deadlocked execution (an execution where a cycle is never resolved and stands forever), we first define the persistent cycles. $\mathbf{C}$ is a *persistant cycle* of the execution α iff $\exists k$ such that $\forall k' \geqslant k$: $\mathbf{C} \in s_{k'}.A$.

DEFINITION 2.2. Let $\alpha \in execs(R_0)$. α is a *deadlocked execution* iff there exist a persistent cycle in α.

In González de Mendívil *et al.* (1994) is proved the following Lemma, specifying that every fair execution of R_0 such that its behaviour is well formed, is a non-deadlocked execution.

Lemma 2.3. *Let $\alpha \in fairexecs(R_0)$, if beh$(\alpha) \in H_0$ then α is a non-deadlocked execution.*

The liveness condition of the criteria of correctness provided at the begining of this Section is explicitly included in the Lemma 2.3 and the safety condition is implicitly included in the preconditions of the action $abort(.)$. Furthermore, the following simple property holds: if $\alpha \in fairexecs(R_0)$: beh$(\alpha) \in H_0$ then any deadlock is resolved exactly once. This property, which has been traditionally introduced in the literature as a condition of correctness (Singhal, 1989), is a direct consequence of our specification.

As a conclusion, $fairbehs(R_0) \cap H_0$ represents the specification of a deadlock resolution algorithm for the single request model.

2.2. A first approach to the resolution algorithm. In this first approximation the system is still seen as a whole and not as a distributed system.

However, since the proposed solution is designed to be distributed, that solution cannot be based on the knowledge of the whole WFG. We need a solution which could be distributed upon the sites of the distributed system in such a way that each part contributes to the global algorithm. The asynchrony due to the communications is capture in this abstract solution because the information which is handled by the algorithm, is based on information sending and receiving through the paths defined by the arcs. To maintain consistency of the vision of the WFG, each arc will now have a *version*, thanks to which the algorithm will be able to distinguish obsolete and valid information. The information shared by the different sites is sent under the form of a *mark*.

A mark m is a tuple $m = (j, \langle i,j\rangle, version(\langle i,j\rangle))$, where j is the node which originaly generated the mark (called the *initiator* of the mark), $\langle i,j\rangle$ is the arc where m had been first created, and $version(\langle i,j\rangle)$ is the version of $\langle i,j\rangle$ when m had been created. The set of marks m is denoted M, and M^* is the set of the finite strigs μ built on M (including ε the empty string). The marks which are travelling through an arc at a particular instant are called the *marks* of this arc; these marks form a string which represents the sending order. The marks generated by the algorithm are transmitted backwardly through the arcs of the WFG, that is, if m is a mark of $\langle i,j\rangle \in A$, then goes from j to i.

The automaton which represents the proposed solution at this level is denoted R_1 and is specified in the following paragraphs.

– States of R_1. A state of the automaton R_1 is an extension of a state of R_0. Its new components are $marks(\langle i,j\rangle)$, the string of marks of $\langle i,j\rangle$; and for each node j the set $create(j) = \{i \in pred_A(j) \mid i$ must create the mark $(j, \langle i,j\rangle, version(\langle i,j\rangle))\}$.

A particular state s of automaton R_1 is then defined by: $s.A$; $s.victims$; $\forall \langle i,j\rangle \in B$, $s.version(\langle i,j\rangle)$, $s.marks(\langle i,j\rangle)$; and $\forall i \in N$, $s.create(i)$.

– Start States of R_1. $s_0 \in start(R_1)$ iff $s_0.A = \varnothing$; $s_0.victims = \varnothing$; $\forall \langle i,j\rangle \in B$: $s_0.version(\langle i,j\rangle) = 0$, $s_0.marks(\langle i,j\rangle) = \varepsilon$, and $\forall i \in N$, $s_0.create(i) = \varnothing$.

– Actions signature of R_1. Its external signature is the same as R_0, and its internal actions are $int(R_1) = \{create\_mark(\langle i,j\rangle)$, $trans\_mark(m, \langle i,j\rangle)$, $elim\_mark(m, \langle i,j\rangle)$, $lose\_mark(m, \langle i,j\rangle)\}$, where $\langle i,j\rangle \in B$, $m \in M$.

– States of R_1. Informally the algorithm runs as follows: when a node detects the possibility of a cycle, it generates a mark ($create\_mark(.)$). This mark is transmitted in the backward directions of the graph. Transmission is performed if the priority of the initiator of the mark is higer than the priority of the nodes through which the mark is travelling ($trans\_mark(.)$); otherwise the mark is eliminated ($elim\_mark(.)$) and the arc which eliminated it may create a new higer priority mark. If no path traversed by a mark is a cycle then eventually the mark will be lost ($lose\_mark(.)$). If a mark reaches the arc which originated it and if they have same version, then the algorithm concludes to the existance of a cycle and breaks it by aborting the chosen node ($abort(.)$). Fig. 3 shows the actions and steps of the automaton R_1. Well formed histories of automaton R_1 are the same as those of automaton R_0, that is, H_0.

```
                    INPUT ACTIONS
add_arc($(\langle i,j\rangle)$)
    eff $s'.A \leftarrow s.A + \{\langle i,j\rangle\}$
    $s'.create(j) \leftarrow s.create(j) + \{i\}$
    $s'.version(\langle i,j\rangle) \leftarrow s.version(\langle i,j\rangle) + 1$
del_arc($(\langle i,j\rangle)$)
    eff $s'.A \leftarrow s.A - \{\langle i,j\rangle\}$
    $s'.create(j) \leftarrow s.create(j) - \{i\}$
    $s'.marks(\langle i,j\rangle) \leftarrow \varepsilon$

                    OUTPUT ACTIONS
abort($(i)$)
    pre $i \notin victims$
    $\exists\langle i,k\rangle \in A : marks(\langle i,k\rangle) = m \bullet \mu$,
        $m = (i, \langle j,i\rangle, v)$,
        $\langle j,i\rangle \in A \wedge v = version(\langle j,i\rangle)$
    eff $s'.A \leftarrow s.A - incom_{s.A}(i)$
    $s'.victims \leftarrow s.victims + \{i\}$
    $\forall j \in pred_{s.A}(i)$ :
        $s'.marks(\langle j,i\rangle) \leftarrow \varepsilon$
    $s'.create(i) \leftarrow \varnothing$
```

Fig. 3. The automaton R_1 (to be continued).

```
INTERNAL ACTIONS
create_mark($\langle i,j\rangle$)
  pre $j \notin victims,\ succ_A(j) \neq \emptyset,\ \langle i,j\rangle \in A,$
    $i \in create(j)$
  eff $m = (j, \langle i,j\rangle, version(\langle i,j\rangle))$
    $s'.marks(\langle i,j\rangle) \leftarrow s.marks(\langle i,j\rangle) \bullet m$
    $s'.create(j) \leftarrow s.create(j) - \{i\}$
trans_mark($m, \langle i,j\rangle$)
  pre $i \notin victims,\ pred_A(i) \neq \emptyset,\ \langle i,j\rangle \in A,$
    $marks(\langle i,i\rangle) = m \bullet \mu$
      $m = (r\langle l,r\rangle, v),\ r > i$
  eff $s'.marks(\langle i,j\rangle) \leftarrow \mu$
    $\forall k \in pred_{s.A}(i)$ :
      $s'.marks(\langle k,i\rangle) \leftarrow s.marks(\langle k,i\rangle) \bullet m$
elim_mark($m, \langle i,j\rangle$)
  pre $i \notin victims,\ pred_A(i) \neq \emptyset,\ \langle i,j\rangle \in A,$
    $marks(\langle i,j\rangle) = m \bullet \mu$
      $m = (r\langle l,r\rangle, v),\ r < i$
  eff $s'.marks(\langle i,j\rangle) \leftarrow \mu$
    $s'.create(i) \leftarrow s.create(i) + pred_{s.A}(i)$
lose_mark($m, \langle i,j\rangle$)
  pre $marks(\langle i,j\rangle) = m \bullet \mu$
      $m = (r\langle l,r\rangle, v)$
    $pred_A(i) = \emptyset \vee i \in victims \vee$
      $(r = i \wedge \langle l,r\rangle \in A \wedge version(\langle l,r\rangle) \neq v)$
  eff $s'.marks(\langle i,j\rangle) \leftarrow \mu$
```

Fig. 3. The automaton R_1.

– Partition of $R_1.part(R_1) = \{C_l^i, \forall i \in N\} \cup \{D_l^{ij}, \forall\langle i,j\rangle \in B\} \cup \{E_l^{ij}, \forall\langle i,j\rangle \in B\}$, where $C_l^i = \{abort(i)\}$; $D_l^{ij} = \{create\_mark(\langle i,j\rangle)\}$ and $E_l^{ij} = \{trans\_mark(m,\langle i,j\rangle),\ lose\_mark(m,\langle i,j\rangle),\ elim\_mark(m,\langle i,j\rangle) \mid \forall m \in \Omega\}$.

In González de Mendívil *et al.* (1994), we formally proved that $(fairbehs(R_1) \cap H_0) \subseteq (fairbehs(R_0) \cap H_0)$, and therefore, R_1 under the well formed histories H_0 verifies the criteria of correctness of the specification ($fairbehs$ $(R_0) \cap H_0$).

3. Towards a distributed deadlock resolution algorithm. In this Section, we proposed a distribution of the algorithm presented in the Section 2.2. In order to achieve this goal, it is necessary to introduce a first approach to the distributed Manager. The approach is based on the definition of a set of intermediate nodes which indicates the relation between a blocked node and the node which blocks it. A new set of well formed histories is defined as a subset of the histories for the single request model, to model the formation of the arcs by the Manager. A simple modification of the automaton R_1 is exposed, based on the properties which are obtained from the new histories and the set of nodes. Finally, it is developed an automaton which includes the concepts of communication channel and site of the distributed system.

3.1. The intermediate nodes and the distributed manager. We now introduce a new topography of the system, in order to capture the distributed aspect. P will denote the set of the previously defined actors, and E will be the set of the *intermediate nodes*, introduced in order to capture the communication sequences between two actors. Then, we define N as the set of both sets of node, and B the set of the possible arcs, what makes us able to reuse any previous result on N and B (the reader can easily check that care had been taken in the previous sections to allow such a mathematical trick). Henceforth, we will distinguish the two kinds of node (by notation p_i or e_{ij}, and P or E) only when necessary; otherwise, set N and its nodes $n, i \dots$ are freely used. Given as an example to help the reader, in the following Definition 3.1, in(Σ) and out(Σ) are rigourously the ones given in Definition 2.1, with new N and B.

More formaly, we have: let $P = \{p_i,\ i \in I\}$, $E = \{(p_i, p_j) \mid i, j \in I,\ i \neq j\}$ (a pair (p_i, p_j) will be denoted e_{ij}); $N = P \cup E$; $B = \{\langle p_i, e_{ij}\rangle, \langle e_{ij}, p_j\rangle \mid p_i, p_j \in P,\ e_{ij} \in E\}$. $(N, >)$ is supposed to be totally ordered, and with the property: $\forall p \in P,\ e \in E : p > e$.

Using the set of intermediate nodes the situation *node p_i waits for node p_j* is represented by two arcs $\langle p_i, e_{ij}\rangle$ and $\langle e_{ij}, p_i\rangle$, so a new set of well formed histories H_1 is needed in order to take into account this new interpretation (Fig. 4 helps in understanding the follwing statements). Since our Manager is of the single request model, it has a certain behaviour: it represents the

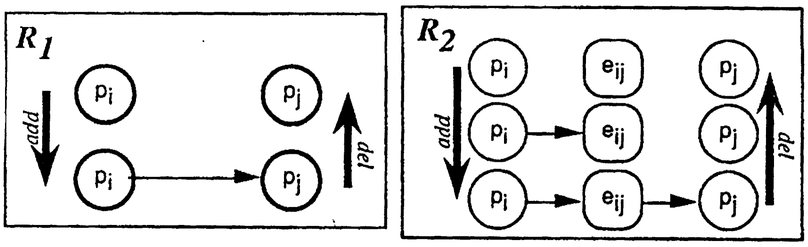

Fig. 4. Behaviour of adjacent arcs in R_2.

wait-for relations by two edges that will be added and deleted in a determined order. This behaviour is similar to that considered by a great part of published DDD/R algorithms that use the WFG (Knapp, 1987); its interpretation is quite simple: both arcs of R_2 must behave rigourously like the single one of R_1 they represent. We restate this in order to find the mathematical expression of these properties:

(a) an arc $\langle e_{ij}, p_j\rangle$ cannot be added if previously the arc $\langle p_i, e_{ij}\rangle$ has not been added and remains;

(b) an arc $\langle p_i, e_{ij}\rangle$ cannot be deleted if previously the arc $\langle e_{ij}, p_j\rangle$ has not been deleted.

Definition 3.1. Let Σ be the signature of actions: $\Sigma = (in(\Sigma), \emptyset, out(\Sigma))$:

$$\begin{aligned} in(\Sigma) = \{&add\_arc(\langle p_i, e_{ij}\rangle),\ add\_arc(\langle e_{ij}, p_j\rangle),\\ &del\_arc(\langle p_i, e_{ij}\rangle),\ del\_arc(\langle e_{ij}, p_j\rangle) \mid\\ &\forall\langle p_i, e_{ij}\rangle,\ \langle e_{ij}, pj\rangle \in B\}\\ out(\Sigma) = \{&abort(p_i),\ abort(e_{ij}) \mid \forall p_i \in P,\ \forall e_{ij} \in E\}. \end{aligned}$$

Let β be a sequence of actions in Σ. The behaviour $\beta = \pi_1\pi_2\ldots\pi_k\ldots$ is a well formed history ($\beta \in H_1$), iff the following conditions hold:

(i) $\beta \in H_0$ //β is a single request behaviour

(ii) $\pi_{k'} = add\_arc(\langle e_{ij}, p_j\rangle) \Rightarrow$ // Two adjacent arcs behave like
$\exists k < k',\ \pi_k = add\_arc(\langle p_i, e_{ij}\rangle)$ // a single one when created
$\wedge \pi_k \ldots \pi_{k'} \cap \{del\_arc\langle e_{ij}, p_j\rangle),\ del\_arc(\langle p_i, e_{ij}\rangle)\} = \emptyset$;

(iii) $\pi_{k'} = del\_arc(\langle p_i, e_{ij}\rangle) \Rightarrow$ // and when deleted
$\exists k < k',\ \pi_k = add\_arc(\langle p_i, e_{ij}\rangle)$
$\wedge \pi_k \ldots \pi_{k'} \cap \{del\_arc\langle e_{ij}, p_j\rangle),\ abort(p_j)\} = \emptyset.$

It is trivial to prove that $H_1 \subset H_0$; therefore, $(fairbehs(R_1) \cap H_1) \subset (fairbehs(R_1) \cap H_0)$ and R_1 under the well formed histories H_1 maintains the criteria of correctness. We will now prove some properties of R_1 under the well formed histories H_1 that will enable us to simplificate this automaton, which will then be denoted R_{1_e}. First we claim that $abort(e_{ij})$ is never enabled, and therefore, can be omitted in R_{1_e} (in order to stay consistent with the IOA model, an *omitted* action is in fact an action renamed to *null*). The following property will help us.

Property 3.2. *Let* $\alpha \in execs(R_1)$*. If mis a mark of* $\langle e, p\rangle \in s_k.A$ *then its initiator has a higher priority than e.*

Proof. Since $m = (r\langle.\rangle, .)$ stands in $s_k.marks(\langle e_{ij}, p_j\rangle)$, one of the following actions had been performed in $k' \leqslant k$: $\pi_{k'} = create\_mark(\langle e_{ij}, p_j\rangle)$ or $\pi_{k'} = trans\_mark(m, \langle p_j, e_{ij}\rangle)$. In the last case m has to be $(n, \langle.\rangle, .)$ with $r > p_j > e_{ij}$. In both cases $r > e_{ij}$.

Due to the Property 3.2 the preconditions of $elim\_mark(m, \langle e_{ij}, p_j\rangle)$ are always false, so there will be disabled in every state of automaton R_1. Thus they can be transformed in *null* actions without effects by renaming. Such actions will not be taken into account. The checking of $r > e_{ij}$ in precondition of $trans\_mark(m, \langle e_{ij}, p_j\rangle)$ can also be eliminated as it is always true.

Property 3.3. $\forall \alpha \in execs(R_1)$, $\forall e \in E$: $abort(e)$ *is disabled in every state of* α.

Proof. Property 3.2 assures that if there exists a mark m such that m stands in $s.marks(\langle e_{ij}, p_j\rangle)$, being $m = (r, \langle.\rangle, .)$, it is verified that $r > e_{ij}$. Therefore $r \neq e_{ij}$ and the preconditions of $abort(e_{ij})$ actions are never verified in any state of any execution of the automatn R_1.

By Property 3.3 $abort(e)$ actions could be transformed in *null* actions, however, we do not rename such actions in order to maintain $ext(R_1) = ext(R_{1_e})$. Note that internal actions can be renamed because it does not affects the external signature.

Furthermore, $\forall e_{ij} \in E$ and in every state of the executions of the automaton R_1, $e_{ij} \notin victims$. Consequently, the preconditions of $create\_mark(\langle p_i, e_{ij}\rangle)$ and $trans\_mark(m, \langle e_{ij}, p_j\rangle)$ actions can be simplified.

Property 3.4. *Let $\alpha \in execs(R_1)$: $beh(\alpha) \in H_1$, let s_k be a state of α such that $\langle e_{ij}, p_j\rangle \in s_k.A$, then $\langle p_i, e_{ij}\rangle \in s_k.A$ and consequently $pred_{s_k.A}(e_{ij}) = \{p_i\}$.*

Proof. The proof is a direct consequence of Definition 3.1 of the well formed histories.

Due to Property 3.4, it is sufficient to check the existence of $\langle e_{ij}, p_j\rangle$ arc in $create\_mark(\langle p_i, e_{ij}\rangle)$ and $trans\_mark(m, \langle e_{ij}, p_j\rangle)$ actions. The checking of the existence of $\langle p_i, e_{ij}\rangle$ arc can be omitted. Previous properties imply that if $\langle e_{ij}, p_j\rangle \in s.A$ then $pred_{s.A}(e_{ij}) \neq \emptyset$, $e_{ij} \notin victims$ and the initiator of a mark of $\langle e_{ij}, p_j\rangle$ is always in P. Therefore $lose\_mark(m, \langle e, p\rangle)$ actions are disabled in every execution on the automaton, and they can be renaming as $null$ actions.

The particular action $create\_mark(\langle p, e\rangle)$ requieres further work.

Lemma 3.5. *Let $\alpha \in fairexecs(R_1)$, such that $beh(\alpha) \in H_1$. If there exists a persistant cycle* **C** *then there exists a state $s_{k'}$ in α with $k' > k$ and an arc $\langle e_{i,i+1}, p_{i+1}\rangle \in$C, such that mark $m = (p_{i+1}, \langle e_{i,i+1}, p_{i+1}\rangle, v)$ with $v = s_k.version(\langle e_{i,i+1}, p_{i+1}\rangle)$ is a mark of $\langle e_{i,i+1}, p_{i+1}\rangle$.*

Proof. Without loss of generaliy, let s_k be the state of α when the cycle C$\subseteq s_k.A$ has just been formed. Let $\pi_k = add\_arc(\langle . \rangle)$ be the last action which closes the cycle. By the well formed histories H_1 such arc$\langle . \rangle$ is of type $\langle e_{i,i+1}, p_{i+1}\rangle$. So $\pi_k = add\_arc(\langle e_{i,i+1}, p_{i+1}\rangle)$, and this implies that since s_k $create\_mark(\langle e_{i,i+1}, p_{i+1}\rangle)$ will be enabled. Fairness then ensures its execution.

The Lemma 3.5 ensures that at least, when a cycle has been formed, one action $create\_mark(\langle e, p\rangle)$ had been executed. In González de Mendívil *et al.* (1994) is proved that this guarantees that an action $abort(.)$ will eventually be enabled in order to resolve the cycle. Therefore, the actions $create\_mark(\langle p, e\rangle)$ are unecessary in R_{1e}.

Finally, the automaton R_{1e} is obtained from R_1 by the transformations presented above.

– States of $R_{1e}.states(R_{1e}) = \{s \in states(R_1)$ / s is reachable state by a behaviour $\beta \in H_1\}$.

– Start States of $R_{1e}.start(R_{1e}) = start(R_1)$.

– Action signature of R_{1e}. Its external signature is the same as R_1 and its internal actions are $int(R_{1e}) = \{create\_mark(\langle e, p\rangle), trans\_mark(m, \langle p, e\rangle),$ $trans\_mark(m, \langle e, p\rangle),$ $elim\_mark(m, \langle p, e\rangle),$ $lose\_mark(m, \langle p, e\rangle),$ $null\}$, where $\langle p, e\rangle,$ $\langle e, p\rangle \in B$ and $m \in M$.

– Steps of $R_{1e}.steps(R_{1e}) = \{s, \pi, s') \in steps(R_1)/\pi \in acts(R_{1e})\}$.

– Partition of $R_{1e}.part(R_{1e}) = part(R_1)$ with the renaming of the actions.

The automaton R_{1e} may be interpreted as the execution module (Tuttle, 1987) of the $fairexecs(R_1)$ such that their behaviours are included in H_1. Under this consideration is trivial that $(fairbehs(R_{1e}) \cap H_1) = (fairbehs(R_1) \cap H_1)$. The criteria of correctness are maintained for the automaton R_{1e}.

3.2. Distributed deadlock resolution algorithm. In this subsection, the concept of site and communication channel is introduced in order to obtain the distributed deadlock resolution algorithm. Moreover the global knowledge is no longer used: any variable of state is explicitely local. However some global concepts are kept, such as a persistant cycle, defined to be a cycle whose arcs are locally persistant.

Given $P = \{p_i,\ i \in I\}$, a function $Site: P \rightarrow I$ associates its site to a node. From now on, we consider that each site has an unique process, and therefore indices are characteristic of the site. As a consequence, we will now simplficate $Site(p_i)$ by i, being i the i-th site of the system. The nodes $e_{ij} \in E$, belong to both sites i and j. Between two different sites there is always a reliable FIFO communication channel, denoted $channel_{j,i}$ by which the messages may travel form site j to site i.

The new automaton, denoted R_2, is similar to R_{1e} but with the additional renaming of actions $trans\_mark(m, \langle e, p\rangle)$ by $trans\_probe(m, \langle e, p\rangle)$. We use such names without confusion. The algorithm is edge-chasing and uses *probes* as the special messages for the resolution. A *probe s* is a pair $(m, \langle p_i, e_{ij}\rangle)$, being $\langle p_i, e_{ij}\rangle$ the arc that should receive the mark $m \in M$. The set of probes is denoted S, and S^* is the set of the finite strings σ built on S.

– States of R_2. The WFG is composed by the local graphs of the sites, denoted $WFG_i = (N_i, A_i)$ being $N_i = P_i \cup E_i,$ $P_i = \{p_i\}$ and $E_i = \{e_{ij}, e_{ji}\}$. Similar consideration is made for $create_i(.)$, $marks_i(.)$, $victims_i$. Parameters

```
                    INPUT ACTIONS
add_arc($(\langle p_i, e_{ij}\rangle)$)
  eff $s'.A_i \leftarrow s.A_i + \{\langle p_i, e_{ij}\rangle\}$
    $s'.version_i(\langle p_i, e_{ij}\rangle) \leftarrow s.version_i(\langle p_i, e_{ij}\rangle) + 1$
    $s'.create_i(e_{ij}) \leftarrow s.create_i(e_{ij}) + \{p_i\}$
add_arc($(\langle e_{ij}, p_j\rangle)$)
  eff $s'A_j \leftarrow s.A_j + \{\langle e_{ij}, p_j\rangle\}$
    $s'.version_j(\langle e_{ij}, p_j\rangle) \leftarrow s.version_j(\langle e_{ij}, p_j\rangle) + 1$
    $s'.create_j(p_j) \leftarrow s.create_j(p_j) + \{e_{ij}\}$
del_arc($(\langle p_i, e_{ij}\rangle)$)
  eff $s'.A_i \leftarrow s.A_i - \{\langle p_i, e_{ij}\rangle\}$
    $s'.marks_i(\langle p_i, e_{ij}\rangle) \leftarrow \varepsilon$
    $s'.create_i(e_{ij}) \leftarrow s.create_i(e_{ij}) - \{p_i\}$
del_arc($(\langle e_{ij}, p_j\rangle)$)
  eff $s'.A_j \leftarrow s.A_j - \{\langle e_{ij}, p_j\rangle\}$
    $s'.marks_j(\langle e_{ij}, p_j\rangle) \leftarrow \varepsilon$
    $s'.create_j(p_j) \leftarrow s.create_j(p_j) - \{e_{ij}\}$

                    OUTPUT ACTIONS
abort($p_i$)
  pre $p_i \notin victims_i$
    $\exists\langle p_i, e_{ik}\rangle \in A_i : marks_i(\langle p_i, e_{ik}\rangle) = m \bullet \mu,$
        $m = (p_i, \langle e_{ij}, p_i\rangle, v)$
        $\langle e_{ji}, p_i\rangle \in A_i \wedge v = version_i(\langle e_{ij}, p_i\rangle)$
  eff $s'.A_i \leftarrow s.A_i - incom_{s.A_i}(i)$
    $s'.victims_i \leftarrow s.victims_i + \{p_i\}$
    $\forall e_{ji} \in pred_{s.A_i}(p_i)$ :
        $s'.marks_i(\langle e_{ji}, p_i\rangle) \leftarrow \varepsilon$
    $s'.create_i(p_i) \leftarrow \emptyset$
```

Fig. 5. The automaton R_2 (to be continued).

```
                    INTERNAL ACTIONS
create_mark($(\langle e_{ij}, p_j\rangle)$)
    pre $p_j \notin victims_j,\ succ_{A_j}(p_j) \neq \emptyset,\ \langle e_{ij}, p_j\rangle \in A_j,$
      $e_{ij} \in create_j(p_j)$
    eff $m = (p_j, \langle e_{ij}, p_j\rangle, version_j(\langle e_{ij}, p_j\rangle))$
      $s'.marks_j(\langle e_{ij}, p_j\rangle) \leftarrow s.marks_j(\langle e_{ij}, p_j\rangle) \bullet m$
      $s'.create_j(p_j) \leftarrow s.create_j(p_j) - \{e_{ij}\}$
trans_mark($m, \langle p_i, e_{ij}\rangle$)
    pre $p_i \notin victims_i,\ \langle p_i, e_{ij}\rangle \in A_i,\ pred_{A_i}(p_i) \neq \emptyset$
      $marks_i(\langle p_i, e_{ij}\rangle) = m \bullet \mu$
         $m = (r\langle l, r\rangle, v),\ r > p_i$
    eff $s'.marks_i(\langle p_i, e_{ij}\rangle) \leftarrow \mu$
      $\forall e_{ki} \in pred_{s.A}(p_i)$ :
         $s'.marks_i(\langle e_{ki}, p_i\rangle) \leftarrow s.marks_i(\langle e_{ki}, p_i\rangle) \bullet m$
elim_mark($m, \langle p_i, e_{ij}\rangle$)
    pre $p_i \notin victims_i,\ \langle p_i, e_{ij}\rangle \in A_i,\ pred_{A_i}(p_i) \neq \emptyset$
      $marks_i(\langle p_i, e_{ij}\rangle) = m \bullet \mu$
         $m = (r\langle l, r\rangle, v),\ r < p_i.$
    eff $s'.marks_i(\langle p_i, e_{ij}\rangle) \leftarrow \mu$
      $s'.create_i(p_i) \leftarrow s.create_i(p_i) + pred_{s.A_i}(p_i)$
lose_mark($m, \langle p_i, e_{ij}\rangle$)
    pre $marks_i(\langle p_i, e_{ij}\rangle) = m \bullet \mu$
         $m = (r\langle l, r\rangle, v)$
      $pred_A(p_i) = \emptyset \vee p_i \in victims_i \vee$
         $(r = p_i \wedge \langle l, r\rangle \in A_i \wedge v \neq version_i(\langle l, r\rangle))$
    eff $s'.marks_i(\langle p_i, e_{ij}\rangle) \leftarrow \mu$
trans_probe($m, \langle e_{ij}, p_j\rangle$)
    pre $\langle e_{ij}, p_j\rangle \in A_j$
      $marks_j(\langle e_{ij}, p_j\rangle) = m \bullet \mu$
    eff $s'.marks_j(\langle e_{ij}, p_j\rangle) \leftarrow \mu$
      $s = (m, \langle p_i, e_{ij}\rangle)$
      $s'.channel_{j,i} \leftarrow s.channel_{j,i} \bullet s$
```

Fig. 5. The automaton R_2 (to be continued).

```
**receive_probe**$(m, \langle p_i, e_{ij} \rangle)$
  pre $channel_{j,i} = s \bullet \sigma$
  $s = (m, \langle p_i, e_{ij} \rangle)$
  eff $s'.channel_{j,i} \leftarrow \sigma$
  $s'.marks_i(\langle p_i, e_{ij} \rangle) \leftarrow s.marks_i(\langle p_i, e_{ij} \rangle) \bullet m$
```

Fig. 5. The automaton R_2.

are excluded for clearance. The $channel_{j,i}$ with $i \neq j$, is the new part of the state; it contains the probes travelling from j to i.

– States of $R_2.s_0 \in start(R_2)$ if $\forall i \in I\colon s_0.A_i = \varnothing$; $s_0.victims_i = \varnothing$; $s_0.version_i(.) = 0$; $s_0.marks_i(.) = \varepsilon$; $s_0.create_i(n) = \varnothing$; and $\forall j \in I\colon s_0.channel_{j,i} = \varepsilon$.

– Actions signature of R_2. Its external signature is the same as R_{1e} and its internal actions are $int(R_2) = int(R_{1e}) \cup \{receive\_probe(m, \langle p, e \rangle)\}$, where $int(R_{1e})$ is taken after its action renaming.

– States of R_2. The steps of R_2 are defined in Fig. 5.

– Partition of R_2. Its classes are the same as R_{1e}, plus aditional classes F_2^{ij} being $\forall \langle p_i, e_{ij} \rangle \in B$, $F_2^{ij} = \{receive\_probe(m, \langle p_i, e_{ij} \rangle) \mid \forall m \in M\}$.

By inspecting the automaton R_2, it is clear that it's simple exercise to find out the automata, denoted R_{2i} for each site, such that the composition for every site $(\Pi_{i \in I} R_{2i})$, itself composed by the communication automata S is equal to R_2. For example, the Fig. 6 shows an IOA, denoted S, satisfying the specification of the communication net. One can note that the partition of such automaton corresponds with the classes F_2^{ij}.

By hiding the actions due to the communication of probes among the automata $\Phi = \{trans\_probe(.),\ receive\_probe(.)\}$, it is simple to show that $R_2 = Hide_\Phi((\Pi_{i \in I} R_{2i}).S)$. It is also obvious that $ext(R_2) = ext(Hide_\Phi ((\Pi_{i \in I} R_{2i}).S))$, then we only need to prove that $(fairbehs(R_2) \cap H_1) \subseteq (fairbehs(R_{1e}) \cap H_1)$. If the property holds then it is assured that the proposed distributed solution is correct for the resolution of deadlocks.

4. Proof of correctness. In order to prove that the automaton R_2 satisfies R_{1e} under the well formed histories H_1, it is necessary to show the existence of a possibilities mapping (Tuttle, 1987). Such mapping is a function defined as $h\colon states(R_2) \to \mathcal{P}(states(R_{1e}))^1$. The existence of the possibilities mapping

INPUT ACTIONS	OUTPUT ACTIONS
trans_probe$(m, \langle e_{ij}, p_j \rangle)$	**receive_probe**$(m, \langle p_i, e_{ij} \rangle)$
eff $s = (m, \langle p_i, e_{ij} \rangle)$	pre $channel_{j,i} = s \bullet \Sigma$
$s'.channel_{j,i} \leftarrow s.channel_{j,i} \bullet s$	$s = (m, \langle p_i, e_{ij} \rangle)$
	eff $s'.channel_{j,i} \leftarrow \Sigma$

Fig. 6. The automaton S.

guarantees that the preconditions of the actions in R_2 implies the preconditions of the actions in R_{1e}. In the following *null* actions and the actions $abort(e_{ij})$ are not taken into account as it was indicated in the Subsection 3.1.

DEFINITION 4.1. If $s \in states(R_2)$ then $h(s) = \{t \in states(R_{1e})/ P_h(s,t)$ property holds$\}$. $P_h(s,t)$ is defined by the following conditions:

(1a) $\langle p_i, e_{ij} \rangle \in t.A \iff \langle p_i, e_{ij} \rangle \in s.A_i$;

(1b) $\langle e_{ij}, p_j \rangle \in t.A \iff \langle e_{ij}, p_j \rangle \in s.A_j$;

(2) $p_i \in t.victims \iff p_i \in s.victims_i$;

(3a) $t.version(\langle p_i, e_{ij} \rangle) = v \iff s.version_i(\langle p_i, e_{ij} \rangle) = v$;

(3b) $t.version(\langle e_{ij}, p_j \rangle) = v \iff s.version_j(\langle e_{ij}, p_j \rangle) = v$;

(4a) $t.(\langle p_i, e_{ij} \rangle) = \mu' \bullet \mu$; $\mu, \mu' \in M^*$ being $\mu = m_1 \dots m_k \dots \iff s.marks_i(\langle p_i, e_{ij} \rangle) = \mu' \wedge s.channel_{j,i} = s_1 \dots s_k \dots$ beeing $s_k = (m_k, \langle p_i, e_{ij} \rangle)$;

(4b) m in $t.marks(\langle e_{ij}, p_j \rangle) \iff m$ in $s.marks_j(\langle e_{ij}, p_j \rangle)$;

(5a) $p_i \in t.create(e_{ij}) \iff p_i \in s.create_i(e_{ij})$;

(5b) $e_{ij} \in t.create(p_j) \iff e_{ij} \in s.create_j(p_j)$.

ASSUMPTION 4.2. The communication channels are reliable, that is, messages arrive in finite time and messages between two sites are received in the same order as they were sent (FIFO). It is supposed that the manager and the resolution algorithm share the same communication channel between two sites, if the algorithm transmits information as a result of an action over an arc (for

[1] To define a possibilities mapping $h{:}states(R_2) \rightarrow \mathcal{P}(states(R_{1e}))$ under the restriction imposed by the behaviours in H_1 is equivalent to define a possibilities mapping $h{:}states(R_2.M_1) \rightarrow \mathcal{P}(states(R_{1e}.M_1))$ being M_1 the automaton which represents the manager whose $fairbehs(M_1) = H_1$, and where $R_2.M_1$ is the composition of the manager and the resolution algorithm in order to form a non-deadlocked system.

example, send a probe) and later the manager transmits information as a result of an operation over that arc (for example, delete the arc), that information will be received and processed in that order in the destination site. Thus, in the proposed example, this assumption assures that a certain arc still remains when it has received a probe.

Theorem 4.3. *The mapping h (Definition 4.1) is a possibilities mapping under the well formed histories H_1.*

Proof. (1) $ext(R_2) = ext(R_{1e})$ by definition of the automata.

(2) It is obvious observing the definitions of starting state of R_2 and R_{1e} that the mapping h verifies $\forall s_0 \in start(R_2)$, $\exists t_0 start(R_{1e}: t_0 \in h(s_0)$.

(3) Let s be a reachable state of R_2, $t \in h(s)$ a reachable state of R_{1e}, and $(s, \pi, s') \in steps(R_2)$. We claim that (i) if $\pi \in acts(R_{1e})$ then $(t, \pi, t') \in steps(R_{1e}$ with $t' \in h(s')$, and (ii) if $\pi \notin acts(R_{1e})$ then $t \in h(s')$. The proof is obvious from the preconditions and effects of the actions of R_2 and R_{1e}. However the actions $del\_arc(\langle p_i, e_{ij})$ and $receive\_probe(m, \langle p_i, e_{ij}\rangle)$ require further work.

Case 1. $\pi = del\_arc(\langle p_i, e_{ij}\rangle)$, $\pi \in acts(R_{1e})$. $s'.A_i = s.A_i - \{\langle p_i, e_{ij}\rangle\}$ $\equiv t'.A = t.A - \{\langle p_i, e_{ij}\rangle\}$, by (1a). $s'.mark_i(\langle p_i, e_{ij}\rangle) = \varepsilon$, due to the well formed histories H_1, the arc $\langle e_{ij}, p_j\rangle$ has been deleted and no probe $s = (m, \langle p_i, e_{ij})$ created by the arc $\langle e_{ij}, p_j\rangle$ can have been sent after the arc had been deleted, if s in $s.channel_{ji} \equiv t'.marks(\langle p_i, e_{ij}) \neq \varepsilon$, due to (4b), which is a contradiction. The Assumption 4.2 assures the arc $\langle p_i, e_{ij}\rangle$ is not spontaneously deleted and therefore, only it can be deleted if $s.channel_{j,i} = \varepsilon$. So $t \in h(s)$, $t' \in h(s')$ and $(t, \pi, t') \in steps(R_{1e})$.

Case 2. $\pi = receive\_probe(m, \langle p_i, e_{ij}\rangle)$, $\pi \notin acts(R_{1e})$. $s = (m, \langle p_i, e_{ij}\rangle)$, s in the head of $s.channel_{j,i} \equiv m$ in the head of $t.marks(\langle p_i, e_{ij}\rangle)$, due to (4a). In addition by Assumption 4.2, the arc $\langle p_i, e_{ij}\rangle$ there exists. By the execution of $\pi.m \in s'.mark_i(\langle p_i, e_{ij}\rangle) \equiv m \in t.marks(\langle p_i, e_{ij}\rangle)$, due to (4a). Therefore, $t \in h(s)$, $t \in h(s')$.

In the following Theorem 4.4, we use the generic classes:

$C_1 = \{abort(p_i)\}$;

$D_1 = \{create\_mark(\langle e_{ij}, p_j\rangle)\}$;

$E_1 = \{trans\_mark(m, \langle p_i, e_{ij}\rangle), lose\_mark(m, \langle p_i, e_{ij}\rangle),$
$elim\_mark(m, \langle p_i, e_{ij}\rangle)\}$;

$E_1 = \{trans\_mark(m, \langle e_{ij}, p_j\rangle)$

(or $trans\_probe(m, \langle e_{ij}, p_j\rangle)$ by renaming)};

$F_2 = \{receive\_probe(m, \langle p_i, e_{ij}\rangle)\}$.

The four initial classes are in $part(R_{1e})$, and $part(R_2) = part(R_{1e}) \cup \{F_2\}$.

Theorem 4.4. *Let R_2 and R_{1e} be the automata previously defined and let h be the possibilities mapping from R_2 to R_{1e} (Definition 4.1). If well formed histories H_1 (Definition 3.1) are verified then $(fairbehs(R_2) \cap H_1) \subseteq (fairbehs(R_{1e}) \cap H_1)$.*

Proof. Let $\alpha \in fairexecs(R_2)$, such that beh$(\alpha) \in H_1$. Due to the results[2] in Tuttle (1987) $\exists \beta \in execs(R_{1e})$: $\beta C_h \alpha$. We claim that it is enough to prove that β is a fair execution of R_{1e}. In that case, since there exists a possibilities mapping from R_2 to R_{1e}, sched$(\alpha)|acts(R_{1e})$ = sched(β) (Tuttle, 1987); moreover, since $ext(R_2) = ext(R_{1e})$, it can be shown that beh(α) = beh(β) and beh$(\beta) \in H_1$. Therefore, $(fairbehs(R_2) \cap H_1 \subseteq (fairbehs(R_{1e}) \cap H_1)$.

Let $\alpha \in fairexecs(R_2)$: beh$(\alpha) \in H_1$, and $\beta \in execs(R_{1e})$: $\beta C_h \alpha$. We claim $\beta \in fairexecs(R_{1e})$.

Case 1. β is finite. Assume β is not fair. Let be $\beta = t_0 \ldots \pi_r t_r$. A prefix of β is denoted β_r if its endstate is t_r, so $\beta = \beta_r$. As β is not a fair, a locally controlled action π must be enabled in the state t_r, that is, there is a class C_1, D_1, or E_1 such that $\pi \in C$ (C is one of such classes).

Case 1.1. Let α be a finite execution, $\alpha = s_0 \pi_1 s_1 \ldots \pi_k s_k$, that is $\alpha = \alpha_k$.

Let α_r be the first prefix such that $t_r \in h(s_r)$. Then $\forall r \leqslant q \leqslant k$, $t_r \in h(s_q)$. For an action π in the classes E_1 or D_1 if π is enabled in t_r is also enabled in s_k, and then α is not fair, contradicting the hypothesis. For the action π in the class C_1, $\pi = abort(p_i)$, as it is enabled in t_r, there is a cycle

[2] From Tuttle (1987): Let h be a possibilities mapping from automaton A to automaton B. Let α and β be finite executions of A and B respectively. It is said that (a) β *finitecorresponds* to α under h, denoted $\beta FC_h \alpha$, if $sched(\beta){=}sched(\alpha)|B$ and the final state of β is a possibility for the final state of α; (b) β corresponds to α under h, denoted $\beta C_h \alpha$, if for every finite prefix α_i of α there is a finite prefix β_i of β such that $\beta_i FC_h \alpha_i$ and β is the limit of the β_i; (c) Let h be a possibilities mapping from automaton A to automaton B. If α is an execution of A then there is an execution β of B such that $\beta C_h \alpha$.

$\mathbf{C} \subseteq t_r.A$, and by $h(.)$, $\mathbf{C} \subseteq \cup_i s_q.A_i.t_r.marks(\langle p_i, e_{i,i+1}\rangle) = m \cdot \mu$ where m is the mark to enable the abort action. By $h(.), s_k.marks_i(\langle p_i, e_{i,i+1}\rangle) = m \cdot \mu$ or $s_k.marks_i(\langle p_i, e_{i,i+1}\rangle) = \varepsilon$ and $s_k.channel_{i+1,i} = s \ldots$ being $s = (m, \langle p_i, e_{i,i+1}\rangle)$. By the preconditions of $receive\_probe(m, \langle p, e\rangle)$, such action is enabled in s_k and then α is not fait, contradicting the hypothesis.

Case 1.2. Let α be an infinite execution, α_r its first prefix such that $t_r \in h(s_r)$. Then $\forall q \geqslant r,\ t_r. \in h(s_q)$. For every action π in the classes E_1 or D_1 if π is enabled in t_r, it is also enabled in s_q, otherwise this action must appear in β, what is impossible. Therefore, α is not fair because there exists a class infinitely enabled whose actions are never performed. For the action π in the class $C_1, \pi = abort(p_i)$, as it is enabled in t_r, a cycle $\mathbf{C} \subseteq t_r.A$, and by $h(.)$, $\forall q \geqslant r$, $\mathbf{C} \subseteq \cup_i s_q.A_i.\ t_r.marks(\langle p_i, e_{i,i+1}\rangle) = m \ldots$ and by $h(.)$, $s_q.marks_i(\langle p_i, e_{i,i+1}\rangle) = m \ldots$ or $s_q.marks_i(\langle p_i, e_{i,i+1}\rangle) = \varepsilon$ and $s_q.channel_{i+1,i} = s \ldots$, being $s = (m, \langle p_i, e_{i,i+1}\rangle)$. By the preconditions of $receive\_probe(m, \langle p_i, e_{i,i+1}\rangle)$, and the fairness assumption of α, there exists $q' > r$ being $\pi_{q'} = receive\_probe(m, \langle p_i, e_{i,i+1}\rangle)$. In $s_{q'}, abort(p_i)$ is henceforth enabled (it can not be executed, otherwise it would appear in β) and, α is not fair, contradicting the hypothesis.

Case 2. $\beta \in execs(R_{1e})$ is infinite (α is necessarily infinite). As α is fair, due to the fairness definition that is assumed in the IOA model (Tuttle, 1987) there are two possible cases:

Case 2.1. Action $\pi \in C_1, D_1, E_1$, or F_2 appear infinitely often in α. Action $\pi \in C_1, D_1$, or E_1 appear infinitely often in β; therefore, β is a fair execution.

Case 2.2. States in which no action of C_1, D_1, E_1, or F_2, is enabled appear infinitely often in α. The proof for the actions of C_1, D_1, E_1, is trivial because if there exists a state in α in which an action of the those classes is not enabled, then it is also disabled in a state in β, by definition if the mapping h. Suppose β is not fair due to the fact that an action $abort(p_i)$ is enabled since t_r, and never performed. As $\beta C_h \alpha$, there exists a pprefix α_r such that $t_r \in h(s_r)$. As $abort(p_i)$ is not executed in β, the same happens in α, and therefore, there is a persistant cycle $\mathbf{C}$ in α since s_r. Let $q > r$, by the preconditions of $abort(p_i)$ in R_{1e}, m is the head of $t_r.marks(\langle p_i, e_{i+1}\rangle)$; by using $h(.), s_q.marks_i(\langle p_i, e_{i,i+1}\rangle) = m \cdot \mu$ or $s_q.marks_i(\langle p_i, e_{i,i+1}\rangle) = \varepsilon$ and $s_q.channel_{i+1,i} = s \ldots$, being $s = (m'\langle p_i, e_{i,i+1}\rangle)$. Since α is fair

$receive\ probe(m, \langle p_i, e_{i,i+1}\rangle)$ will be executed for some $q' > r$. The action $abort(p_i)$ is enabled for all $q'' \geqslant q'$ in the states of α. Therefore, α is not fair execution, what contradicts the hypothesis.

5. Conclusions. In this work, we have presented the first attempt in the literature to provide a hierarchical correctness proof for a distributed deadlock resolution algorithm. The proposed method for approach to the deadlock resolution problem based on the Input-Output Automata Model (Lynch and Tuttle, 1989; Tuttle, 1987) allows the development of the solution in a natural way of successive levels of abstraction (correctness is proven by level satisfaction). This approach is in contrast with the invariant technique of proof given in Kshemkalyani and Singhal (1991) for a deadlock resolution algorithm where the gap between the specification of the problem and the final solution is large. An interesting contribution of our work is the specification of the deadlock resolution from the natural point of view of the resolution, and not from that the (intermediate) detection, which has induced the development of erroneous algorithms. In the question of the use the Wait-For-Graph (WFG) or the use of the Resource-Allocation-Graph (RAG) (Singhal, 1989), we can say that both of them are suitable for the problem of deadlock in the single request model but authors which uses the RAG also needs an implicity WFG in the proof of correctness (Knapp, 1987; Kshemkalyani and Singhal, 1991; Elmagarmid, 1988). In our work, RAG may be used without additional complexity.

Finally, this work provides an adequate methodology for the study and development of another deadlock resolution algorithms for different resource allocation models as the AND model. In a future work, we intend the application of the hierarchical methodology to prove the correction of a resolution algorithm for the AND model which is an extension of the detection algorithm in González de Mendívil *et al.* (1992).

REFERENCES

Chandy, K.M, and J.Misra (1982). A distributed algorithm for detecting resource deadlocks in distributed systems. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. pp. 157–164.

Choudhary, A.L, W.H.Kohler, J.A.Stankovic and D.Towsley (1989). A modified priority-based probe algorithm for distributed deadlock detection and resolution. *IEEE Trans-*

actions on Software Engineering, **15**, 10–17.

Elmagarmid, A.K. (1988). A distributed deadlock detection and resolution algorithm and its correctness. *IEEE Transactions on Software Engineering*, 1443–1452.

González de Mendívil, J.R., J.R.Garitagoitia and A.Córdoba (1992). A simple distributed deadlock detection algorithm. In *Proceedings of the 6-th Annual European Computer Conference, The CompEuro92.* Hague, pp. 497–502.

González de Mendívil, J.R., C.F.Alastruey and J.R.Garitagoitia (1993). A distributed deadlock detection algorithm for the AND model. *Microprocessing and Microprogramming*, **38**, 385–392.

González de Mendívil, J.R., J.Bernabeu, J.R.Garitagoitia and Federico Fariña (1994). *A Deadlock Resolution Algorithm Based on the Input-Output Automata Model.* Polytechnic University of Valencia, DSIC-II/9/94.

Knapp, E. (1987). Deadlock detection in distributed databases. *ACM Computing Surveys*, **3**(4), 303–328.

Kshemkalyani, A.D., and M.Singhal (1991). Invariant-based verification of a distributed deadlock detection algorithm. *IEEE Transactions on Software Engineering*, **17**(8), 789–799.

Lynch, N., and M.Tuttle (1989). An introduction to input/output automata. *CWI Quarterly*, **2**(3), 219–246.

Mitchell, D.P., and M.J.Merritt (1984). A distributed algorithm for deadlock detection and resolution. In *Proceedings of ACM Conference on Principles of Distributed Computing.* Vol. 8. pp. 282–284.

Obermarck, R.L. (1982). Distributed deadlock detection algorithm. *ACM Transactions on Database Systems*, **6**, 187–210.

Singhal, M. (1989). Deadlock detection in distributed systems. *IEEE Computer*, **1**, 37–48.

Sinha, M.K., and N.Natarajan (1985). A priority based distributed deadlock detection algorithm. *IEEE Transactions on Software Engineering*, **SE-11**(1), 67–80.

Tuttle, M.R. (1987). Hierarchical Correctness proofs for distributed algorithms. *MIT*, **8**.

Received November 1994

José R. González de Mendívil was born in the Basque Country in 1963. He received the degree of Physicist (area: Electronics and Control) from the University of the Basque Country in 1987, and the degree of Doctor in Physics from the same university in 1993. He is professor in the Department of Electricity and Electronics in the University of the Basque Country. His research interests are mathematical and computing modelling of functional differential equations, control and identification of delay-differential systems and control of industrial processes.

José Bernabeu received his Master in Physics from the University of Valencia and his MS in computer science and PhD from the Georgia Institute of Technology (USA) in 1988. He has been a research scientist at Georgia Tech., and currently is a professor at the Polytechnic University of Valencia (Spain), and senior staff engineer at Sun Microsystems Laboratories (USA). His research interest cover the area of distributed computing, coherency of shared memory as well as operating system design.

Akim Demaile was born in France in 1970. After having been graduated from the Polytechnic school, he has been admitted in Telecom Paris. He has been admitted as visit student in the Department of the Electricity and Electronics (University of the Basque Country) during 1994. His research interest cover the area of distributed systems.

PADĖČIŲ BE IŠEITIES IŠSKYRIMO ALGORITMAS

José Ramón GONZÁLEZ DE MENDÍVIL, José BERNABEU,
Carlos F. ALASTRUEY, Akim DEMAILE

Nagrinėjama padėčių be išeities išskyrimo problema sistemose su vienos paraiškos modeliu. Pasiūlyta naujas algoritmas problemai spręsti ir hierarchinis šio algoritmo korektiškumo įrodymo metodas.