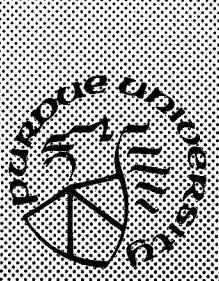
6-1-1988

Complete Specification of DDM Mechanisms

Thomas L. Casavant
Purdue University, tomc@ecn.purdue.edu

Weng H. Cheong
Purdue University, weheong@ecn.purdue.edu

Ali Sajassi
Purdue University, nahid@ecn.purdue.edu



Complete Specification of DDM Mechanisms

Thomas L. Casavant
Weng H. Cheong
Ali Sajassi

TR-EE 88-24
June 1988

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Complete Specification of DDM Mechanisms

Thomas L. Casavant : tomc@ecn.purdue.edu
Weng H. Cheong : wcheong@ecn.purdue.edu
Ali Sajassi : nahid@ecn.purdue.edu

School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

ABSTRACT

The specification of DDM (Distributed Decision-Making) algorithms is addressed. The modeling technique presented is based on well-known extensions to Petri-Nets (PNs). Transition-enabling functions with a domain corresponding to the marking of a net are used to express the semantics of decision-making. Furthermore, the algorithm structural characteristics of global state representation and topology are incorporated. Finally, the dynamic nature of evolution of system state, interaction with processes external to the computation, and the interprocess communication aspects of the mechanism are also modeled. The elements of analysis associated with this model are described, but not detailed in the scope of this paper.

1. Introduction

Distributed Decision-Making (DDM) algorithms represent a large class of computations critical to the reliable operation of distributed computing systems. However, to date, there have been no unified techniques for modeling which incorporate *all* of the salient features of these computations. Furthermore, successful specification and modeling of this class of computations may provide a basis for formalisms which deal with the class of all distributed computations.

This paper describes a complete specification technique centered on the considerably powerful modeling capabilities of extended Petri-Nets [Pet77, Pet81]. The extensions to Petri-Nets are in the dimension of computation modeling power [Cia87] rather than in the time dimension. In particular, we model the semantics and structure of DDMs with

This work is supported by the National Science Foundation under Grant Numbers ECS-8800910 and CCR-8717895.

the PN class as extended by the use of *inhibitor arcs*. The use of this extension is then made much more practical from a specification point of view by using the notion of *transition enabling functions*. The scope of this paper is limited to specification power only. Analytical use of this modeling technique appears elsewhere.

2. Class of Mechanisms Modeled

The paradigm of **Distributed Decision-Making (DDM)** [TeS81a, TeS81b] has been examined in a number of forms (e.g., [Ho80, Sta85]). We define a **DDM algorithm** as a distributed computation in which a multiplicity of independent, physically distributed, autonomous modules of computation work cooperatively to make decisions. This is a logical extension of Enslow's definition [Ens78] of a distributed computing system. We further define the decision-making to be an ongoing process in which individual modules attempt to utilize a local view of global state (based on limited information) to make independent decisions aimed at meeting global objectives. Examples of DDM include decentralized control [Lar79, Sta80], distributed data-base management [BeG81], simulation utilizing cellular automata [FrH85, HaD76, SaW85, ThM86], and distributed fault diagnosis [HoK84]. Distributed scheduling [CaK88] will be used in this paper as an example of the use of this model.

3. Properties Modeled

In short, we attempt to precisely model all *structural* and *semantic* characteristics of this class of computation. Structural description is that part of overall algorithm description which remains unchanged throughout algorithm execution. The structure of an algorithm may be thought of as the framework or lattice which supports the semantics of the algorithm. The structure of sequential algorithms would normally include the data structures employed. Similarly, the structural description of DDM algorithms

include topology of decision-making entities, static components of state, and some representation of static global knowledge at each node such as the number and location of other nodes in the system. Semantic characteristics are those which describe the actions of the algorithm during its execution — the effect that the distributed components have on their environment.

More specifically, we define five DDM characteristics which must be modeled.

1. *Topological structure of the computation.* Since DDM mechanisms must explicitly communicate in order to function cooperatively, the underlying message-passing structure may be defined. This may correspond to the actual topology of the physical processor interconnection, or the logical connection of the decision-making entities.

2. *Communication process/events.* As noted earlier, explicit message-passing must take place to support cooperation in the DDM process. In addition to the structure of the interconnection (characteristic 1), the timing, coordination, and meaning of individual communication *events* must be modeled. In the modeling technique described here, we define equivalence classes of states (PN markings) to focus attention on the central role of message-passing and the events defined by this process.

3. *Semantics of the DDM.* The decision making policy itself is represented typically by a collection of relations which define the decisions to be made based on a combination of static (structural) information and current (dynamic) state information.

4. *Evolution of system state.* Since the dynamic state of the system to be controlled is an input to the DDM process, the model must include a representation of this aspect. The state may evolve as a result of either the DDM process itself, or from the influence of an external (stochastically perceived by the DDM mechanism).

5. *System-external world interaction* As mention in 4, some external process affects the state of the system dynamically. This modeling technique supports specification of the interface between the system state and an arbitrary external process which may

affect, or be affected by, the DDM process itself.

Other techniques to model computations of this nature have provided either complete specification of systems with a very carefully chosen set of states and events [MaF86], or mechanisms which have broader generality which use more than one modeling formalism to complete the specification [CaK86]. The distinguishing characteristics of the technique described here are the generality to any form of DDM which meets the definition in section 2, while remaining within a restricted and well-understood mathematical formalism — inhibitor-arc-enhanced PNs (PN_i).

4. Modeling Technique

The modeling technique proposed in this paper is based on the use of PNs [Pet81] to model each of the characteristics of the previous section. With this proposed technique, one can characterize each of the factors which influence the behavior of a distributed computation and system.

4.1. Topological Structure

In a distributed computing system, each DDM element is autonomous (and usually identical). This characteristic of a DDM relies directly on the definition of neighbor relation [CaK86]. The neighbor relation is used to model the interaction between a pair of DDM elements. Basically, this neighbor relation is expressed by the use of PNs with transition enabling (TE) functions which are the decision making processes with regards to its neighbors. For example, a distributed computation with five nodes may be represented as in Figure 1.

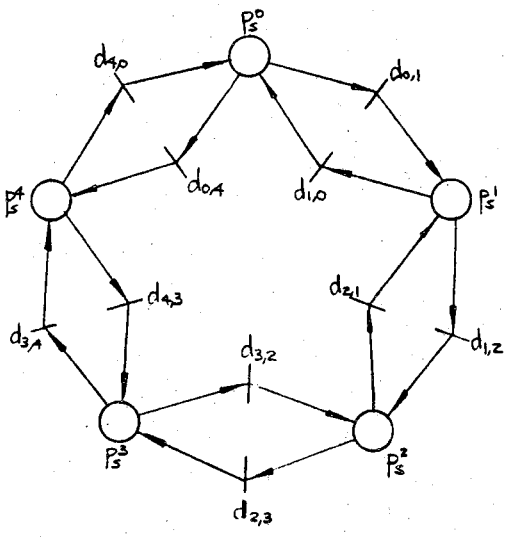


Figure 1. Petri-net representation of distributed computation.

Each node of the distributed computation is represented as a (set of) places in the PN representation. This is illustrated in Figure 1 as P_s^i. The decision-making process of each node with regard to one of its neighbors is represented as a TE function in the PN representation.

Definition 1: Given a PN representation of the topological structure of a distributed computation, we define the following:

$D_l(i) = \{...,1,0,1,...\}$ represents the set of possible decisions of v_i with respect to its neighbors.

$d_{i,j} \in D_l(i)$ denotes the decision of v_i with respect to v_j

4.2. Communication Process

In a DDM, the inter-node communication is represented by an abstraction called **phases** [CaK86a]. The phase concept is used to characterize the exchange of information between nodes at discrete points in time. This is a distinguishing feature of a distributed computing system in that information is sent periodically between cooperating elements

only when the sender chooses. The basis of the sender's decision to send information is modeled by a TE function in the PN representation of a DDM. For example, each node of a distributed computation with three phases may be represented as in Figure 2.

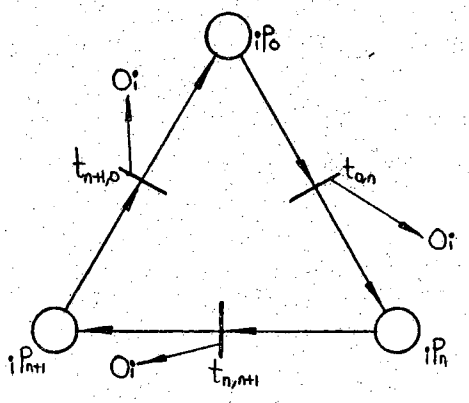


Figure 2. Petri-net representation of phases in distributed computation.

Each phase of a node is represented as a place in the PN representation. The basis of the decision of each node with regards to one its neighbor is represented by a transition in the PN representation. Hence, a change in phase can be used as a mechanism to cause a node to make a decision with regard to its neighbors.

Definition 2: Given a PN representation of the semantics of DDM, we define:

$P(i) = \{P_0 , P_n , P_{n+1}\}$ represents the set of possible current phase of node v_i.

${}_iP_j \in \mathrm{P(i)}$ denotes node v_i has P_j as its current phase. $t_{i,j}$ denotes the basis used in making a decision by a node to change from phase P_i to phase P_j.

O_i denotes output of a transition to node v_i.

4.3. Semantics

The semantics of a DDM are embedded within an instance of a PN model. The elements of the PN model consist of places and transitions. First, a place in the PN model is used to represent either an autonomous and identical DDM node or a phase that is associated with one of the DDM nodes. In general, each DDM node can be interpreted,

for example, as a logical unit of memory in a distributed database management system. In particular, a token in the place representing a particular phase denotes that the node is currently in that phase. Further, there can be only one phase-place which will have a token at any point at each node. Second, the decision-making process semantics are embedded in the transitions, $d_{i,j}$, where each $d_{i,j}$ has a TE function associated with it. Note that a PN with TE functions can be translated to a PN with inhibitor arcs [Cia87] and PN with inhibitor arcs can be further translated to a Turing Machine. However, for ease of exposition of the essence of a distributed computation, PNs with TE functions are used.

4.4. System State Evolution

For DDMs, each node is allowed to influence its neighboring nodes. With this PN modeling technique, the nature of influence of a node on its neighboring nodes can be modeled by a set of criteria which must be satisfied before any change to the state of a neighboring node is allowed. In particular, this set of criteria is embedded in the TE function that is associated with the transition, $d_{i,j}$, that provide the link between node v_i and its neighbor, v_j. In addition, permission of the neighboring node to allow another node to influence its state is also embedded in the TE function just described. Therefore, the set of criteria expressed in the TE function must be completely satisfied and permission from the neighboring node must be given before a node is allowed to change the state of one of its neighboring nodes. Further, the state of each node can be represented in the PN model as the number of tokens in the place representing the node. For example, a distributed computing system with five nodes and the condition that each node is allowed to influence one of its neighbor only when its phase changes from P_0 to P_n may be represented as in figure 3.

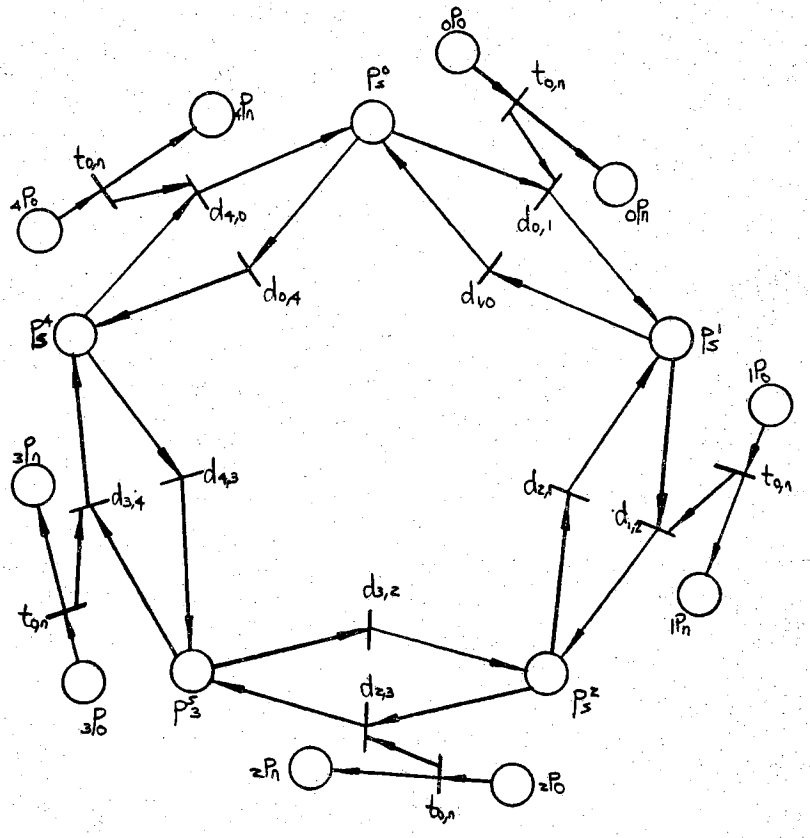


Figure 3. Petri-net representation of a distributed computation with five nodes and two phases.

4.5. System/External World Interaction

With the use of the PN modeling technique, the interaction of a distributed computing system with the external world can be easily modeled. In detail, the external world is represented as input places, (e.g., nj_i as in Section 6), in the PN representation. Each computation node interacts with the external world through a transition with a TE function associated with it in the PN model. The TE function is used to express the relationship between the input from the external world and the computation node. The TE function also imposes a limit on the use of the node by the external world. With regards to the influence on the external world by the computing system node, a transition with input places representing the (apparently) stochastic process carried out by a node of the

computing system is used to represent this relationship. The interaction of a distributed computation and the external world may be represented as in Figure 4.

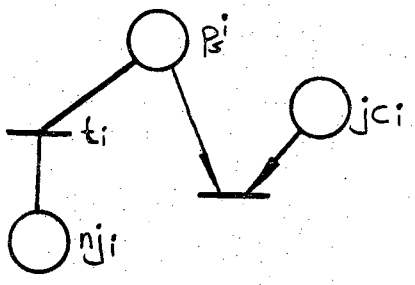


Figure 4. PN representation of the interaction between
a distributed computation and external world.

Definition 3: Given a PN representation of the system/external world interaction, we define the following:

I(i) = $\{nj_0, nj_1, ..., nj_n\}$ represents the set of input places for the external world.

O(i) = $\{jc_0, jc_1, ..., jc_n\}$ represents the set of input places to represent what appear as stochastic processes internal to a node of a DCS from the point of view of the DDM mechanism.

t_i denotes the link between input from external world to computation node v_i.

E_{t_i} denotes the TE function describing the relationship between the input place from the external world to the computation node v_i.

5. Forms of Analysis Supported

In this section we explain some of our motivations behind developments of analytical tools for a general class of extended Petri-Nets. A complete discussion on this subject will be presented elsewhere. Up to now, there has been extensive work on modeling and analytical power of restricted classes of PNs such as **State Machines, Marked Graphs, and Free Choice Nets.** Most analysis properties of these classes of PNs have been well formulated [Pet81,Rei83]. However, there is a trade-off between modeling power and analytical power within these classes; as the modeling power increases, analytical power decreases. For example, with state machines we acquire high analytical decision power (because they are equivalent to the finite state machine automata and formal language theory [Pet81]) but very limited modeling power because of its deficiency in modeling concurrency. Marked graphs, on the other hand, can model concurrency or waiting, which characterizes synchronization, but can not model conflict or data-dependent decisions. Most of the properties of marked graphs such as safeness, liveness, and reachability have been well investigated [Mur77, Rei83]. Free choice nets offer greater modeling power in the way they allow both the conflicts of state machines and the concurrency of marked graphs but in a more restricted way than the general PN. However, the analytical power of this class is less than that of state machines or marked graphs.

None of the above three classes have sufficient modeling power for representation of general computation models. In order to increase modeling power to account for any type of system, *zero* testing capabilities are required. It is been shown that PNs with zero testing capabilities have equivalent modeling power to Turing Machines, which in turn can represent any computable function [Pet81,Cia87]. By including additional extensions to the basic PN, we can achieve a high degree of flexibility in modeling power, but at the expense of analytical decision power. Several types of extended PN are classified by Ciardo [Cia87]. These extended PN are categorized as:

1) inhibitor arcs (PN_i)

2) transition priorities (PN_p)

3) enabling functions (PN_e)

4) variable input/output bags (PN_v)

with increased model specification flexibility from first to last. The fourth class is of little interest to us because no study on the analysis of net invariant or other logical properties of PN with variable input/output bags are available to this date. Although ease of express for PN_e is greater than for PN_p and PN_p is greater than for PN_i , they all have the same modeling power. Further, they are equivalent in the sense that reachability graphs for each of them are the same [Cia87].

In our modeling technique, we use PNs with enabling functions for their high degree of expressivity and then perform a transformation on the PN_e into their equivalent class-PN_i . From there we can partition the model further and analyze each subnet separately. For example, the subnet corresponding to the phase component of the DDM mechanism discussed in Section 6 can be analyzed separately. It can be seen that this subnet is live and conservative (total number of tokens in the set of places remains constant after each transition firing). These properties can be characterized by a linear system of equations and can be manipulated and analyzed easily.

Some of these properties can be related to some well defined terminologies of classical and modern control theory. For example, Murata [Mur77] defines *controllability* in terms of a transition-to-place incidence matrix — A, for any given PN, and states that a necessary conditions that a PN be completely controllable is rank(A)=p; where p is the number of places in that PN. Milner [Mil80] defines observability in terms of liveness of the net (every transition can be fired starting from any reachable marking). Stability and response time of a system can also be defined in terms of liveness of its PN model. In a stable system, the response of the system to externally induced perturbations in system state approaches an equilibrium state (whether optimal or not) after some time. In

a PN model of a system, stability can be considered to be the condition when some set of transitions are never enabled. For instance, in the load-balancing example of Section 6, a decision transition of a processor can become disabled if loads of neighboring processors are in balance with respect to the processor's internal load (which is indication of steady state for the system). Response time of the system can also be defined in a related way in terms of number of transition firings before the sequence comes to halt. Since response time in the model example of Section 6 is measured in terms of number of decision's transition firings, it can be interpreted both in terms of absolute and relative measurements. However, relative measurements gives us a more accurate basis of comparison of one system to an another. We define relative response as the ratio between number of decision transition firings and total number of transition firings before the system arrives at equilibrium state or within some specified limits.

In conclusion, our objectives here are to develop transformations that preserve most properties of nets and transform an extended PN model (e.g. PN_e or PN_i)
into the restricted subclasses of PN where the analysis can be performed. In some instances, these transformations are not possible. In such cases tools maybe developed for direct analysis of a subset of the extended PN model.

6. Illustrative Example

In this section we will demonstrate the use of the modeling technique proposed in this paper for the complete specification of DDM mechanisms by an example dealing with distributed load-balancing [CaK86]. The concept of load may be abstract, and can represent a variety of different actual demands upon the schedulable resources in the system. Each node is responsible for scheduling its own resources and for making decisions regarding the commitment of its resources to the use of other processors in the system.

To fully capture all these salient features dealing with transitions, load transfers, and phase changes, a PN model with TE functions may be used as shown in Figure 5.

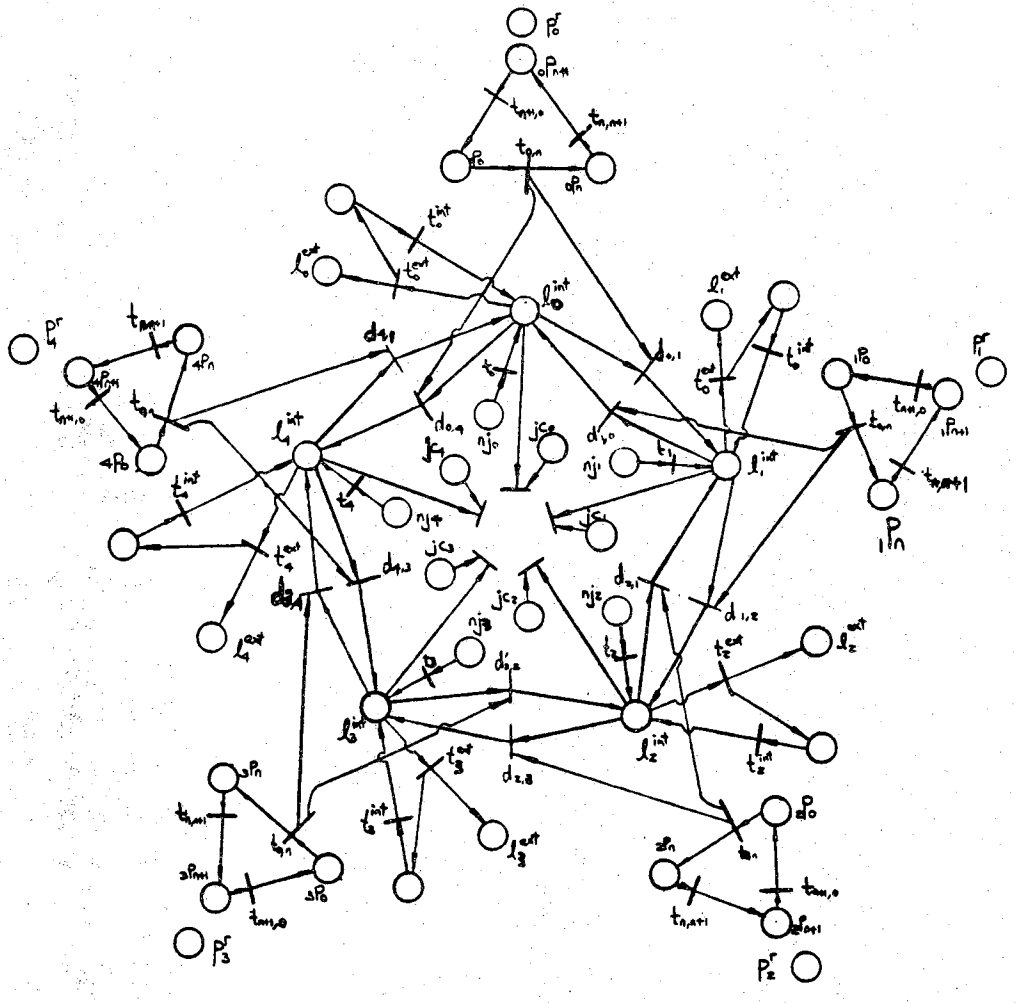


Figure 5. Petri net model for a distributed computation with 5 computation nodes and 3 phases associated with each node.

Definition 4: Given a PN model as in Figure 5, we define the following:

Places:

$$P_i^r = \begin{cases} n+2 & \text{if } \textit{last known phase of node } v_i \textit{ is } P_{n+1} \\ n+1 & \text{if } \textit{last known phase of node } v_i \textit{ is } P_n \\ 1 & \text{if } \textit{last known phase of node } v_i \textit{ is } P_0 \\ 0 & \textit{otherwise} \end{cases}$$

$${}_jP_i = \begin{cases} 1 & \text{if } \textit{node } v_j \textit{ is in phase } P_i \\ 0 & \textit{otherwise} \end{cases}$$

$$nj_i = \begin{cases} 1 & \text{if } \textit{new job arrives at node } v_i \\ 0 & \textit{otherwise} \end{cases}$$

$$jc_i = \begin{cases} 1 & \text{if } \textit{a job completion happens at node } v_i \\ 0 & \textit{otherwise} \end{cases}$$

l_i^{int} *denotes current load at node* v_i

l_j^{ext} *denotes the last known value of load of* v_j *as sent* v_i *by* v_j

Decisions: (neighbors of node v_i are nodes v_{n1} and v_{n2})

$$d_{i,n1} = \begin{cases} 1 & \text{if}((l_i^{int} < l_{n1}^{ext}) \text{ and } (l_{n1}^{ext} \leqq l_{n2}^{ext})) \\ 0 & \textit{otherwise} \end{cases}$$

$$d_{i,n2} = \begin{cases} 1 & \text{if}((l_i^{int} < l_{n2}^{ext}) \text{ and } (l_{n2}^{ext} < l_{n1}^{ext})) \\ 0 & \textit{otherwise} \end{cases}$$

Transition enabling functions:

$$E_{t_i} = \begin{cases} 1 & \text{if } l_i^{int} < \textit{maximum limit for node } v_i \\ 0 & \textit{otherwise} \end{cases}$$

$$E_{t_{0,n}} = \begin{cases} 1 & \text{if } \textit{phase } P_0 \textit{ is completed} \\ 0 & \textit{otherwise} \end{cases}$$

$$E_{t_{n,n+1}} = \begin{cases} 1 & \text{if } \textit{phase } P_n \textit{ is completed} \\ 0 & \textit{otherwise} \end{cases}$$

$$E_{t_{n+1,0}} = \begin{cases} 1 & \text{if } \textit{phase } P_{n+1} \textit{ is completed} \\ 0 & \textit{otherwise} \end{cases}$$

$$E_{t_i^{upd}} = \begin{cases} 1 & \text{if } \textit{updating of information on load value of node } v_i \textit{ is required} \\ 0 & \textit{otherwise} \end{cases}$$

$$E_{l_i^{int}} = \begin{cases} 1 & \text{if } \textit{updating process is completed} \\ 0 & \textit{otherwise} \end{cases}$$

From Figure 5, we can observe that the topological structure of the distributed load-balancing system, the semantics of each distributed node of computation, the communication process between neighboring nodes, the distributed system state evolution of each node, and the interaction between the distributed system and the external world are fully represented by this PN model with the relevant TE functions. In particular, l_i^{int} in Figure 5 corresponds to p_s^i in Figures 1 and 3.

The mathematical description of this algorithm in the notation of Communicating Finite Automata as shown in Figure 5 is included in the appendix. Take note of the correspondence between certain terms in the mathematical CFA description and the PN model:

$$P_i^{in} = {}_iP_0 \text{ or } {}_iP_n \text{ or } {}_iP_{n+1}$$

$$l_j^{ext} = {}_il_j^{ext}$$

7. Conclusions

This paper has presented a modeling technique which allows for complete specification of Distributed Decision-Making (DDM) algorithms. These computations represent a large class of computations critical to the reliable operation of distributed computing systems. Successful specification and modeling of this class of computations may provide a basis for formalisms which deal with the class of all distributed computations.

The technique described is centered on the modeling capabilities of extended Petri-Nets. These extensions are in the dimension of computation modeling power rather than in time. Semantics and structure of DDMs are modeled with the PN class as extended

by the use of *inhibitor arcs*. The use of this extension is then made much more practical from a specification point of view by using the notion of *transition enabling functions*. This paper has not covered in detail the analytical use of this modeling technique.

References

[BeG81] P.A. Bernstein, N. Goodman, "Concurrency Control in Distributed Data Base Systems," *Computing Surveys,* Vol. 13, No. 2, Jun. 1981, pp. 185-221.

[CaK86] T.L. Casavant and J.G. Kuhl, "A Formal Model of Distributed Decision-making and its Application to Distributed Load-Balancing," *6th IEEE International Conference on Distributed Computing Systems,* Cambridge, Massachusetts, May 1986, pp. 232-239.

[CaK87] T.L. Casavant, J.G. Kuhl, "Analysis of Three Dynamic Load-Balancing Strategies with Varying Global Information Requirements," *7th IEEE International Conference on Distributed Computing Systems,* West Berlin, West Germany, September 1987, pp. 185-192.

[CaK88] T. L. Casavant, J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," *IEEE Transactions on Software Engineering,* Vol. SE-14, No. 2, February, 1988, pp. 141-154.

[Cia87] G. Ciardo, "Toward a Definition of Modeling Power for Stochastic Petri Net Models," International Workshop on Petri Nets and Performance Models, Madison, Wisconsin, August 1987, pp. 54-62.

[Ens78] P.H. Jr. Enslow, "What is a "Distributed" Data Processing System," *Computer,* Vol. 11, No. 1, Jan. 1978, pp. 13-21.

[FrH85] U. Frisch, B. Hasslacher, Y. Pomeau, "A Lattice Gas Automaton for the Navier-Stokes Equation," *Preprint LA-UR-85-3503,* Los Alamos, 1985.

[HaD76] J. Hardy, O. DePazzis, Y. Pomeau, "Molecular Dynamics of a Classical Lattice Gas: Transport Properties and Time Correlation Functions," *Phys. Rev.,* A13(1949), 1976.

[Ho80] Y. Ho, "Team Decision Theory and Information Structures," *Proceedings of the IEEE,* Vol. 68, No. 6, Jun. 1980, pp. 644-654.

[HoK84] S.H. Hosseini, J.G. Kuhl, S.M. Reddy, "A Diagnosis Algorithm for Distributed Computing Systems with Dynamic Failure and Repair," *IEEE Transactions on Computers,* Vol. C-33, No. 3, Mar. 1984, pp. 223-233.

[Lar79] R.E. Larson, *Tutorial: Distributed Control,* IEEE Press, New York, 1979.

[Mil80] R. Milner, "A Calculus of Communicating Systems," LNCS 92, Springer-Verlag, 1980

[Mur77] T. Murata, "State Equation, Controllability, and Maximal Matchings of Petri Nets," IEEE Transactions on Automatic Control, June 1977, pp. 412-415.

[Pet77] J.L. Peterson, "Petri Nets," *Computing Surveys,* Vol. 9, Sep. 1977, pp. 223-252.

[Pet81] J. L. Peterson, "Petri Net Theory and the Modeling of Systems," 1981, pp. 79-110.

[Rei83] W. Reisig, "Petri Nets, An Introduction," 1983, pp. 61-109.

[SaW85] J. B. Salem, S. Wolfram, "Thermodynamics and Hyrodynamics with Cellular Automata," *Thinking Machines Corporation Technical Report Series,* Nov. 1985.

[Sta80] J.A. Stankovic, *A Comprehensive Framework for Evaluating Decentralized Control,* Proceedings 1980 International Conference on Parallel Processing, Aug. 1980, pp. 181-187.

[Sta85] J.A. Stankovic, "An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling," *IEEE Transactions on Computers,* Vol. C-34, No. 2, Feb. 1985, pp. 117-130.

[TeS81a] R.R. Tenney, N.R. Jr. Sandell, "Structures for Distributed Decisionmaking," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-11, No. 8, Aug. 1981, pp. 517-526.

[TeS81b] R.R. Tenney, N.R. Jr. Sandell, "Strategies for Distributed Decisionmaking," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-11, No. 8, Aug. 1981, pp. 527-537.

[ThM86] Thinking Machines Corporation, "Introduction to Data Level Parallelism, Chapter 3: Fluid Dynamics," *Technical Report Number 8.14,* April 1986, pp. 15-24.

APPENDIX
- CFA Semantics of ml1 -

NOTE: This section contains a section of the paper identified as [CaK86] in order to aid the reader in understanding the example of section 6.

This example, which will be called *ml1*, (the distributed environment is an *m*-node *l*oop), behaves as follows.

ml1: Each node examines the load of itself, and its two neighbors. One unit of load will be "given" to the neighbor with the least load if that load is less than its own load. Otherwise, no load is "given". Ties are arbitrarily broken in one direction.

The topology of this system is defined as follows:

$$\Phi = (V, A)$$

where

$$V = \{ v_0, v_1, \cdots, v_{m-1} \}; m = |V|$$

$$A = \{ (v_i, v_j) \mid (j = (i+1) \bmod m)\ V \quad (i = (j+1) \bmod m) \}$$

hence, the neighbor relation is defined as:

$$N(i) = \{ v_{(i-1) \bmod m}, v_{(i+1) \bmod m} \}$$

We further refine the decision sets to be:

$$D_l(i) = D_r(i) = \{ 0, 1 \}$$

Finally, the transition function is defined as follows. There are six cases of transitions to consider ($n = 1$). Even though we are considering a one-phase policy in this example, we will maintain the practice of using n to denote the number of phases in order to allow a simple extension of this example to a multiple phase example in the next section.

1) locally : $p_0 \rightarrow p_n$
2) locally : $p_n \rightarrow p_{n+1}$
3) locally : $p_{n+1} \rightarrow p_0$
4) at a neighbor : $p_0 \rightarrow p_n$
5) at a neighbor : $p_n \rightarrow p_{n+1}$
6) at a neighbor : $p_{n+1} \rightarrow p_0$

$$\delta_i(l^{int},\ l^{ext},\ p,\ d_{i,n_1},\ d_{i,n_2},\ p^r_{n_1},\ p^r_{n_2},\ p^{rin}_{n_1},\ p^{rin}_{n_2},\ d_{n_1,i},\ d_{n_2,i},\ l^{rext}_{n_1},\ l^{rext}_{n_2})$$

(1) =

$$(l^{int},\ l^{int},\ p_n,\ d_{i,n_1},\ d_{i,n_2},\ p^r_{n_1},\ p^r_{n_2})$$
$$\text{if } (p{=}p_0)\ \&\ (p^r_{n_1}{=}p^{rin}_{n_1})\ \&\ (p^r_{n_2}{=}p^{rin}_{n_2})$$

where:

$$d_{i,n_1} = \begin{cases} 1 & \text{if } ((l^{int} > l^{rext}_{n_1})\ \&\ (l^{rext}_{n_1} \leqq l^{rext}_{n_2})) \\ 0 & \text{otherwise} \end{cases}$$

$$d_{i,n_2} = \begin{cases} 1 & \text{if } ((l^{int} > l^{rext}_{n_2})\ \&\ (l^{rext}_{n_2} < l^{rext}_{n_1})) \\ 0 & \text{otherwise} \end{cases}$$

(2) =

$$(l^{int} - d_{i,n_1} - d_{i,n_2},\ l^{int} - d_{i,n_1} - d_{i,n_2},\ p_{n+1},\ d_{i,n_1},\ d_{i,n_2},\ p^r_{n_1},\ p^r_{n_2})$$
$$\text{if } (p = p_n)\ \&\ (p^r_{n_1} = p^{rin}_{n_1})\ \&\ (p^r_{n_2} = p^{rin}_{n_2})$$

(3) =

$$(l^{int},\ l^{int},\ p_0,\ d_{i,n_1},\ d_{i,n_2},\ p^r_{n_1},\ p^r_{n_2})$$
$$\text{if } (p = p_{n+1})\ \&\ (p^r_{n_1} = p^{rin}_{n_1})\ \&\ (p^r_{n_2} = p^{rin}_{n_2})$$

(4) = $(l^{int},\ l^{ext},\ p,\ d_{i,n_1},\ d_{i,n_2},\ p^{r1}_{n_1},\ p^{r2}_{n_2})$

$$\text{if } ((p^r_{n_1} \neq p^{rin}_{n_1})\ \&\ (p^r_{n_1} = p_0))\ V$$
$$((p^r_{n_2} \neq p^{rin}_{n_2})\ \&\ (p^r_{n_2} = p_0))$$

where:

$$ri = \begin{cases} rin & \text{if } (p^r_{n_i} \neq p^{rin}_{n_i})\ \&\ (p^r_{n_i} = p_0) \\ r & \text{otherwise} \end{cases}$$

for:

$$i \in \{1, 2\}$$

(5) $= (l^{int} + \sum_{j \in C} d_{j,i},\ l^{ext},\ p,\ d_{i,n_1},\ d_{i,n_2},\ p^{r1}_{n_1},\ p^{r2}_{n_2})$

if $((p^r_{n_1} \neq p^{rin}_{n_1})\ \&\ (p^r_{n_1} = p_n))\ V$

$((p^r_{n_2} \neq p^{rin}_{n_2})\ \&\ (p^r_{n_2} = p_n))$

where:

$$C = \{\ j \mid p^r_j \neq p^{rin}_j\ \&\ p^r_j = p_n\ \&\ j \in \{n_1\ ,\ n_2\}\ \}$$

and

$$ri = \begin{cases} rin & \text{if } (p^r_{n_i} \neq p^{rin}_{n_i})\ \&\ (p^r_{n_i} = p_n) \\ r & \text{otherwise} \end{cases}$$

for:

$i \in \{1, 2\}$

(6) $=$

$(l^{int},\ l^{ext},\ p,\ d_{i,n_1},\ d_{i,n_2},\ p^{r1}_{n_1},\ p^{r2}_{n_2})$

if $((p^r_{n_1} \neq p^{rin}_{n_1})\ \&\ (p^r_{n_1} = p_{n+1}))\ V$

$((p^r_{n_2} \neq p^{rin}_{n_2})\ \&\ (p^r_{n_2} = p_{n+1}))$

where:

$$ri = \begin{cases} rin & \text{if } (p^r_{n_i} \neq p^{rin}_{n_i})\ \&\ (p^r_{n_i} = p_{n+1}) \\ r & \textit{otherwise} \end{cases}$$

for:

$i \in \{1, 2\}$

During transition (1), decisions are made and transmitted to neighbors. The local record of each neighbor's state does not change, but the local phase does change to p_n. In (2), the local phase is updated to p_{n+1} and l_{int} and l_{ext} are both updated to reflect the local decisions reached during the previous phase change. The purpose of (3) is to transmit the new value of l_{int} to each neighbor. The value which was sent during (2) only reflected the effect of locally made decisions, and not the decisions of any neighbors which underwent transitions of type (2). This is a very subtle point since the full effect of the decisions made locally and at neighbors is not felt upon l_{int} until after transition (5) in which each neighbor's decisions is used to calculate the next value of internal load.

Transitions (4) and (6) simply change the internally recorded values of a neighbor's current phase.