THE DISTRIBUTED SYSTEM

POOL

L. Gerlach, K.T. Malowaniec,
H. Scheidig, R. Spurk
Fachbereich Informatik
Universität des Saarlandes
D–6600 Saarbrücken 11

Abstract

The development of a distributed system POOL based on a network which efficiently connects many processing elements is the subject of research supported by the DFG, SFB 124"VLSI-Entwurfsmethoden und Parallelität" (Teilprojekt D3). This contribution attempts to give an overview of the work done so far within this project.

1. Introduction and Motivation

We wish to build a distributed system which consists of (many) units U_i connected by a network T:

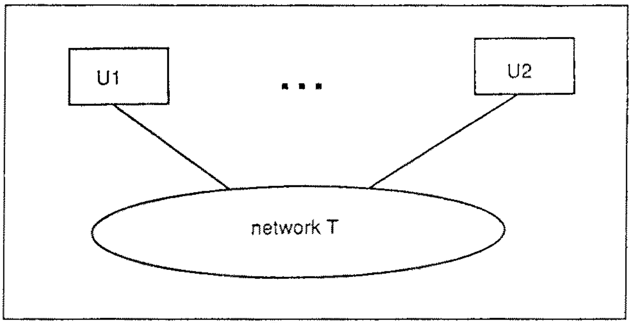


figure 1

At present we can discern essentially two approaches to the construction of such a distributed system:

A. THE "DIRECT", LANGUAGE-ORIENTED METHOD

This method tries to use and extend an existing language L in order

- to allow the formulation of subprograms (subtasks) P_i,
- and to perform the mapping of P_i onto the components U_i of the system.

Frequently a host (which can be considered as a special component U_i) is used to produce the input to the system (data as well as program modules P_i) and to "download" the single units U_i.

As examples the following systems can be named: the system FPS [FPS 86] and the parallel computer designed in the suprenum project [SUPRENUM] (with L = OCCAM, FORTRAN and Concurrent Modula-2, respectively), both intended for number-crunching purposes, the connection machine [Hillis 86] and the LISP-machine of [Guzman 83] for symbol manipulation (typical in the A.I.-area) with L = LISP.

This method is particularly well suited for applications which are elementary in the following sense:

- there is a 1-1-correspondence between logical (user-defined) objects P_i and physical objects U_i,
- every (logical) operation on P_i can easily be expressed by a simple instruction (program) on U_i, and
- the requirements which have to be imposed upon the communication system can be deduced from simple neighbourhood relationships between objects P_i.

Typical examples (pixel operations and transistor simulation) are given in [Hillis 86].

B. THE DISTRIBUTED OPERATING SYSTEMS APPROACH

This approach tries to design an operating system which itself is the object of distribution. The operating system attempts to equip the user with higher level objects o

- as entities of distribution at the user level,
- which can be mapped in a transparent and automatic way onto the available physical units,

thus providing an object-oriented programming environment appropriate for distributed systems.

The process of building higher-level objects o can be considered as a generalization of the usual method of defining abstract objects:
An object o is an active element which

- sends/accepts requests (a request is coded as a remote procedure call addressing an entry e of o which serves as an access point to a certain subtask of o), and
- which may delegate the execution of some subtasks to other objects (slaves) allocated by the system on demand.

Our distributed system POOL follows this operating system approach (see [Scheidig 83], [Scheidig 85]); the system ARGUS [Liskov 84] can be named as a further example.

2. The Structure of the Distributed System POOL

The structure of the system POOL can be illustrated by the following figure:

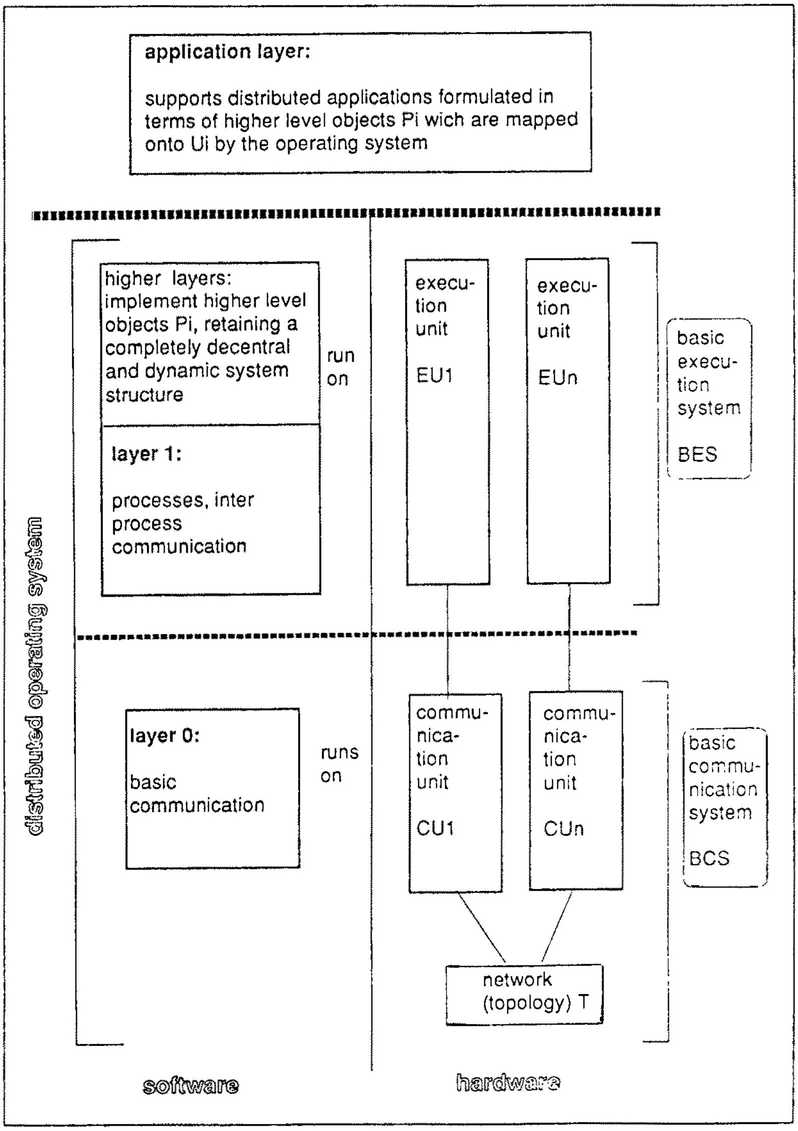

application layer:
supports distributed applications formulated in terms of higher level objects Pi wich are mapped onto Ui by the operating system
higher layers: implement higher level objects Pi, retaining a completely decentral and dynamic system structure
layer 1:
processes, inter process communication
run on
execution unit EU1
execution unit EUn
basic execution system BES
distributed operating system
layer 0:
basic communication
runs on
communication unit CU1
communication unit CUn
basic communication system BCS
network (topology) T
software
hardware

figure 2

The following contribution is divided into three parts:

- In sections 3, 4, we describe the Basic Communication System running on the communication units CU.
- In sections 5, 6, 7, we introduce two versions of POOL already implemented. We discuss the higher system layers running on the execution units EU and the basic mechanisms necessary to support distributed applications. We briefly address the question of how a "remote user" can find access to POOL via the campus-wide network CANTUS (viewing POOL as one of many nodes connected to CANTUS).
- In section 8, we comprise some ideas about the user interface of a distributed systems.

3. The Model of the Basic Communication System and its Implementation

The efficiency of the Basic Communication System (BCS) is of prime importance because the spectrum of applications supported by the distributed system depends on BCS performance:

the better the ratio of

execution-time (EU-time) to communication-time (CU-time),

the higher the granularity of the applications that can be chosen.

This goal of high performance of the BCS leads to a number of requirements concerning

- the units U_i,
- the topology T, and
- the method of exchanging data among units U_i.

3.1 Requirements, Properties

A. SEPARATION OF COMMUNICATION AND EXECUTION IN THE U:

A unit U of POOL consists of two processing elements U = (CU, EU) (compare figure 2). The communication unit CU implements the basic communication since the execution unit EU realizes the higher levels of the operating system.

The CU-units are homogeneous in the system. Thus we can choose specifically designed hardware which guarantees the required efficiency of the basic communication.

The EU-units need not necessarily be homogeneous; on the contrary, we want to be able to integrate a new and attractive processing element as EU into the systemat any time . The EU's can fully concentrate on their execution tasks; they are not required to contribute to the transport of data.

B. CHOICE OF NETWORK TOPOLOGY T

As topology T of the network which interconnects the units U of our distributed system we choose the structure referred to as "cube connected cycles" (CCC), see [Preparata 81]. CCC can be viewed as an optimization of the hypercube topology—retaining most of its advantages and adding the following important features:

b1. Constant local complexity: A CCC-network T of dimension d consists of $n = d \cdot 2^d$ elements. However, the number of edges emanating from every node is constant (3) and does not depend on d. This property makes the network T easily extensible and reduces substantially the costs of assembling T.

b2. The choice of one specific topology does not really restrict the generality because we know that an arbitrary network can be simulated by CCC with not too much overhead (the overhead is a logarithmic function of the number of elements).

b3. There exists a simple (static) routing method for CCC; moreover, routing can be improved by employing a dynamic (random routing) version which tries to distribute the communication load over BCS.

b4. There exists a simple, automatic layout algorithm for CCC which is applied in order to produce a back panel board for CCC. This board realizes the complete physical interconnection structure thus eliminating the necessity of wiring. This back panel can be viewed as a generalization of a conventional bus system (see [Krass]).

C. "NAMING", TWO-LEVEL ROUTING

We use arbitrary identifiers to name and identify POOL-objects (e.g. POOL-processes). A two-level algorithm has to be applied to allow an object U_0 to communicate with an object U_n:

Step 1: U_0 emanates first a multicast message: M(area, "object with identification $id(U_n)$ please send answer to U_0"). Every unit U in the area covered by this multicast checks whether its own identification $id(U)$ (attached to it when created) is equal to $id(U_n)$. If such a U exists then it sends its network address $adr(U)$ back to U_0.

Step 2: If step 1 produces a result $adr(U)$ then U_0 and U_n can henceforth communicate via point-to-point communication using the network addresses $adr(U_0)$ and $adr(U_n)$ to define a path from U_0 to U_n (and vice versa).

Notice that step 1 has to be repeated if the point-to-point communication fails (e.g. due to a hardware error in U_0, U_n or in one of the intermediate nodes on the path from U_0 to U_n).

Notice furthermore that the logical properties of multicasts and suitable assumptions concerning names (in particular concerning the uniqueness of names) have to be taken into account in the design of the higher levels of the distributed operating system.

D. TASK OF THE BASIC COMMUNICATION SYSTEM BCS

The task of the basic communication system BCS can now be summarized as follows: BCS has to perform

- point-to-point communication, and
- multicast operations

efficiently.

3.2 Working Principle of the Basic Communication System

Figure 3 shows how a single communication unit U_i works:

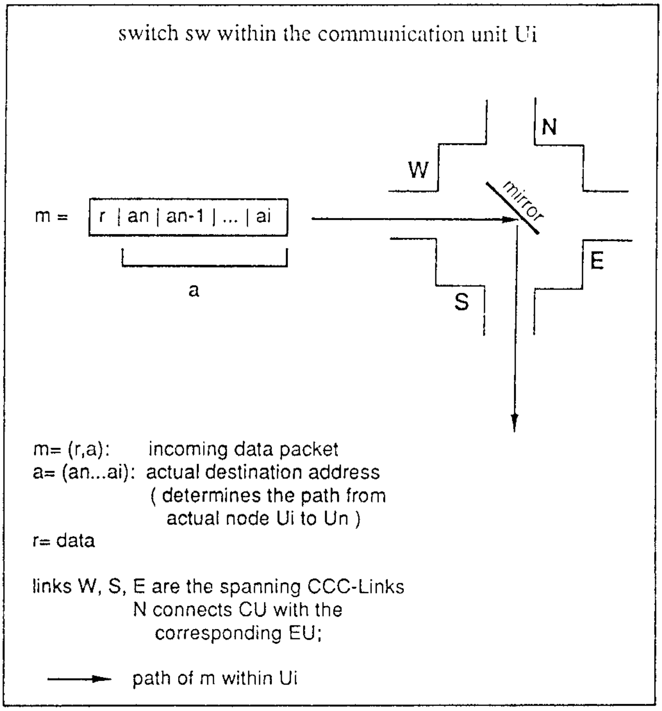

switch sw within the communication unit Ui
m = r | an | an-1 | ... | ai
a
W
N
E
S
mirror
m= (r,a): incoming data packet
a= (an...ai): actual destination address (determines the path from actual node Ui to Un)
r= data
links W, S, E are the spanning CCC-Links
N connects CU with the corresponding EU;
path of m within Ui

figure 3

The leading address information ai (2 bits) of the header of the incoming data packet m turns the "mirror" into the position appropriate to "reflect" the rest of m into the desired direction (that means it puts $m' = (r, a_n, a_{n-1}, ..., a_{i-1})$ without further inspection onto the addressed link $L \in \{W, N, E, S\}$).

We call this working principle of CUs the "turning mirror" method.

The whole process starts in U_0 with the (complete) address $a = adr(U_n)$, each node of the path from U_0 to U_n operating as just explained.

3.3 Handling of Conflicts

The turning mirror method fails in the following obvious case:

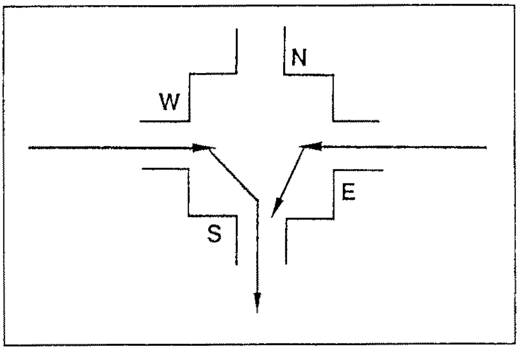


figure 4

Here, a conflict occurs because two data packets m1, m2 arrive at the same time which both have to be put onto the same output link, say S.

As a solution of this conflict we do not want to apply usual store-and-forward techniques because this would severely decrease the efficiency of the BCS (and increase the costs of the CU-units).

Instead we argue as follows:
For several reasons (in particular to handle hardware failures) we have to provide an end-to-end protocol between communicating instances U_0 and U_n. If we succeed in reducing the probability of the above-mentioned conflicts so that it does not exceed the probability of other failures, then we can also hold the end-partners responsible for the handling of these conflicts.

Thus the question arises of how we can decrease (minimize) the number of conflicts when applying the turning mirror method.

This is easy - we simply increase the capacity of our links $\{W, N, E, S\}$. A link $L \in \{W, N, E, S\}$ is a logical entity which need not necessarily be realized by one single physical channel. We will in fact use p physical channels C to implement every link L and we will choose one of them which is free whenever we have to put a data packet on this link.

Thus the "turning mirror principle with conflict solution" works as follows:

- If packet m arrives at U_i then the leading address information (2 Bits) determines the output link L.

- A free channel C (if any) of the p channels realizing L is chosen.
- The packet m is "reflected" onto C.
- If no free channel C is available then the sender U_0 is notified immediately (i.e. by hardware signals back the chain of intermediate nodes $U_i, U_{i-1}, ..., U_0$); U_0 can take the appropriate measures.

The packet m dynamically constructs a path from U_0, ... U_i, ... to U_n by occupying a free channel at every intermediate node U_i. We therefore call this method "dynamic circuit switching" (in analogy to the usual static circuit switching scheme used in telephone systems, for example).

Note that the minimal packet length of m is chosen in such a path that the longest noncyclic path PATH_max in the CCC-network is fully covered by m.

The following now holds:

a. The dynamic circuit switching method can be implemented efficiently (mostly in silicon!), see 3.4 below.

b. The correct functioning of the method can be ensured (see 4 below) and the appropriate number p of channels can be determined (as a function of the CCC-dimension and other parameters).

c. The time t needed to transport a packet m from U_0 to U_n (if no conflicts occur) is given by

   t = number_of_hops · SW-time.

   SW-time denotes the time required by the turning mirror algorithm within a single CU.

   End-to-end actions are easy because the absence of an acknowledge over a time interval t > t_error with

   t_error = 2 · t_max (t_max = time for PATH_max, see above)

   signals a failure.

d. Multicast can also be implemented efficiently on the basis of the point-to-point communication described so far. There are several possible ways which differ by the amount of software support by which the point-to-point communication has to be enhanced in order to realize multicast.

3.4. Hardware and software architecture of CUs

The resulting architecture of a CU-unit can now be summarized as follows:

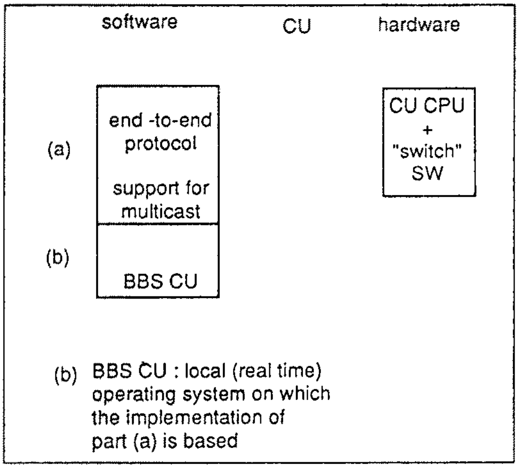

software
CU
hardware
(a)
end -to-end protocol
support for multicast
(b)
BBS CU
CU CPU + "switch" SW
(b) BBS CU : local (real time) operating system on which the implementation of part (a) is based

figure 5

The structure of the component SW (switch) is described by the following figure.

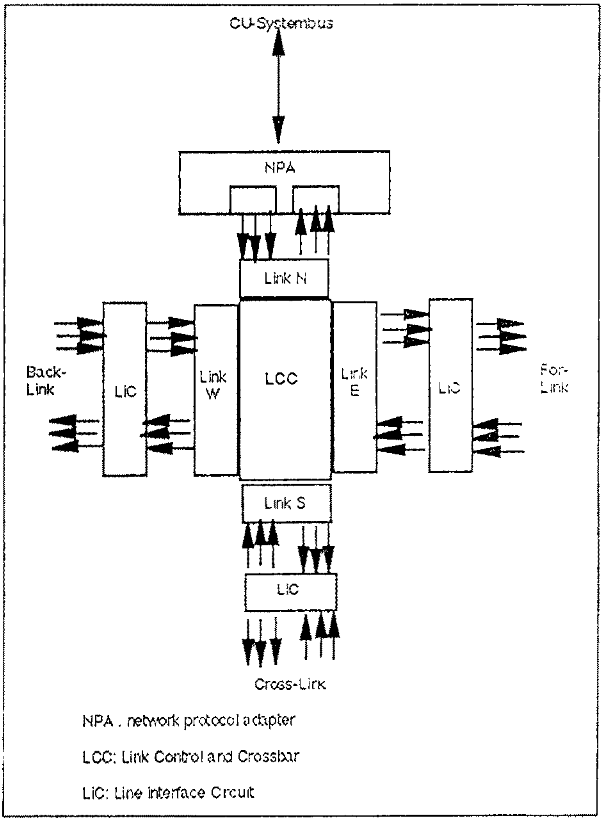

CU-Systembus
NPA
Link N
Back-Link
LiC
Link W
LCC
Link E
LiC
For-Link
Link S
LiC
Cross-Link
NPA . network protocol adapter
LCC: Link Control and Crossbar
LiC: Line interface Circuit

figure 6

The "heart" LCC of the switch SW is realized by a two dimensional daisy chain (see [Schneider]). The design of the LCC as a VLSI-Chip has just begun.

3.5. Performance

Applying the VLSI-design techniques available on the market today for the manufacturing of the LCC-chip would lead to the following performance figures:

The overall (gross) transport capacity e of one CU is given by

$$e = n \cdot p \cdot s$$

with n = number of CU-links (4, including the connection CU - EU),
p = number of channels per link (8),
s = capacity of one channel (20 [Mbit/s])

as 640 [Mbit/s].

We have to notice that the actually available "net"-capacity is at most 10% of this maximal value (guaranteeing a tolerable small rate of conflicts). There are, however, several ways of further increasing the capacity of the system (e.g. by "cascading" LCC-Chips).

4. The Verification of the Basic Communication System

The architecture of the BCS was described in section 3. We will call a network with a switching method based on the turning mirror principle a Dynamic Circuit Switching Network. This section describes an analytical model and the design of such networks. We will first describe a model and then show some suitable topologies; afterwards, we will verify our communication system using the analytical model for the topologies considered.

4.1. The Analytical Model

In this section we will describe one analytical model of a Dynamic Circuit Switching Network (DCSN). We will begin with a definition of the model parameters and then focus on the model itself. Finally, we will show how to use the model for network design.

4.1.1. The Model Parameters

The input parameters of the network model are: network traffic, network topology and a routing function.

NETWORK TRAFFIC

We have a network of N nodes, numbered $[0..N-1]$. Each node can send messages to every other node.

The messages from node k to node l are sent with the Poisson distribution at arrival rate $\lambda_1[k,l], (k,l \in [0..N-1])$.

The length of messages in the network is exponentially distributed with mean $E(L)$.

The speed of transmission of every communication channel in the network is S.

The service rate can be computed as $\mu = S/E(L)$.

The traffic in a network with N nodes can be described by a matrix of arrival rates $\lambda_1[k,l], (k,l \in [0..N-1])$ and a service rate of $\mu = S/E(L)$.

NETWORK TOPOLOGY

The Dynamic Circuit Switching Network consists of a set of switches connected by simplex communication channels. Figure 7 shows one example of a network topology. It consists of four nodes numbered 0 .. 3. The arrows represent unidirectional communication channels. The rectangles numbered 0, 1, 2, and 3 represent the switches in the appropriate nodes. The rectangle with number -1 represents the rest of the network apart from the switches. The communication channels are grouped into unidirectional edges. For example, it can be seen that edge (1,2) consists of two communication channels, edge (2,1) also consists of two unidirectional channels and edge (1,-1) consists of one communication channel.

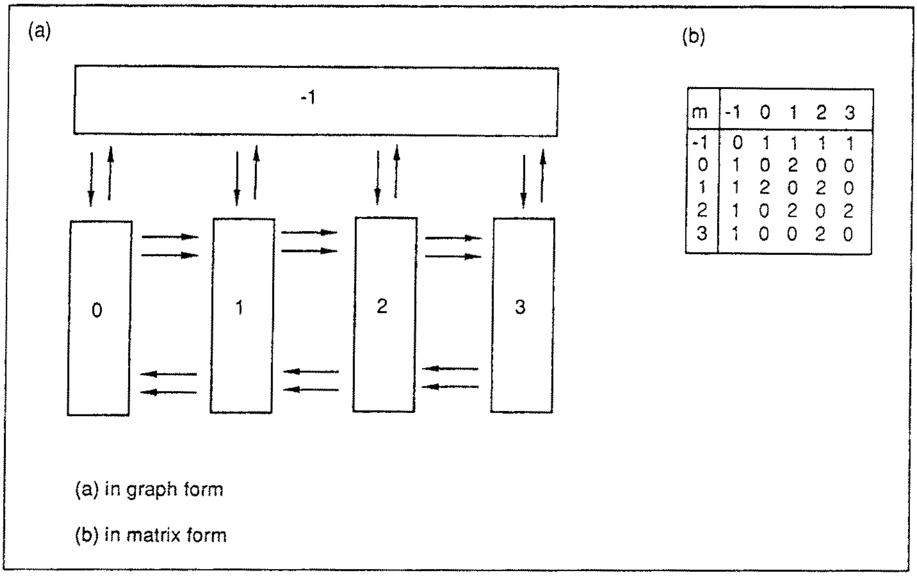
The topology of the network can be described by a matrix $m[i,j], (i,j \in [-1, 0..N-1])$.

For any $i,j \in [0..N-1], m[i,j] = k, (k \in [0,1..])$ means that there are k simplex communication channels from node i to node j.

For any $j \in [0..N-1], m[-1,j] = k, (k \in [0,1..])$ means that there are k simplex communication channels from the CU processor to the switch in node j.

For any $i \in [0..N-1], m[i,-1] = k, (k \in [0,1..])$ means that there are k simplex communication channels from the switch to the CU processor in node i.

If $m[i,j] = 0$, then there is no edge from i to j. If $m[i,j] = k$ and $k > 0$, then an edge exists from i to j and it consists of k simplex communication channels.



m	-1	0	1	2	3
-1	0	1	1	1	1
0	1	0	2	0	0
1	1	2	0	2	0
2	1	0	2	0	2
3	1	0	0	2	0

(a) in graph form

(b) in matrix form

figure 7

ROUTING IN THE NETWORK

Routing in the DCSN is static, which means that all messages from node k to node l always take the same path.

Routing can be defined for a particular topology as an algorithm for a routing function. The routing function NextNode(Current,Dest) returns the address of the next node as a function of the current node's address and the destination node's address.

Routes in static routing can only be changed statically by a topology update or a change in the routing algorithm. They cannot be changed dynamically. Examples of dynamic routing are adaptive routing with routes depending on network state, or random routing with routes depending on random number generation.

4.1.2. Model Description

The parameters of the network model are: number of nodes N; arrival rates $\lambda_1[k,l]$, $(k,l \in [0..N-1])$; mean message length $E(L)$; speed of transmission S in a communication channel; service rate μ; network topology $m[i,j]$, $(i,j \in [-1,0..N-1])$; and the routing algorithm.

The following questions arise: what is the probability $P[k,l]$ that the message originating in node k with destination in node l will be lost; and what is mean $E(P)$, variance $Var(P)$, and standard deviation σ_P of the probability of message loss in the network.

Furthermore, we have the following design problem: what value should be chosen for the number of communication channels $m[i,j]$ on the edges of our topology in order to obtain a network with desired mean message loss probability $E(P)$?

We intend to construct a model of a network to solve this problem. First, we will decompose the network into elementary queueing systems, then use rules of queueing theory for these systems. Afterwards, the single elements will be composed into a network of queueing systems.

DECOMPOSITION

We are interested in finding the probability $B[i,j], (i,j \in [-1, 0..N-1])$ of a message being lost while attempting to go through edge (i,j). For this purpose, we must compute the traffic going through this edge and find a suitable queueing system model for it. We must decompose our network into these elementary queueing systems.

In order to solve this problem, we begin with the tentative assumption that we always have a sufficient number of communication channels on each edge to transfer all incoming messages: each edge is said to have an infinite number of channels.

The arrival distribution on edge (i,j) is the sum of many independent Poisson distributions. It is also itself a Poisson distribution. We will compute an arrival rate $\lambda[i,j]$ for this distribution.

The arrival rate of messages entering the network in node j, $(j \in [0..N-1])$ is:

$$\lambda[-1,j] \sum_{l=0}^{N-1} \lambda_1[j,l]$$

The arrival rate of messages leaving the network in node i, $(i \in [0..N-1])$ is:

$$\lambda[i,-1] \sum_{k=0}^{N-1} \lambda_1[k,i]$$

The arrival rate of messages going through the edge (i,j), $(i,j \in [0..N-1])$ is:

$$\lambda[i,j] = \begin{cases} \sum\limits_{\substack{k,l \in [0..N-1]: \\ \textit{route from } k \textit{ to } l \\ \textit{goes through } (i,j)}} \lambda_1[k,l] & \text{if } m[i,j] \geq 0 \\ & \\ 0 & \text{if } m[i,j] = 0 \end{cases}$$

The matrix $\lambda[i,j], (i,j \in [-1, 0..N-1])$ describes the arrival processes on each edge of the network.

A service distribution on edge (i,j) is an exponential distribution with a mean equal to E(L)/S and service rate μ = S/E(L), where E(L) is the mean message length and S is a speed of transmission in a communication channel.

A single edge can be described by the following queueing system: We have the Poisson arrival distribution and exponential service time distribution. There are an infinite number of communication channels available on edge (i,j). Each newly arriving message is always given its private communication channel. This is the infinite-server system M/M/∞.

Yet in reality there are only $m[i,j]$ communication channels available on edge (i,j). Each newly arriving message is given its private communication channel. However, if the message arrives when all communication channels are occupied, the message is lost. It is an $m[i,j]$-server loss system.

The output of an M/M/∞ system still has Poisson arrival distribution. It is important in networks with many such systems, where the output from one system can be an input to another. In this case, the network is a combination of many M/M/∞ systems and an analytical model of this network can easily be constructed since the arrival distribution of single edges can be computed as shown above.

Yet the output of $M/M/m[i,j]/m[i,j]$ no longer has a Poisson arrival distribution. In a network with many such systems, the output of one system may be an input to another. In this case, the network is a combination of many $G/M/m[i,j]/m[i,j]$ systems, with a general arrival distribution, in spite of the Poisson arrival distribution at the network entrance. It is difficult to construct an analytical model for such a network because arrival distribution cannot be computed in such a way as shown above. Only a simulation model is possible in this case.

We are interested in designing a high-quality network with very low loss probability $B[i,j]$. Here, we notice that the $M/M/m[i,j]/m[i,j]$ system with very low loss probability $B[i,j]$ can approximate an M/M/∞ system.

$$M/M/m[i,j]/m[i,j] \xrightarrow{B[i,j]\to 0} M/M/\infty$$

A network of many M/M/∞ systems can be approximated analogously by a network of many $M/M/m[i,j]/m[i,j]$ systems with very low loss probabilities $B[i,j]$. Arrival distribution of output from these queueing systems can be approximated by a Poisson distribution. Thus we can compute arrival distribution as shown earlier and can easily construct an analytical model of the network.

SINGLE EDGE CALCULATIONS

We have shown that a network can be decomposed into its one-edge elements. Each edge can be modeled by an elementary queueing system $M/M/m[i,j]/m[i,j]$. We will use this queueing model to perform a calculation for a single edge.

We are looking for the probability $B[i,j], (i,j \in [-1,0..N-1])$ for a message loss on edge (i,j). This value can be computed with Erlang's loss formula [Kleinrock 75]. If we apply this formula to our model we will arrive at:

$$B[i,j] = \frac{\dfrac{(\lambda[i,j]/\mu)^{m[i,j]}}{m[i,j]!}}{\displaystyle\sum_{k=0}^{m[i,j]} \frac{(\lambda[i,j]/\mu)^k}{k!}}$$

The matrix $B[i,j]$, $(i,j \in [-1,0..N-1])$ describes the loss probability on all edges in the network.

FINDINGS

We have made computations for all single edges. We will now compose single queuing systems (representing edges) to model a complete network. Based upon the model, we will derive results for the complete network.

We want to find the probability $P[k,l]$ that a message from node k to node l will be lost.

$$P[k,l] = 1 - (1 - B[-1,k]) \cdot (1 - B[l,-1]) \cdot \prod_{\substack{i,j \in [0..N-1]: \\ \textit{route from k to l} \\ \textit{goes through (i,j)}}} (1 - B[i,j])$$

We can compute the mean probability that the message will be lost in the network:

$$E(P) = \frac{\displaystyle\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} (P[k,l] \cdot \lambda_1[k,l])}{\displaystyle\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} \lambda_1[k,l]}$$

The second moment of this distribution is:

$$E(P^2) = \frac{\displaystyle\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} (P[k,l]^2 \cdot \lambda_1[k,l])}{\displaystyle\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} \lambda_1[k,l]}$$

The variance of this distribution:

$$Var(P) = E(P^2) - (E(P))^2$$

The standard deviation:

$$\sigma_P = \sqrt{Var(P)}$$

The mean value and standard deviation of P describe the quality of the network.

4.1.3. Network Design

We have a network described by the following parameters: number of nodes N; arrival rates $\lambda_1[k,l], (k,l \in [0..N-1])$; mean message length E(L); speed of transmission S in a communication channel; service rate μ; network topology $m[i,j], (i,j \in [-1,0..N-1])$; and routing algorithm.

We can use the model constructed in section 4.1.2 for designing a network. We have a choice between two operation modes:

The first operation mode is a computation for an existing network: the given is a topology of the network by matrix $m[i,j]$, which describes how many communication channels exist on each edge.

We compute loss probability on each edge $B[i,j], (i,j \in [-1,0..N-1])$, then loss probability between end nodes $P[k,l], (k,l \in [0..N-1])$, then mean probability of loss $E(P)$ and standard deviation of loss σ_P.

The second operation mode is a network optimization: the given is a topology of the network by matrix $m[i,j]$, which describes how many communication channels exist on each edge. We intend to change a number of communication channels on existing edges (with $m[i,j] > 0$) to obtain an optimal network with mean message loss probability E(P) less than given parameter EPLimit.

BMin is initialized to 0.0 and BMax to EPLimit, respectively.

We compute BLimit as (BMin + BMax) / 2.0. Then we find a number of channels $m[i,j]$ on each existing edge which is large enough to receive loss probability on the edge $B[i,j]$ less than BLimit. Then we compute loss probability between end nodes $P[k,l]$, and afterwards the mean probability of loss E(P) and standard deviation of loss σ_P.

If the computed E(P) is less than required by EPLimit, we will set BMin to BLimit. Otherwise if the computed E(P) is greater or equal than required by EPLimit, we will set BMax to BLimit.

Afterwards the computation is repeated as described above. The iteration is stopped when the computed E(P) is less than EPLimit and the difference (Bmax - BMin) is less than an assumed parameter BEpsilon.

We can also mix the two operation modes: For example, we can make a computation for an existing network with a new traffic condition in the first operation mode. If we are not satisfied with the mean loss probability E(P), we can use the second operation mode to obtain required E(P). We can also manipulate the other parameters, e.g., we can change the speed of transmission S or the routing algorithm.

Thus, the model can be used to examine existing networks and to design networks of arbitrary quality.

4.2. Topology Considerations

This section shows two topologies which seem to be suitable for distributed systems. They are two hypercube topologies: Cube- Connected Cycles and Two Way Digit Exchange.

4.2.1. Cube-Connected Cycles Topology

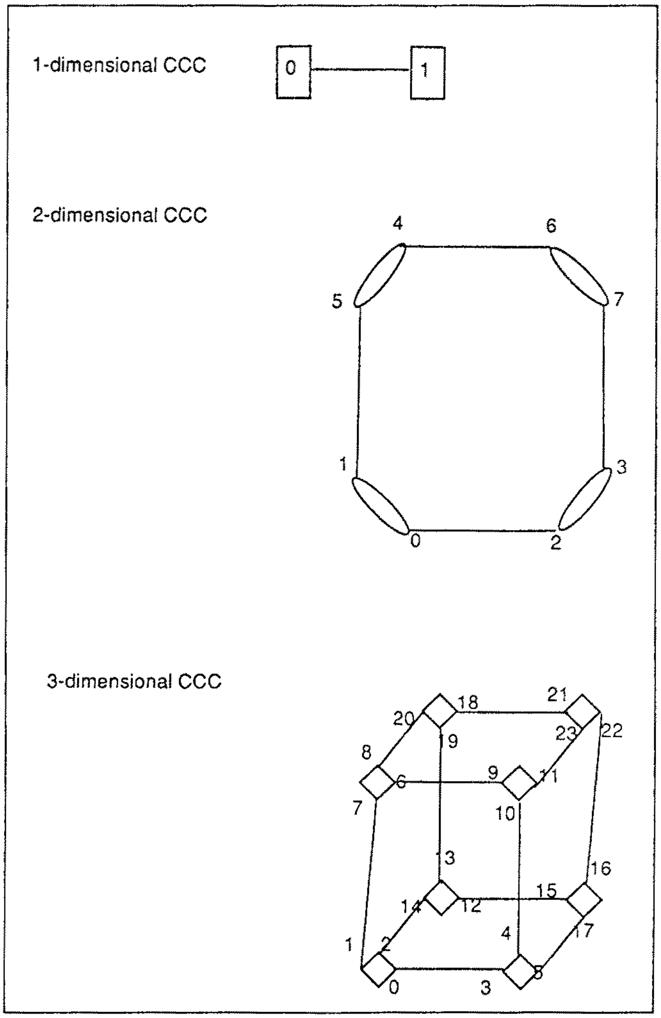


figure 8

Some examples of CCC topology are illustrated in figure 8. A CCC of dimension D consists of $D \cdot 2^D$ nodes. They are grouped into a cycle of D nodes, surrounding each of 2^D vertices of a binary hypercube of dimension D.

One node is connected to exactly three others: one bidirectional link crosses the hypercube in one of the D dimensions. A cross link consists of two unidirectional edges. Two bidirectional or unidirectional links connect nodes to neighbours on the same cycle. A cycle link consists of two edges in the bidirectional case (figure 9) and one edge in the unidirectional case (figure 10). One edge consists of one or more unidirectional communication channels.

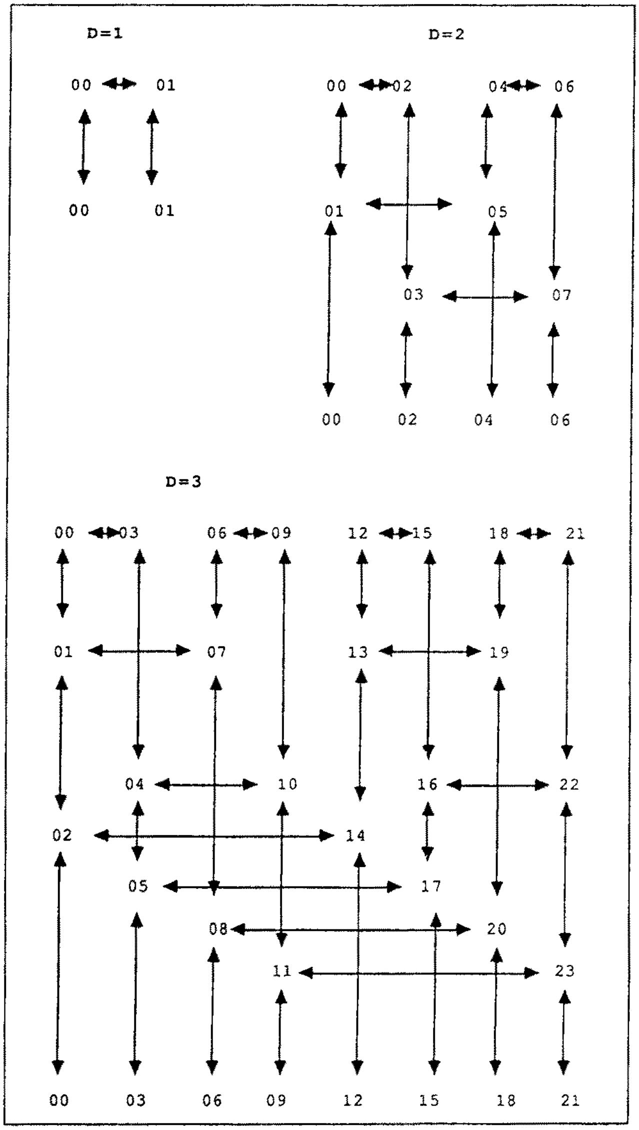


figure 9

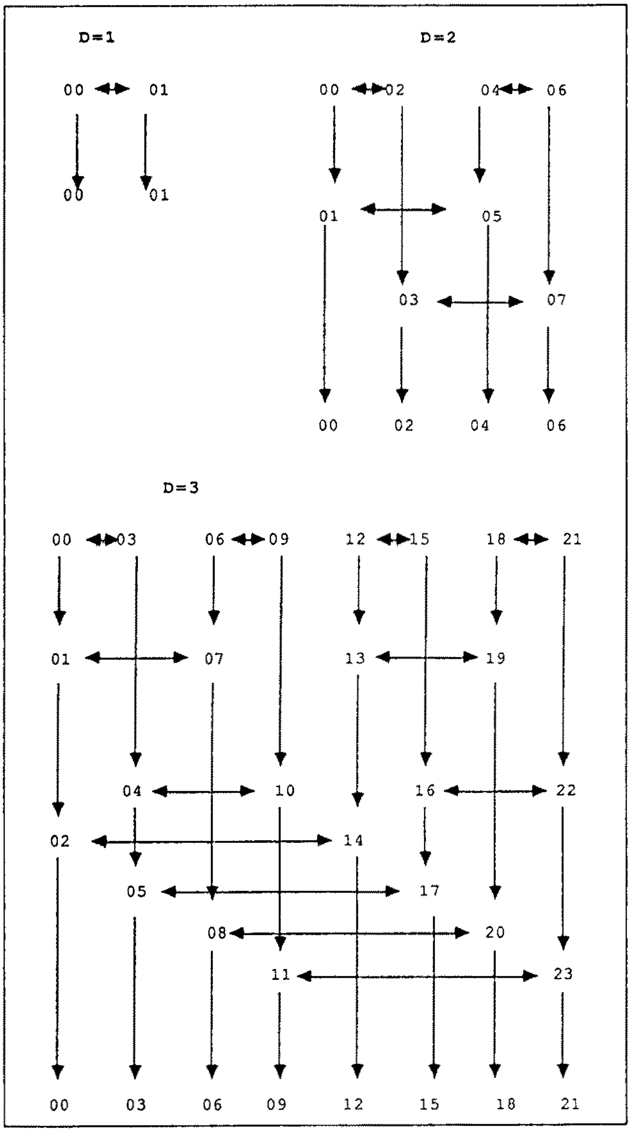


figure 10

An analysis of CCC networks is offered in [Preparata 81] and [Wittie 81]. We will show the results for DCSNs having this topology in section 4.3.

4.2.2. Two Way Digit Exchange Topology

2-WADE topology is related to the CCC topology with unidirectional cycle links. A D-dimensional 2-WADE also consists of $D \cdot 2^D$ nodes. They are also grouped into a cycle of D nodes around each of the 2^D vertices of a binary hypercube of dimension D.

Figure 12 presents some layout examples for the 2-WADE topology. They are similar to those for the CCC topology with unidirectional cycle links in figure 10. The only difference is that each bidirectional cross link is divided into two unidirectional cross links. They do not connect nodes with this same cycle position, but nodes with consecutive cycle positions.

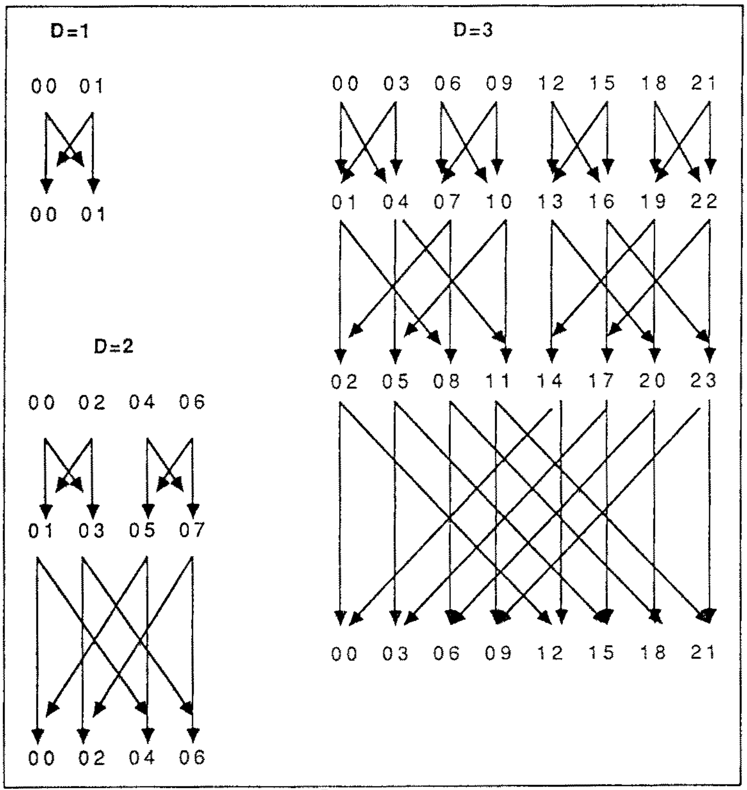


figure 11

Each node is connected to exactly four others: two unidirectional links cross the hypercube in one of the D dimensions. A cross link consists of one unidirectional edge. Two unidirectional links connect each node to neighbours on the same cycle. A cycle link consists of one unidirectional edge. One edge consists of one or more unidirectional communication channels.

An analysis of 2-WADE networks with random routing can be found in [Upfal 84]. We will show the findings for DCSNs having this topology in section 4.3.

4.3. System Verification

This section contains a verification of our communication system. We will apply our analytical model from section 4.1 to the topologies shown in section 4.2. We would like to answer the following questions: How many parallel communication channels do we need in particular links? What is the total number of communication channels connected to one switch? And, which topology is the best?

We assume the following parameters of maximal network traffic. The number of nodes in a topology is

$$N = D \cdot 2^D [nodes]$$

The messages from node k to node l are sent with a Poisson distribution at the arrival rate

$$\lambda_1[k,l] = 100/N[packet/s]$$

This means every node generates 100 messages per second, which are uniformly distributed to all nodes in a network. The length of a message is exponentially distributed with mean

$$E(L) = 8192[bit] = 1[KByte]$$

The speed of transmission for a communication channel is

$$S = 10^7[bit/s] = 10[Mbit/s]$$

The service rate is

$$\mu = S/E(L) = 1220.7[1/s]$$

We will compare three topologies from section 4.2: CCC with bidirectional cycle links, CCC with unidirectional cycle links and 2-WADE. Each topology will be examined with the following dimensions

$$D = 3..6$$

A static routing will be used for all topologies.

We will use the analytical model to design a network with an mean loss probability E(P) less than the following limit

$$EPLimit = 0.001$$

We are interested in the following results:

up - number of channels on unidirectional edge in up link (between switch and CU processor)
cross - number of channels on unidirectional edge in cross link
cycle - number of channels on unidirectional edge in cycle link
total - total number of communication channels per switch (outgoing plus ingoing)

The results are shown in figure 12.

CCC topology with bidirectional cycle links						
D	N	$\lambda 1$	up	cross	cycle	total
3	24	4.1666666	3	3	3,3	24
4	64	1.5625	3	4	4,4	30
5	160	0.625	3	4	4,4	30
6	384	0.2604166	3	4	5,5	34

CCC topology with unidirectional cycle links						
D	N	$\lambda 1$	up	cross	cycle	total
3	24	4.1666666	3	3	4	20
4	64	1.5625	3	4	5	24
5	160	0.625	3	4	5	24
6	384	0.2604166	3	4	6	26

2-WADE topology						
D	N	$\lambda 1$	up	cross	cycle	total
3	24	4.1666666	3	3	4	20
4	64	1.5625	3	4	4	22
5	160	0.625	3	4	5	24
6	384	0.2604166	3	4	5	24

figure 12

Figure 12 shows how many communication channels should be placed on one edge of up, cross and cycle link to obtain a network with the required mean loss probability. Compare figure 12 with figures 9, 10 and 11 to see the relation between edges and links. Unidirectional links consist of one edge and bidirectional links consist of two edges.

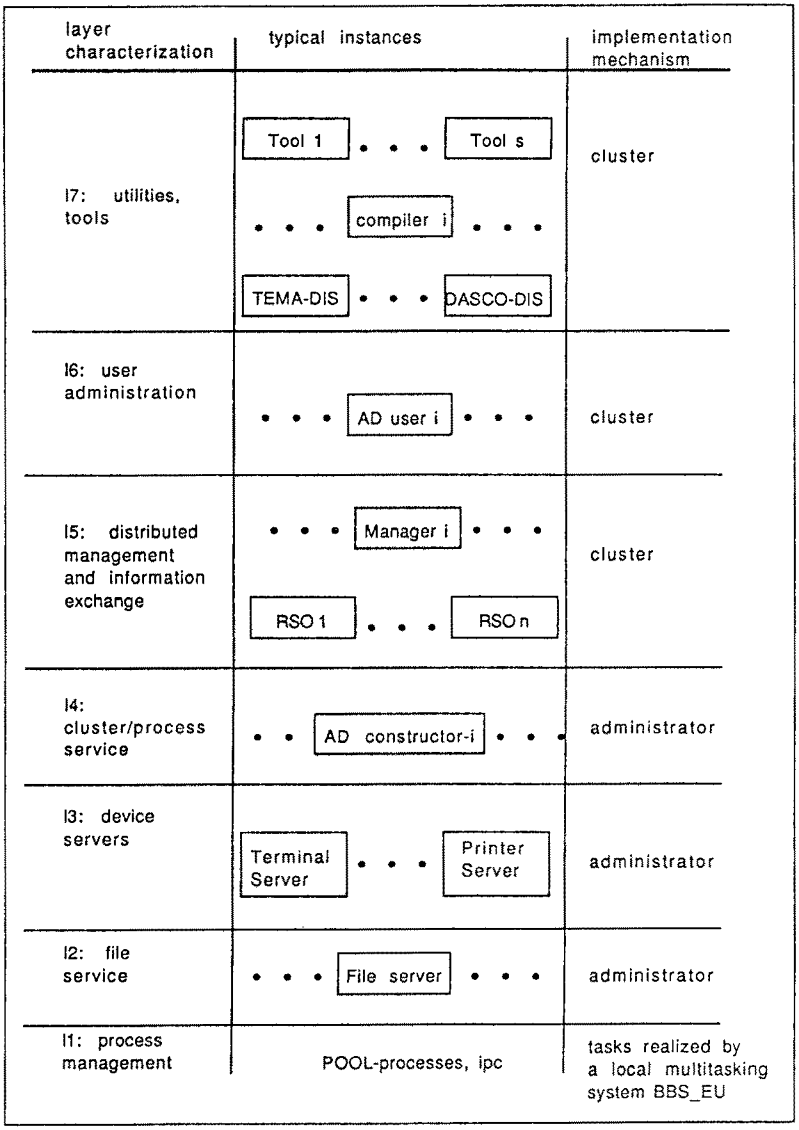
We have used our analytical model from section 4.1 with topologies shown in section 4.2. To verify our communication system, we must answer a number of questions, e.g.: How many parallel communication channels do we need in particular links? What is the total number of communication channels connected to one switch? Which topology is the best?

The quality of each network is almost identical because all have nearly the same E(P). In order to select an optimal topology, we will compare their costs, which are proportional to the number of channels: the better a topology, the less communication channels it has. Figure 12 indicates that the 2-WADE topology is better than CCC because of the smaller number of channels per switch.

We have shown with an analytical model that it is possible to construct Dynamic Circuit Switching Networks to meet our requirements. The computed number of communication channels per node is technologically acceptable. Further study of DCSN can be found in [Malowaniec].

5. The architecture of the basic execution system BES

The basic execution system BES of POOL has a layered structure as shown in the following figure:



layer characterization	typical instances	implementation mechanism
l7: utilities, tools	Tool 1 • • • Tool s • • • compiler i • • • TEMA-DIS • • • DASCO-DIS	cluster
l6: user administration	• • • AD user i • • •	cluster
l5: distributed management and information exchange	• • • Manager i • • • RSO 1 • • • RSO n	cluster
l4: cluster/process service	• • AD constructor-i • • •	administrator
l3: device servers	Terminal Server • • • Printer Server	administrator
l2: file service	• • • File server • • •	administrator
l1: process management	POOL-processes, ipc	tasks realized by a local multitasking system BBS_EU

figure 13

l1: implements POOL-processes, called administrators AD (see below) and cooperating sets of administrators, called clusters (see below).

12: general file service; management of data representing a user within the system (e.g. data for accounting, access control, etc.). Typical instance: AD_file.

13: management of other (slow) devices: virtualization of slow I/O-devices (terminals, printers, tapes, etc.). Typical instances: AD_terminal, AD_printer.

14: cluster configuration service: allocation (deallocation) of nodes, loading, installation of POOL-processes, installation of clusters. Typical instance: AD_constructor (used by PASCAL_D-compiler, see [PASCAL-D]).

15: information exchange, support for user applications: definition of distributed catalogues, access to catalogues, catalogue-based look-up operations, etc. Typical instance: AD_RSO (see section 8).

16: user control: control of access rights (quotas); provides an initial "command interpreter", connects a user to Regional Service Office (RSO), etc. Typical instance: AD_user.

17: utilities (tools): compilers and other utilities; a compiler for PASCAL-D (which is an extension of PASCAL) is available which allows the definition and configuration of administrators (clusters) and remote procedure calls between administrators under program control (see [PASCAL-D]). Typical instances: AD_compiler, AD_TEMA, AD_DASCO (see section 8.3).

ADMINISTRATORS, CLUSTERS

Administrators are the basic abstraction mechanisms supporting distribution at the process level within the POOL system. An administrator can be considered as a generalization of an abstract object adjusted to the requirements of a distributed environment.

An administrator AD possesses the following main properties:

1. AD is a process running on an (arbitrary) EU.
2. AD encapsulates local objects (data) and defines operations on these objects.

3.a. AD allows access to the implemented operations by providing entries (service access points) e1, e2, ..., en.

3.b. An administrator AD_c can use a function e defined by another administrator AD_p by performing a remote procedure call AD_p.e(...).

3.c. AD must accept and handle several calls at the same time.

4. AD can use additional internal processes (running also on EU's) called slave processes. Slave processes provide "internal parallelism" for the simultaneous execution of some of the subtasks on which the arriving remote procedure calls are mapped within the administrator AD.
5. Implementation details of an administrator (including the management and scheduling of slave processes) are hidden from the "outside world".

In particular, administrators are applied as instruments to virtualize and control devices (resources). In the simplest case we have the following typical situation:

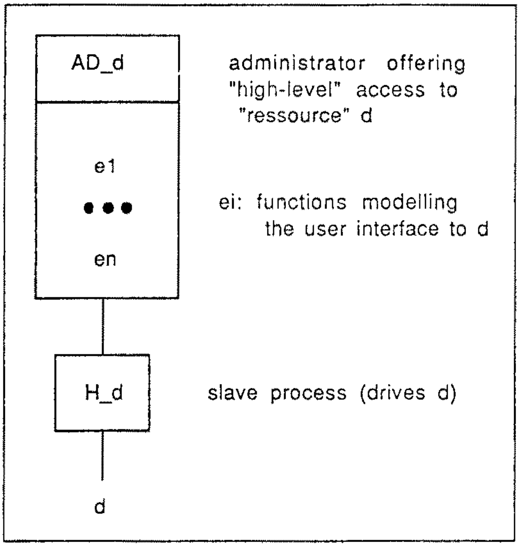

AD_d
administrator offering "high-level" access to "ressource" d
e1
en
ei: functions modelling the user interface to d
H_d
slave process (drives d)
d

figure 14

The administrator AD_d defines functions e1, ..., e2 which realize higher-level access to a resource d since everything connected with the handling of the physical device d is performed by the slave process H_d (H_d is not visible outside of AD).

A remote procedure call constitutes a relationship (c-p-relationship) between the calling administrator AD_c (consumer) and the called instance AD_p (producer). A set of administrators connected by the c-p-relationships is called a cluster. Thus, a cluster is an abstract (computational) network, the elements of which cooperate in order to achieve a common goal.

6. POOL-versions, access to POOL

6.1 POOL-versions

Two POOL-versions are implemented so far:

1. POOL_host: this is a simulation of POOL on a Siemens 7.570-P mainframe computer under the BS2000 operating system.
2. POOL_micro: this is a special architecture providing up to 15 execution units (Z80, MCS68010).

Both versions can be used together (in a transparent way). POOL_host allows us to extend the number of POOL-processes by adding virtual processors and thus to overcome some restrictions imposed by hardware (costs).

POOL_micro will be replaced by the (final) version

3. POOL_CCC: the communication method was described in sections 3, 4; it will provide up to 64 execution units (Z80, MCS68010, MCS68020).

The following table comprises the important characteristics of the two existing POOL-versions:

components	POOL_host	POOL_micro
EU	virtual processor	MCS68010, Z80
CU	virtual processor	Z80
type of elementary communication	a. common memory with semaphores b. ipc	serial/parallel interface, communication protocol
type of higher level communication	datagram, broadcast using link tables	datagram, broadcast (hardware supported)

The following figure shows the current POOL-configuration.

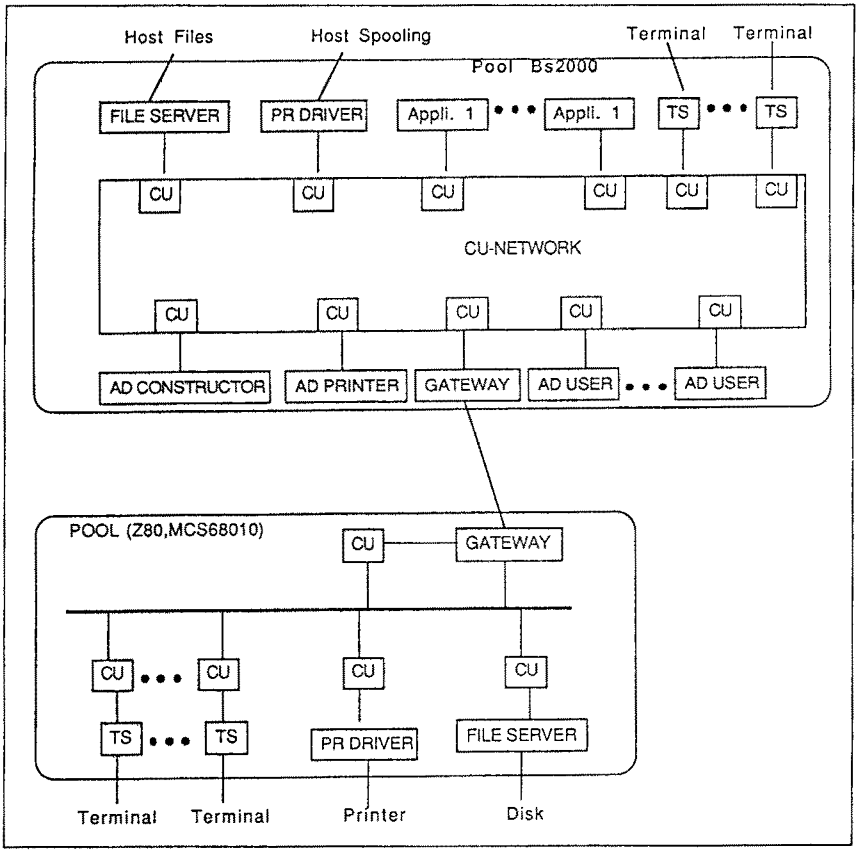


figure 15

6.2 Access to POOL via CANTUS

An obvious advantage can be derived from the fact that the functions of our operating system are distributed: there is no problem in principle to integrate further nodes (hosts, PCs) into POOL, thus increasing the number of available execution units.

The easiest way to add a PC, say, as a new node to POOL is to take the following steps:

1. Provide a simulation of the basic communication functions (using any communication method available for the PC, see below).
2. Provide a simulation of the CU - EU protocol.
3. Implement the (PASCAL) program defining the administrator AD_terminal which allows access to POOL_host and POOL_micro on the PC.

The following figure illustrates the situation:

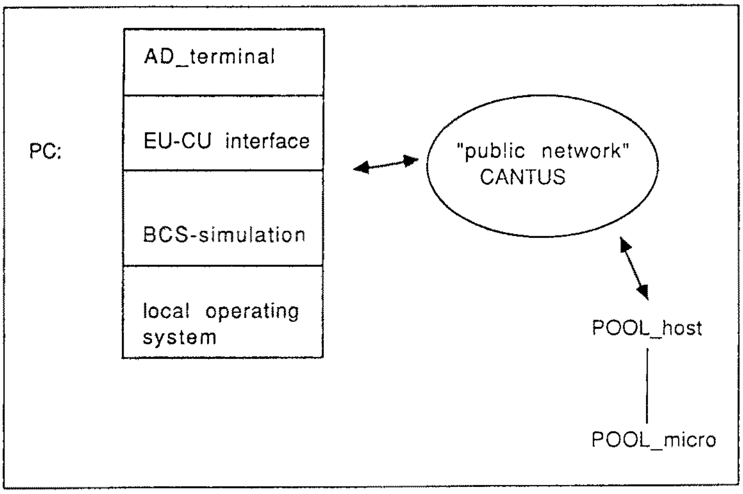


figure 16

The campus-wide network CANTUS (CAmpus NeTwork of the University of Saarland) provides the basis for connecting PCs of different types to POOL. CANTUS is a datagram-based local area network (see [Schuh 84]). At present, PCs of type SUN, SINIX (MX-2) and CP/M are connected to POOL.

6.3. Portability of POOL-processes

The integration of new nodes (PCs) into POOL (as described in 6.2) assumes the portability of POOL-processes (in particular of the instance AD_terminal). Obviously, we have the choice between

(a) an efficient implementation of administrators based on a (local) multitasking system BBS_EU (see [Heubel 86]), or

(b) a version which is portable but less efficient.

The second version uses the language PASCAL to

- describe a "PASCAL-node machine (PNM)" as a basis for the implementation of an administrator,
- define the interface between PNM and the "native" operating system (of the PC) as a set of PASCAL modules, and
- define the interface between the node machine PNM and the administrator running on it as a set of PASCAL modules. The organization of the PNM is illustrated by the following figure:

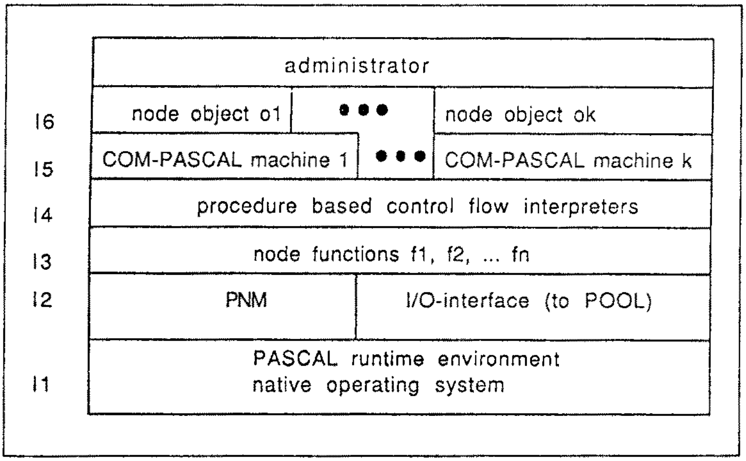


figure 17

l2: PNM is implemented against a PASCAL program on layer 1.

l3: Functions fi are:
  - construction (initialization) / deletion of COM-PASCAL machines;
  - interactions of COM-PASCAL machines;
  - implementation of the EU-CU-interface.

l4: interprets the control flow descriptors defined by layer l5; performs the scheduling of COM-PASCAL machines (providing a "quasi-parallel" mode of operation).

l5: COM-PASCAL machines consist of PASCAL-routines and descriptors defining the control flow (at the procedure level); COM-machines can communicate with each other.

l6: objects (internal processes) needed for the implementation of an administrator, implemented by the COM-machines.

See [Gerlach 85] for a detailed description.

7. Experiences with POOL

At present, most experiences listed below refer to the implementation of the system, not to it use.

- The realization of a distributed system (hardware and software) involves a tremendous amount of work.
- The testing of distributed (system) programs is very difficult and time-consuming.
- The simulation POOL_host of POOL on a host is important in order to provide a reasonable number of processes. However, the actions of POOL_host depend on the general load of the host; the definition of appropriate timer values is not easy.
- Tools are necessary to (automatically) port software modules to execution units of different types.
- Compiler-compiler systems and other (generic) methods must be applied to assist in the production of PASCAL-oriented software modules.
- Without a proper distributed application language the user will not accept and use the system. This experience stimulated the definition and implementation of PASCAL-D.
- The operating system support for PASCAL-D (including the asynchronous remote procedure call) was very good, the implementation of PASCAL-D relatively easy.
- There is much work to be done to provide the kind of environment which allows us to utilize the potentialities of a distributed system.

8. The user interface of the distributed system POOL

8.1. Higher-level communication services

In sections 3, 4 above, concerning the basic communication system (BCS) of POOL, we gave an overview of "low-level" communication services provided by BCS.

Typically, the situation of the user U of BCS is as follows: U wants to access a service S localized in and offered by a POOL-instance AD_S. We assume that the identification id(AD_S) = "string" and that this identification is unique within POOL. Then U has to take the following steps:

1. U has first to perform a search within POOL for an element with id(x) = "string". This search yields – if successful – the network address adr(AD_S).
2. Thereafter U will apply point-to-point-communication using adr(AD_S).

The search in step 1 is implemented as a multicast M(area, id = "string", quota). The first parameter defines the area the search has to cover while the second parameter specifies the object we are looking for (the third parameter will not be discussed here).

Obviously the user U has the problem of choosing a suitable subrange T_{sub} of the network topology T for the area which the multicast has to cover. Moreover, U has to cope with a possible failure of step 1 by modifying T_{sub} and iterating step 1. Thus, U has to rely on a certain knowledge of such technical details as network topology, subareas and so on. Such details are, however, better hidden from the user of a distributed system. We will therefore try to embed the "raw" services offered by BCS into an environment which affords the user an easier and more comfortable handling.

The basic idea is simple: we span a chain of instances (administrators) RSO_1, RSO_2, ..., RSO_n across the distributed system, where RSO stands for Regional Service Office.

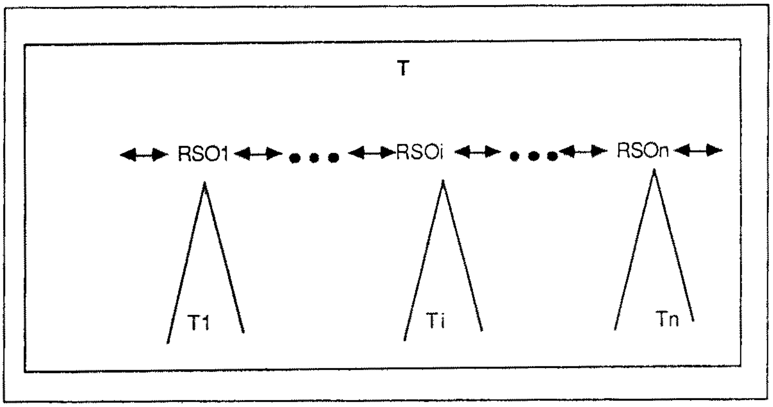


figure 18

These instances RSO_i have the following properties:

1. The chain of RSO-instances is bidirectional; that is, between every two adjacent instances RSO_i, RSO_j ($j = i - 1 \; mod \; n, \; i + 1 \; mod \; n$) exists the c-p-relationship.
2. Every RSO-instance can initiate a multicast $M(T_i, pr(x), ...)$ which covers a subarea T_i of T and looks within T_i for an object with property $pr(x)$. The areas T_i are defined in the following way:
   a. $\bigcup_{i=1}^{n} T_i \supset T$: the multicasts of all the elements T_i cover T completely.
   b. The size of T_i is as small as possible but we do not request that $T_i \cap T_j = 0$ (see property 3 below).
   c. $\|_{i=1}^{n} M(T_i, \; pr(x), ...) = B(T, \; pr(x), ...)$:
   the (parallel) execution of the multicasts initiated by all the RSO-elements implements a broadcast operation B. B yields the network address of an element with $pr(x)$ if such an element exists within T. The broadcast B can be invoked by an arbitrary element RSO_i which instructs its neighbours to act accordingly.
3. The number n of RSO-elements is not fixed; we are free to chose an appropriate number as long as the conditions above are fulfilled.
4. The RSO-chain is set up by the system once; thereafter, it possesses a certain "self-recovering" quality: every RSO_i checks its two neighbours regularly; if one of them, say RSO_j, is inoperable (does not answer) then a new element RSO_j* is elected by

RSO$_i$ which replaces the inaccessible element RSO$_j$. $T_j^* \supseteq T_j$ must hold for the areas covered by RSO$_j*$ and RSO$_j$.

5. The decentralized structure of the POOL-system is retained because all RSO-elements possess equal rights (that is, they provide the same services and differ only with respect to their local databases, see below).
6. Every user U is connected to an RSO-element when performing a "login". The login causes a multicast M(T$_{std}$, "RSO", ...) with a suitable area T$_{std}$ chosen by the system.
7. RSO$_i$ offers "higher-level" services to the user such as
   - construction and management of (structured) catalogues,
   - catalogue-based look-up operations which replace the "raw" multicast operations of BCS,
   - naming services,
   - observation of security mechanisms, allotment of capabilities,
   - mechanisms for the construction, configuration (reconfiguration) of cooperating sets of administrators (clusters),
   - compilation service by AD_PASCAL-D (PASCAL-D is an extension of PASCAL which allows the formulation of distributed applications (see [PASCAL-D]),
   - mail services, ...

Thus we end up with the following situation:

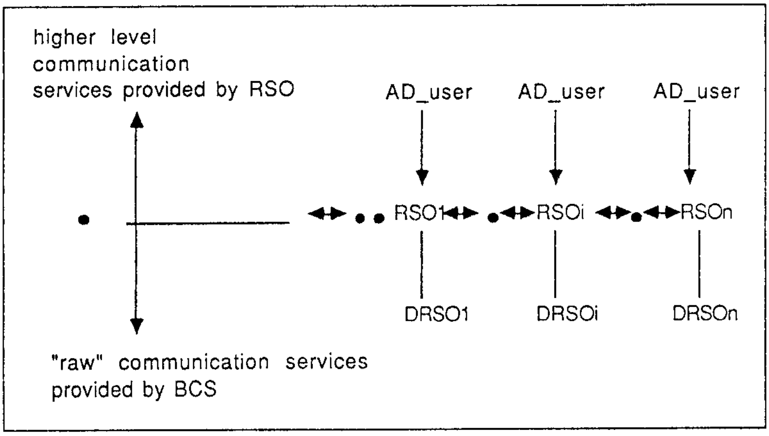


figure 19

The basic communication services are hidden from the user and replaced by higher-level operations. Within the system, every user is represented by a user administrator AD_user which is connected to an RSO-element (see property 6 above).

Note that the RSO-chain constitutes a "borderline" which separates low-level operations, the design and implementation of which must be done very carefully and needs special system programming skill, from higher-level operations which can be implemented using an appropriate higher level programming language as e.g. PASCAL-D. Thus, the introduction of the RSO-mechanism will considerably increase the "productivity" of programming.

Every RSO_i possesses a "private" database $DRSO_i$ which contains all the data specifying and characterizing the clients of RSO_i (that is, the user-administrators connected to RSO_i).

Thus, an RSO-element

- serves as a "local memory" or "cache storage" supporting fast and convenient access to the objects a user is actually working with, and
- informs other instances about global objects defined by one user (or by the system itself).

The reliability of these RSO-based services depends heavily on the availability of the local databases DRSO. If we are not content with the degree of reliability given by the lower system layers offering a general database capacity, then we can try to achieve the necessary redundancy with respect to the data DRSO(U) connected with one user U in the following way:

Let $H(id(U)) = (id(RSO_{i1}), id(RSO_{i2}), ..., id(RSO_{ik}))$ be a general hash function (known to every instance of the system) which yields the identifications of k RSO-elements (k > 1) when applied to the identification id(U) of a user. We assume that every element RSO_{i1}, RSO_{i2}, ... contains in its local database $DRSO_{i1}(U)$, $DRSO_{i2}(U)$, ... a copy of the actual data DRSO(U) of U. If one element RSO_{il} now fails, then U can work with one of the other RSO-elements possessing the redundant information.

For the time being we assume that the user is responsible for the update of its databases $DRSO_{i1}(U)$, $DRSO_{i2}(U)$, ... A systematic approach in the future could lead to a "knowledge-based" user interface where "knowledge" is distributed and can be lost as well as recovered. Note that there are other projects which try to introduce artificial intelligence techniques in order to establish and sustain assumptions about the global state of a distributed system (see e.g. [MOS], [DASH]).

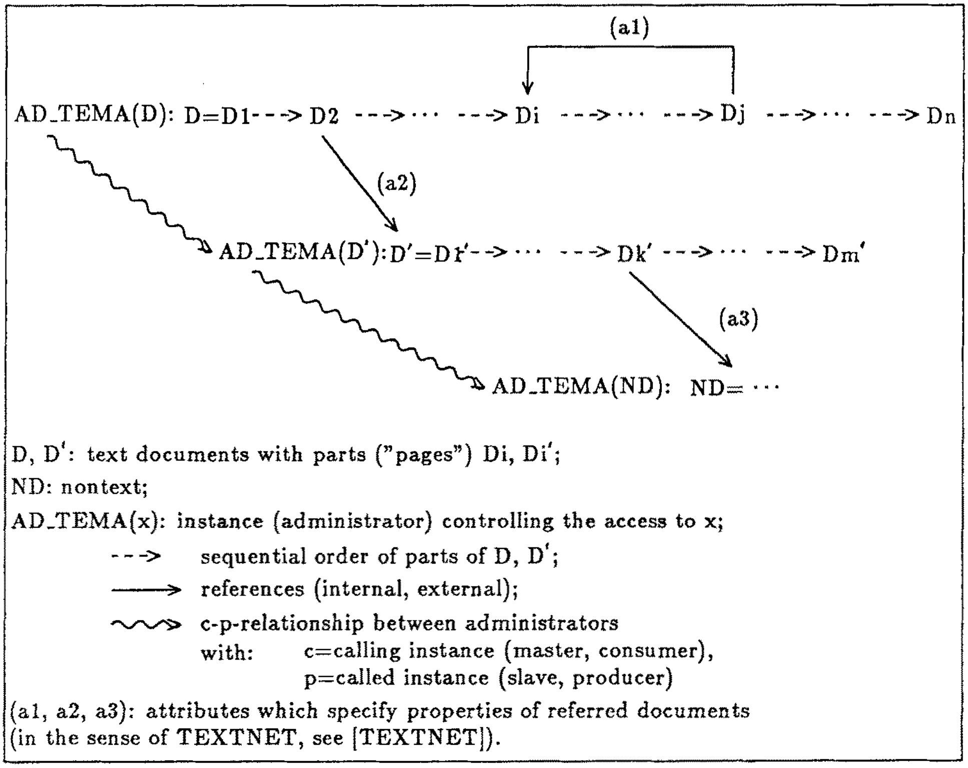
The user interface as defined by the RSO-elements is further extended by a number of tools, that is, programs which are themselves working in distributed mode and which provide certain services to all users.

In the following we describe very briefly two of these tools (for a more detailed description see [VAN], [MEISER], [NILAM]).

8.2. Hypertext and hypertext processing instances AD_TEMA

A hypertext (see [HYPER]) is a structure (graph), the nodes of which are chunks of text and/or "nontext" (meaning any data of other types) connected by references (links). These references can be

- internal references (to parts of the same document), or
- external references (to other documents or nontext). The following figure shows an example of a hypertext:


(a1)
AD_TEMA(D): D=D1 ---> D2 ---> ··· ---> Di ---> ··· ---> Dj ---> ··· ---> Dn
(a2)
AD_TEMA(D'): D'=D1' ---> ··· ---> Dk' ---> ··· ---> Dm'
(a3)
AD_TEMA(ND): ND= ···

D, D': text documents with parts ("pages") Di, Di';
ND: nontext;
AD_TEMA(x): instance (administrator) controlling the access to x;
- - -> sequential order of parts of D, D';
——> references (internal, external);
⟿ c-p-relationship between administrators
with: c=calling instance (master, consumer),
p=called instance (slave, producer)
(a1, a2, a3): attributes which specify properties of referred documents (in the sense of TEXTNET, see [TEXTNET]).

figure 20

Figure 21 below shows that instances (servers) AD_t are used to provide storage for hypertext data of type t (t: text, source code, graphics, ...). An instance AD_TEMA(x) allows the manipulation of exactly one hypertext entity (e.g. document) x: it performs the access to the server on which x is located and causes portions of x to be transported on demand from the server to AD_TEMA and vice versa.

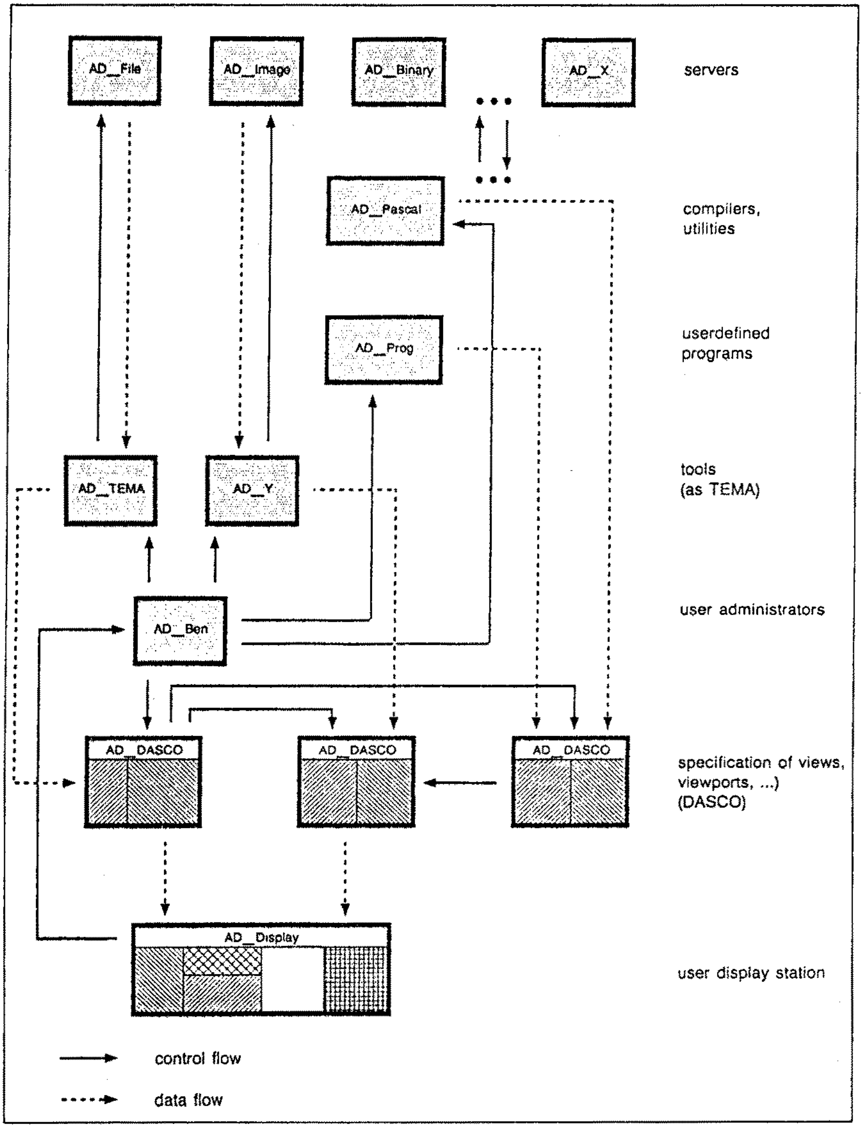


figure 21

The instances AD_TEMA(xi), working on hypertext data xi linked by references, form a network, the structure of which closely resembles the linkage structure of the hypertext; the edges of this network express c-p-relationships between the nodes connected by these edges. We call such a set of cooperating instances a cluster (compare 5).

In the example above, the instances AD_TEMA(D), AD_TEMA(D′) and AD_TEMA(ND) are elements of a cluster. Typical functions of a hypertext editor such as "browsing", for example, must then be implemented as actions of clusters.

8.3. Data stream control instances AD_DASCO

When working in a distributed environment we have the problem of presenting to the user values which are produced by different sources (instances) at different slices of time. We assume that the user operates a user display station (uds) consisting of one device or several devices (displays, monitors) controlled by an instance AD_uds. We assume furthermore that a window manager is part of AD_uds and performs the manipulation of windows on uds.

A user U, however, does not work with a real device (display, window) but with virtual objects called views and viewports.

U can specify views, viewports, their layout and the mapping of virtual objects onto real devices.

There is always a one-to-one correspondence between a viewport and a "value-delivering instance" AD_value which produces and delivers the stream of data presented in this viewport.

We introduce instances AD_DASCO as a means of specifying and controlling these virtual objects and the data flow between value-delivering instances and the "presentation instance" AD_uds.

The construction process of AD_DASCO instances is iterative in the following sense: a value-delivering instance AD_value which corresponds to a viewport vp of a view vw can itself be a composed object and thus specify a view vw' by simply defining AD_value = AD_DASCO' (see the example below). It is furthermore possible to dynamically exchange an instance AD_value with another instance AD_value'.

The following example illustrates the situation where a user is working on a document docl (controlled by AD_TEMA$_1$) and looking into documents doc2, doc3 and into graphical data (controlled by administrators AD_TEMA$_1$, AD_TEMA$_2$ and AD_ND). AD_TEMA$_1$, AD_TEMA$_2$, AD_TEMA$_3$ and AD_ND are value-delivering instances for AD_DASCO$_1$ and AD_DASCO$_2$. AD_DASCO$_2$ defines a view which is actually a viewport of AD_DASCO$_1$.

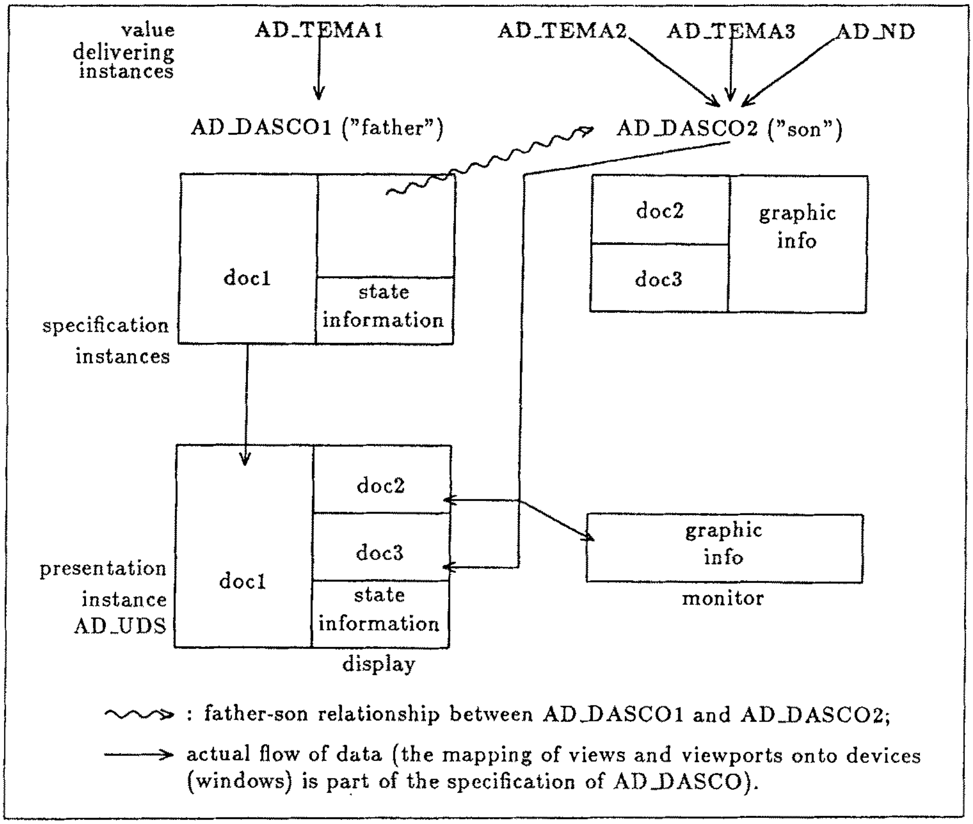

value delivering instances
AD_TEMA1
AD_TEMA2
AD_TEMA3
AD_ND
AD_DASCO1 ("father")
AD_DASCO2 ("son")
specification instances
doc1
state information
doc2
doc3
graphic info
presentation instance AD_UDS
doc1
doc2
doc3
state information
display
graphic info
monitor
: father-son relationship between AD_DASCO1 and AD_DASCO2;
actual flow of data (the mapping of views and viewports onto devices (windows) is part of the specification of AD_DASCO).

figure 22

8.4. Conclusion

The utilities presented in the last sections of this paper constitute an important part of a "parallel user environment" which views the user of a distributed system as one of many cooperating instances (see figure 21). At present we know next to nothing about the nature and structure of such a parallel user environment. It is therefore very important to build an experimental system, to work with it and to explore its potentialities.

The combination of hypertext concepts with the advantages offered by distributed processing and the use of distributed databases can and should be of particular value in supporting the development of distributed programs.

Note that a new computer architecture will not acquire a widespread user community as long as its user interface is "exotic" and/or hard to work with. Users expect "innovative" systems to offer user interfaces which are at least as good as those given by an average PC.

Bibliography

[DASH] Anderson,D.P., Ferrari,D., Rangan,P.V., Tzou,S.-Y.: The DASH Project: Issues in the Design of Very Large Distributed Systems. Report No. UCB/CSD 87/338. Computer Science Division, University of California, Berkeley, California 94720, Jan. 87.

[FPS 86] Floating Point Systems: The FPS T Series. FPS MC TS01 3/86 5M OPP.

[Gerlach 85] Gerlach,L., Spurk,R.: Die Implementierung des verteilten Betriebssystems POOL. Report 07/85, Sonderforschungsbereich 124, Fachbereich Informatik, Universität des Saarlandes, 1985.

[Guzman 83] Guzman,A., Gerzso,M., Norkin,K.B., Vilenkin,S.Y.: The Conversion via Software of a SIMD Processor into a MIMD Processor. in Computer Architectures for Spatially Distributed Data, Ed. by Freeman, H., Pieroni, G.G., Springer-Verlag, 1985.

[Heubel 86] Heubel,T.: Die Virtualisierung von POOL-Verarbeitungsknoten durch ein Basis-Betriebssystem. Internal report, Sonderforschungsbereich 124, Fachbereich Informatik, Universität des Saarlandes, Apr. 1986.

[Hillis 86] Hillis,W.P.: The Connection Machine. The MIT-Press, Cambridge, Massachusetts 02142, 1986.

[HYPER] S. Carmody, W. Gross, T.H. Nelson, D.E. Rice, A. van Dam: A Hypertext Editing System for the /360, Pertinent Concepts in Computer Graphics, M. Faiman, J. Nievergelt, Ed., Illinois Press, March 1969, pp. 291-330.

[Kleinrock 75] Kleinrock,L.: Queueing Systems, Vol. I: Theory. New York: Wiley-Interscience, 1975.

[Krass] Krass,J.: Layout eines verteilten Systems in CCC-Architektur. M.Sc. thesis, Fachbereich Informatik, Universität des Saarlandes, in preparation.

[Liskov 84] Liskov,B.: The Argus Language and System. in Distributed Systems - Methods and Tools for Specification, Lecture Notes in Computer Science 190, Springer-Verlag, 1985.

[Malowaniec] Malowaniec,K.T.: Circuit Switching Networks for Distributed Systems. Ph.D. thesis, Mathematisch-Naturwissenschaftliche Fakultät, Universität des Saarlandes, in preparation.

[MEISER] Meiser,D.: Hypertext-verteilter Editor. M.Sc. thesis, Fachbereich Informatik, Universität des Saarlandes, in preparation.

[MOS] Barak,A., Litman,A.: MOS: A Multicomputer Distributed Operating System. Software-Practice and Experience, Vol. 15(8), 725-737 (Aug. 85).

[NILAM] Nilam,S.: Verteilte Display-Manager. M.Sc. thesis, Fachbereich Informatik, Universität des Saarlandes, in preparation.

[PASCAL-D] C. Neusius, H. Scheidig, R. Spurk: PASCAL-D, a distributed version of PASCAL and its implementation, Universität des Saarlandes, Rechenzentrum, in preparation.

[Preparata 81] Preparata,F.P., Vuillemin,J.: The cube-connected cycles: a versatile network for parallel computation. Commun. ACM 25, 5 (May 1981), 300-309.

[Scheidig 83] Scheidig,H.: POOL. Ein verteiltes System aus vielen Prozessoren - Aufbau und Wirkungsweise. Report 04/83, Sonderforschungsbereich 124, Fachbereich Informatik, Universität des Saarlandes, 1983.

[Scheidig 85] Scheidig,H.: The Ten Laws underlying the Design of the Distributed System POOL. Report 33/85, Sonderforschungsbereich 124, Fachbereich Informatik, Universität des Saarlandes, 1985.

[Schneider] Schneider,M.: Entwicklung eines intelligenten Links in VLSI-Technik für CCC-Netzkommunikation. M.Sc. thesis, Fachbereich Informatik, Universität des Saarlandes, in preparation.

[Schuh 84] Schuh,H.J., Spaniol,P.: CANTUS a packet switching point-to-point network. International Symposium on Communication and Computer Networks, Networks INDIA 84,IFIP/UNESCO, 1984.

[SUPRENUM] Behr,T.M., Giloa,W.K., Mühlenbein,H.: SUPRENUM, the German Supercomputer Project - Rational and Concepts. IEEE International Conference on Parallel Processing, 1986.

[TEXTNET] R.H. Trigg, M. Weiser: TEXTNET, a network based approach to text handling, ACM transactions on office information Systems, Vol. 4, Nr. 1, January 1986, pages 1–23.

[Upfal 84] Upfal,E.: Efficient Schemas for Parallel Communication. J. ACM 31, 3 (July 1984), 507-517.

[VAN] H. Scheidig, D. Meiser, C. Kraus, S. Nilam, D. Prinz: Verteilte Anwendungen auf lokalen Netzen, Universität des Saarlandes, Rechenzentrum, 1987.

[Wittie 81] Wittie,L.D.: Communication structures for large networks of microcomputers. IEEE Trans. Comput. C-30, 4 (Apr. 1981), 264-273.