In: Infotech State of the Art Report on Distributed Computer Systems, 1979.

Distributed Computer Systems - A Design Methodology and its Application to the Design of Distributed Data Base Systems[*]

C. V. Ramamoorthy, G. S. Ho and Benjamin W. Wah

Computer Science Division, Department of EECS,
and the Electronics Research Laboratory
University of California, Berkeley
Berkeley, Calif. 94720

[*] Research supported by the Ballistic Missile Defense Contract DASG60-77-C-0138.

ABSTRACT

In this paper, a systematic design and development methodology for large distributed computer systems is presented. The methodology will provide guidelines for the systematic design and construction of distributed computer systems so that system requirements are satisfied if there exists a feasible design under the given constraints. In order for the methodology to be general enough and applicable to a wide class of computer systems, the design and development approaches are studied in the logical level. Following the methodology, design issues of distributed data base systems are examined. The issues are classified in a top-down fashion from both users' point and designers' point of view. The distributed data base system is decomposed into two successive architectural levels: DCS memory and nodal memory. The nodal memory is further divided into virtual memory and data base machine. The architectural and operational aspectsin each level will be discussed in detail.

1. INTRODUCTION

The character, demand and economics of computer systems have been continuously changing and the need for new computer organizations and architectures seems unavoidable. Until recently major emphasis has been placed on maximizing the hardware utilization of uniprocessor systems or tightly coupled multiprocessor systems. With the advances in large scale integrated logic and memory technology, coupled with the exploding size and complexity of the application areas, the designers are encouraged to consider the distribution of processing, leading to distributed architectures. However, current approaches to the design of computer systems and data bases and their evaluation, unfortunately, are based primary on experience and intuition. The specification, design, implementation and evaluation of large embedded computer systems, such as the air traffic control systems, ballistic missile control systems, patient monitoring systems and large archival data bases, are very expensive, difficult to test adequately, slow to deploy, and difficult to adapt to changing requirements [DAV76]. In order to cope with these problems, a systematic approach for the design and development of large distributed computer systems (DCSs) is urgently needed.

In this paper, a systematic design and development methodology for large DCSs is presented. The methodology will provide guidelines for the systematic design and construction of distributed systems so that system requirements are satisfied if there exists a feasible design under the given constraints. Based on the methodology, the design issues of distributed data base systems and some solutions are examined. The paper is divided into eight sections. In section 2, the difficulties in the design of DCSs and the characteristics and objectives of the methodology are outlined. In section 3, the development phases in the methodology are discussed in detail. In order for the methodology to be general enough and applicable to a wide class of DCSs, the design and development approaches are studied only in the logical level. Specific architectural and control issues of distributed systems like communication protocol, distributed control, reconfiguration, etc., are discussed in a previous paper [RAM76a]. Following the methodology, design issues of distributed data base systems are examined in section 6. These issues are classified in a top-down fashion from both users' point and designers' point of view. We will concentrate on the architectural and operational issues in the rest of the paper. The distributed data base system is decomposed into two successive architectural levels: DCS memory and nodal memory. The nodal memory is further divided into virtual memory and data base machine. The architectural and operational issues in each level will be discussed in sections 6 and 7 respectively. Finally, section 8 provides the conclusion.

1.1 Definition of DCS

Basically, a Distributed Computer System (DCS) is considered as an interconnection of digital systems called Processing Elements (PEs), each having certain processing capabilities and communicating with other. This definition encompasses a wide range of configurations from an unipro-cessor system with different functional units to multipli-city of general purpose computers (e.g. ARPANET). In gen-eral, what people call "distributed systems" vary in charac-ter and scope from each other [RAM76a]. So far there is no accepted definition and basis for classifying these systems. In [JEN75], a texonomy of the interconnection of distributed systems was given based on the transfer strategy, transfer control method and transfer path structure. In [RAV76], distributed systems are categorized based on the distribu-tion of functions among processing agents and the degree of interaction among agents. In this paper, we will limit our discussion to a class of DCSs which have an interconnection of dedicated/shared, programmable, functional Processing Elements (PEs) working on a set of related jobs.

1.2 Motivations for DCS

There are several motivations for the development of distributed computer systems and among them the most impor-tant ones are: i) modularity and structural implementation, ii) reliability and availability, iii) throughput and response time, iv) geographical distribution and resource sharing, and v) application orientation. Modularity and structural implementation is closely related to the ease with which system modifications can be achieved. The DCS design approach encourages modularity and permits expansion of system over a wide range of sizes to suit a given application. It also permits graceful degradation in performance when some modules fail.

Another important motivation for distributed processing is the potential to provide more reliable systems with improved availability despite some system failures. Thus in a distributed processing system, it is possible to provide a comprehensive error detection mechanism and isolate the faulty modules without impairing the entire system operation. For example, in a distributed data base , if multiple copies of data files are kept in different processing nodes, files lost due to a node crash can be recovered from their copies.

Improved throughput and response time can be a major consideration in certain applications. In a distributed system, the throughput can be increased by the exploitation of parallelism. In real time environments, the DCS approach of using multiple processors and multiple data files can meet the time constraints that might otherwise be impossible in a uniprocessor system [WAN72].

Finally, geographical distributions and resource sharing is another important motivation for many of today's distributed systems. In many applications, such as airline reservation systems, processing nodes are dispersed geographically but working in cooperation with each other. In these applications, most of the transactions are local and need fast response time. However, some information such as flight reservations and some utility programs are shared by several users. The distribution of processing and/or data base provides improved response time and ability to handle large volumes of information which might otherwise be impossible in conventional systems.

2. DESIGN METHODOLOGY

2.1 Difficulties in the Design of DCS

The major difficulty in the design of DCSs is the largeness of the systems. The activities of the systems are so varied and so complex that they are beyond the grasp of a single individual. For example, the BMD systems include, besides the data processing subsystem, the radar and missile subsystems. Each of these subsystems requires special expertise to design, implement, and enhance its operations. As a result, each subsystem is usually developed and maintained by a group of experts who have little knowledge of the other subsystems. This produces great difficulties in synchronizing and optimizing the development process. One common problem has been that some final decisions on primitives (essential system characteristics) are made in one subsystem, generally without considering the overall system requirements. These early commitments bias the development process and force and restrict the choices of the other primitives to accommodate them, which, in turn, impose undue constraints on design freedom and reduce the flexibility during integrating and interfacing the system. As a result, the development process is more expensive and time consuming than it should be, and the design that is obtained is usually far from optimum.

Another problem faced in the development of large computer systems is the rapid deployment requirement of the systems. For example, the ballistic missile defense system is developed in a rapidly changing enviroment. The system must react to an ever increasing threat complexity and rapidly improving technology. This forces the requirement for a development approach which will insure rapid deployment but have the flexibility to rapidly react to external changes [VIC76]. However, more too often, systems are designed without taking into account the provision for future changes. When the system evolves, the changes are incorporated into the system in a very disorganized manner. As a result, the unstructuredness (or the entropy) of the system increases enomously [BEL77] and leads to a regenerative, highly non-linear, increase in the effort and cost of system maintainence [LEH76]. In addition to this, the reliability and the integrity of the system are also jeopardized greatly.

In large scale critical real time systems, such as the air traffic control system, the development process is further complicated by the real time constraints and the criticalities of the systems. These systems must perform all the required functions correctly within the given time limits otherwise a large penalty have to be paid (for example, large loss of life due to a plane crash). However, these systems cannot be tested in its real operational enviroments. As a result, system validation has to rely heavily on analysis and simulation. Due to the high complexity of the systems, exhaustive testing will be impossible. The only solution is to design the system in a orderly way so that both validation and testing can be done efficiently.

2.2 Characteristics of the Methodology

The philosophy behind the methodology is based on hierarchical modelling of a DCS. the objective of establishing the hierarchy is to map the overall system requirements successively into lower levels of finer detail. At the top level in the hierarchy, the requirements described are abstract and the coupling between various attributes and associated functions may be loose. As we proceed down the levels, the characteristics of various functions and the attributes are elaborated and become more specific (Figure 1). The use of abstraction at the top level allows a designer to initially express the system requirements in a very general manner and with little regard to the details of

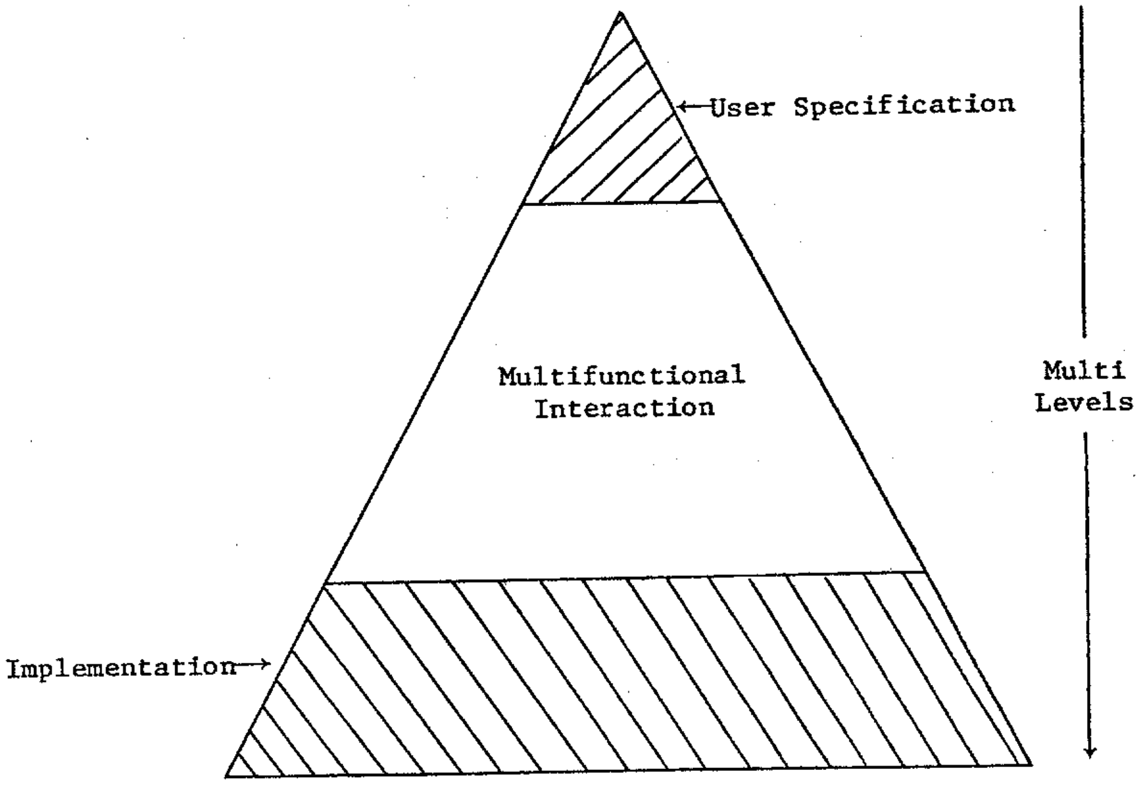


Figure 1. Hierarchical Design

the design and implementation. These initial system requirements are then refined in a step by step manner by gradually introducing more and more details (e.g. constraints and attributes) of the system. This combination of abstraction and stepwise refinement enables the designer to overcome the problem of complexity inherent in the construction of a complex DPS by allowing him to concentrate on the relevant aspects of his design incrementally at any given time, without worrying about other details. By this hierarchical approach, the assumptions and decisions made throughout the design process can be traced systematically and any revisions or modifications of the design as a result of the design development can easily be incorporated. However, it must be emphasised that it does not mean the whole design process can be automated. The engineering decisions will often be very complex and dependent on the experience of the designers. In summary, the design methodology is a first step trying to achieve:

(1) guarantee that the architecture (statement of need, system objectives, and constraints) of the problem be preserved.

(2) support orderly evolution of the system satisfying the constraints such as performance, reliability, etc. without major revisions.

(3) provide formal (mathematically rigorous) basis for the approach allowing precise evaluation of completeness, consistency and correctness of

requirements at any level of definition.

(4) represent effectively and efficiently the decision making constraints of a DPS by a specification language.

(5) provide design attributes and documentation for evolution (growth and modification) so that changes can be made without reconsidering the whole design process.

3. THE FOUR PHASES OF THE METHODOLOGY

The design methodology proposed here can be broken down into four successive phases (Figure 2):

(1) Requirement and specification phase

(2) Design phase

(3) Implementation phase

(4) Evaluation and validation phase

The requirement and specification phase starts with some (possibly incomplete, vague, and informal) system requirements that approximate the desired system, and finishes when the modified and elaborated requirements have been formally encoded and tested to the satisfaction of the system engineers and the "customers". This is the most difficult but also the most important step in the development process. Experience has shown that many design failures are due to either ill-defined (inconsistent and unclear) requirements or misinterpretation of the original problem statement. These account up to 85 percent of requirement errors [BOE75,BEL76]. In order to avoid the above mistakes, a formal specification language is used here. It has been shown by a TRW study that more than 50% of these errors can be avoided by formal specification (Figure 3).

The design phase starts with the requirement specifications and finishes when the system specifications are produced. The objective is to optimize and organize the sys-

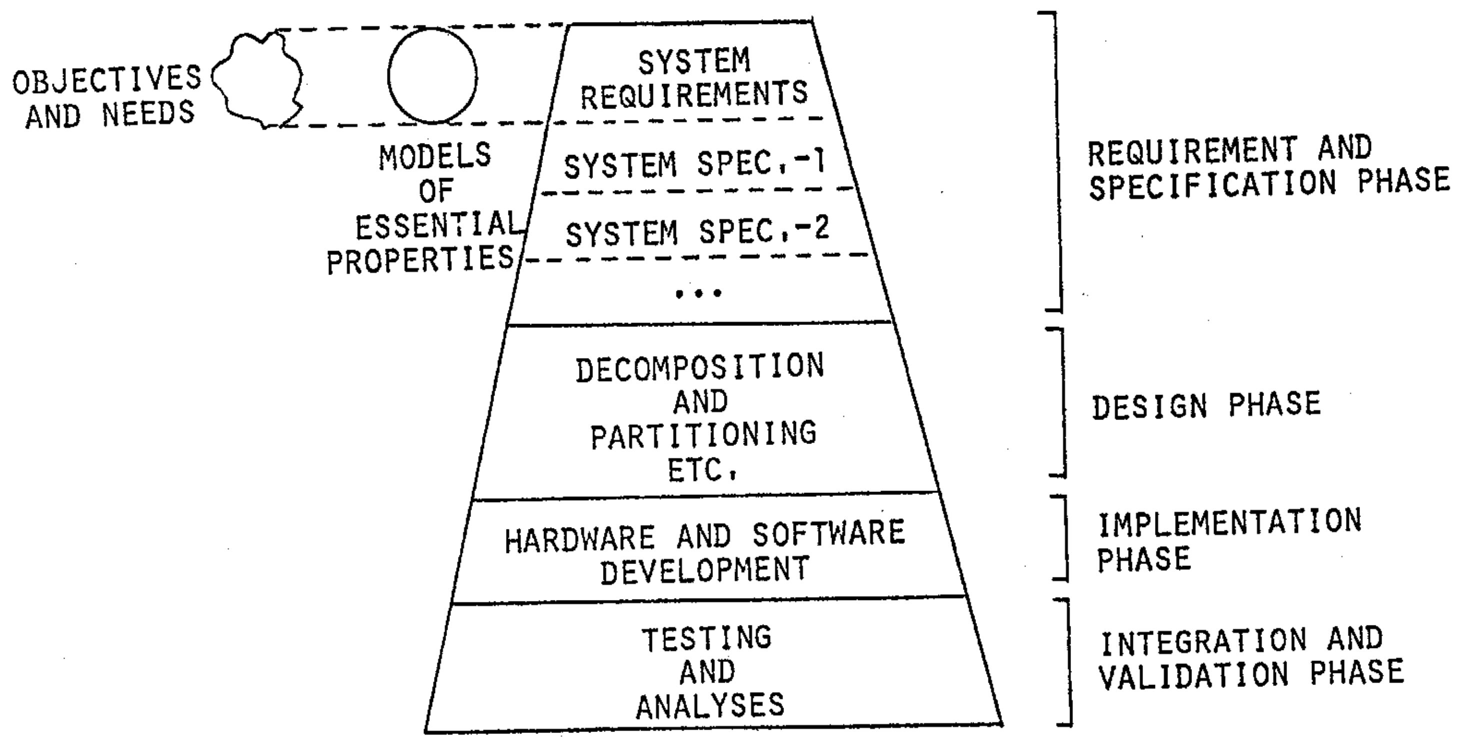

OBJECTIVES AND NEEDS
MODELS OF ESSENTIAL PROPERTIES
SYSTEM REQUIREMENTS
SYSTEM SPEC.-1
SYSTEM SPEC.-2
...
DECOMPOSITION AND PARTITIONING ETC.
HARDWARE AND SOFTWARE DEVELOPMENT
TESTING AND ANALYSES
REQUIREMENT AND SPECIFICATION PHASE
DESIGN PHASE
IMPLEMENTATION PHASE
INTEGRATION AND VALIDATION PHASE

Figure 2 Top Down Development Methodology

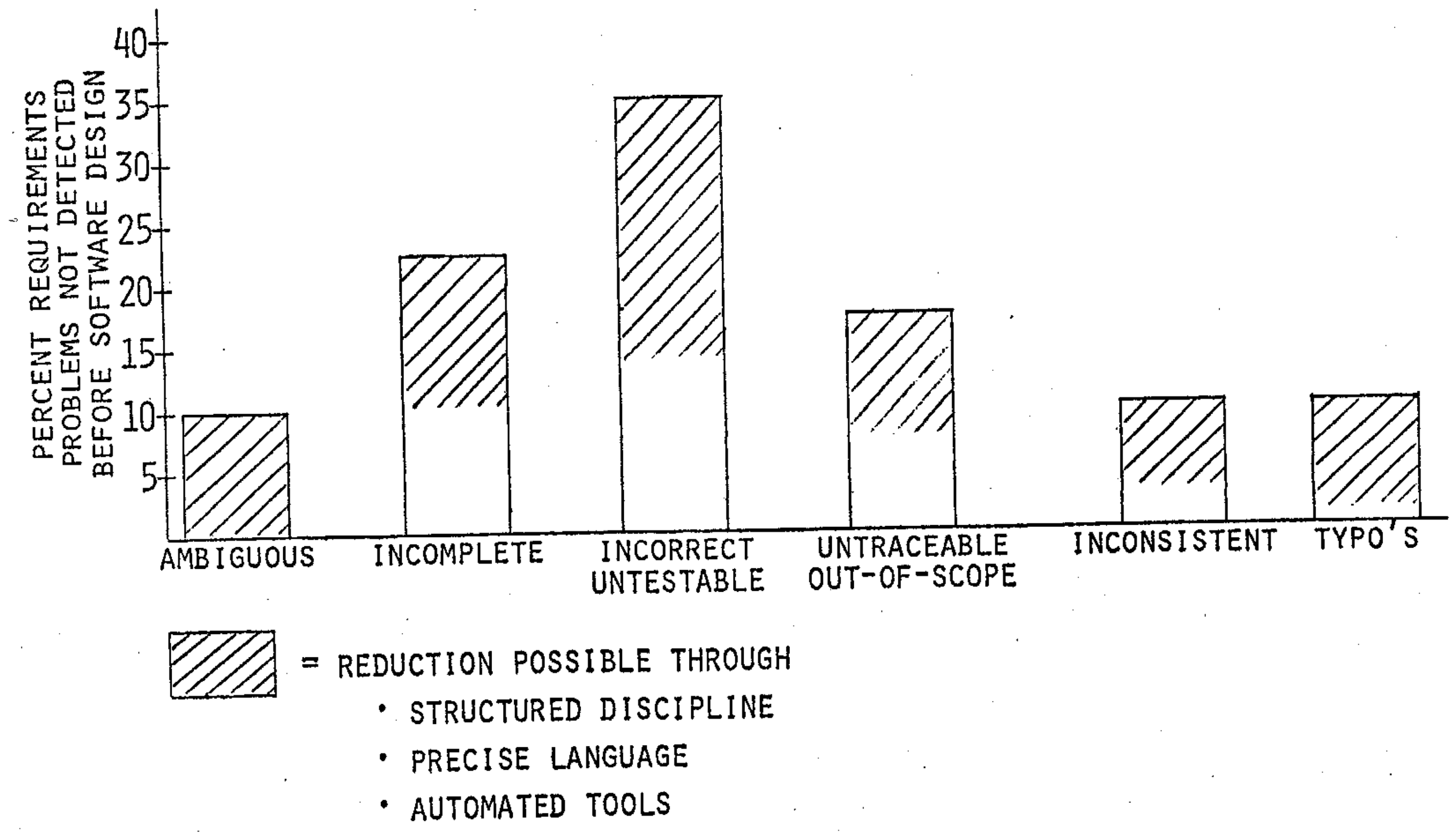


Figure 3 Typical Requirement Error Distribution

tem in a well formed structure. It involves an hierarchy of decomposition and partitioning of the system into subsystems. Decomposition is the process of dividing the system into several levels of components and subcomponents, and partitioning is the process of grouping these components and subcomponents into subsystems so to minimize the among of interactions and to satisfy the performance and reliability constraints of the system. After each decomposition and partitioning step, the subsystems are verified to be consistent to the original system. Any discrepancies and mismatches are corrected before they can propagate into the next level. After the design process, the system functions will be well specified and will be ready for implementation.

The implementation phase takes the system specification and develops the system architecture. It then maps the system functions into either hardware or software functions. It is only at this step that physical constraints and technology comes into consideration.

The final step is the evaluation and validation of the system. This phase uses the bottom up validation approach. It takes the final design and ensures that the system meets the original requirements. This step uses both analytical modelling and simulation. Mistakes or unfulfiled requirements are traced back to the source of the error. The system is then redesigned from that point. Since the system is broken down hierarchically, only the subsystems affected by the error and therefore only those that are stemming from the error point have to be redesigned.

It should be mentioned that the development process is not a straight top down process. Tests and checks are conducted throughout the development process. Whenever errors are found, the design is backed up to the previous level. Therefore there is a feed-back path from each development process back to the previous one. In the following sections, the requirement and specificaton phase, and the design phase of the methodology will be discussed in more detail. However, the implementation and validation phases are too technology and architecture dependent and are beyond the scope of this paper.

3.1 Requirement and Specification Phase

The requirement and specification phase starts with informally specified users' needs and elaborates on them to generate the formal system requirement specifications. These specifictions are used for two purposes: (1) as a problem definition of the design process, (2) as a means against which an implementation may be validated. This phase consists of four major steps (Figure 4): (i) requirement elaboration, (ii) requirement specification and attribute formulation, (iii) process definition, and (iv) verification of requirements.

3.1.1 Requirement Elaboration

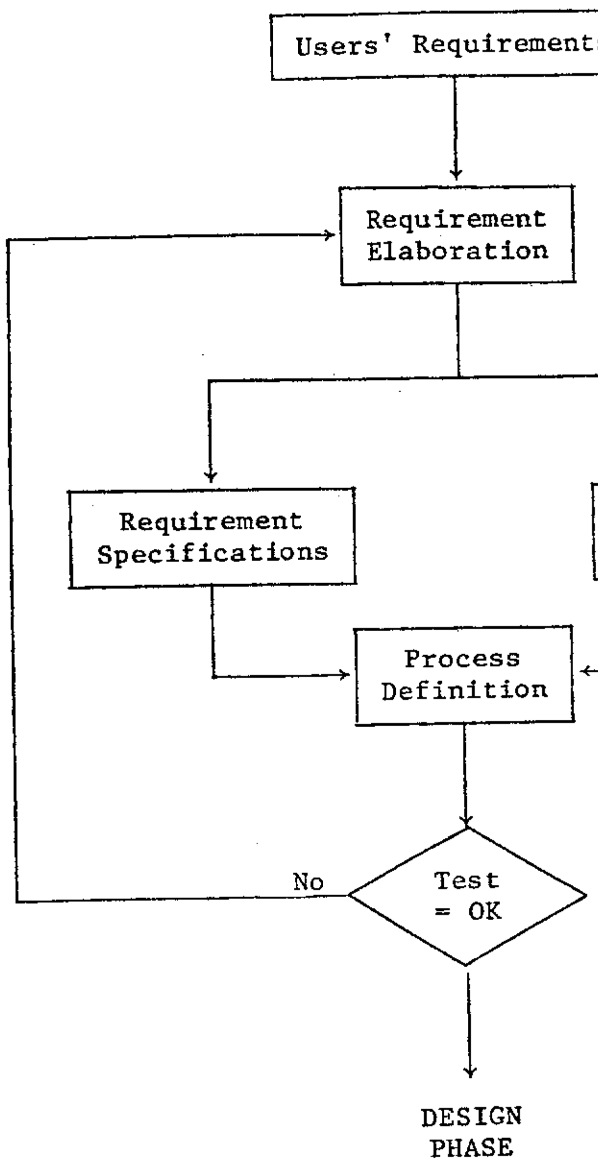


Figure 4. Requirement and Specification Phase

The requirement elaboration step can be considered as a problem understanding stage. The objective is to let the requirement engineers to have a bird's-eye view on the operations of the system. In this step, the closed system approach is chosen. In a closed system, the designed system and its enviroment are considered as a single entity. The activities (between the designed system and its enviroment) in the closed system are identified and investigated in a homogeneous manner. In this way, the closed system configuration can be formulated. For example, the air traffic control system has to interact with the airplanes, the radars, the airline offices, the flight service stations, etc. The closed system configuration can be formulated as shown in Figure 5. Based on the closed system configuration, the system requirements and system attributes can then be formulated.

3.1.2 Requirement Specification and Attribute Formulation

From the required behavior of the system, the system objectives can be formally expressed in the system requirements and system attributes [VIC76]. The system requirements are the objectives and constraints which the system must satisfy. Any system which meets the requirements is a candidate solution to the users' problem. Attributes, on the other hand, specify either options or evaluation criteria for qualitative comparisons of competing systems that meet the system requirements. They are used to specify the

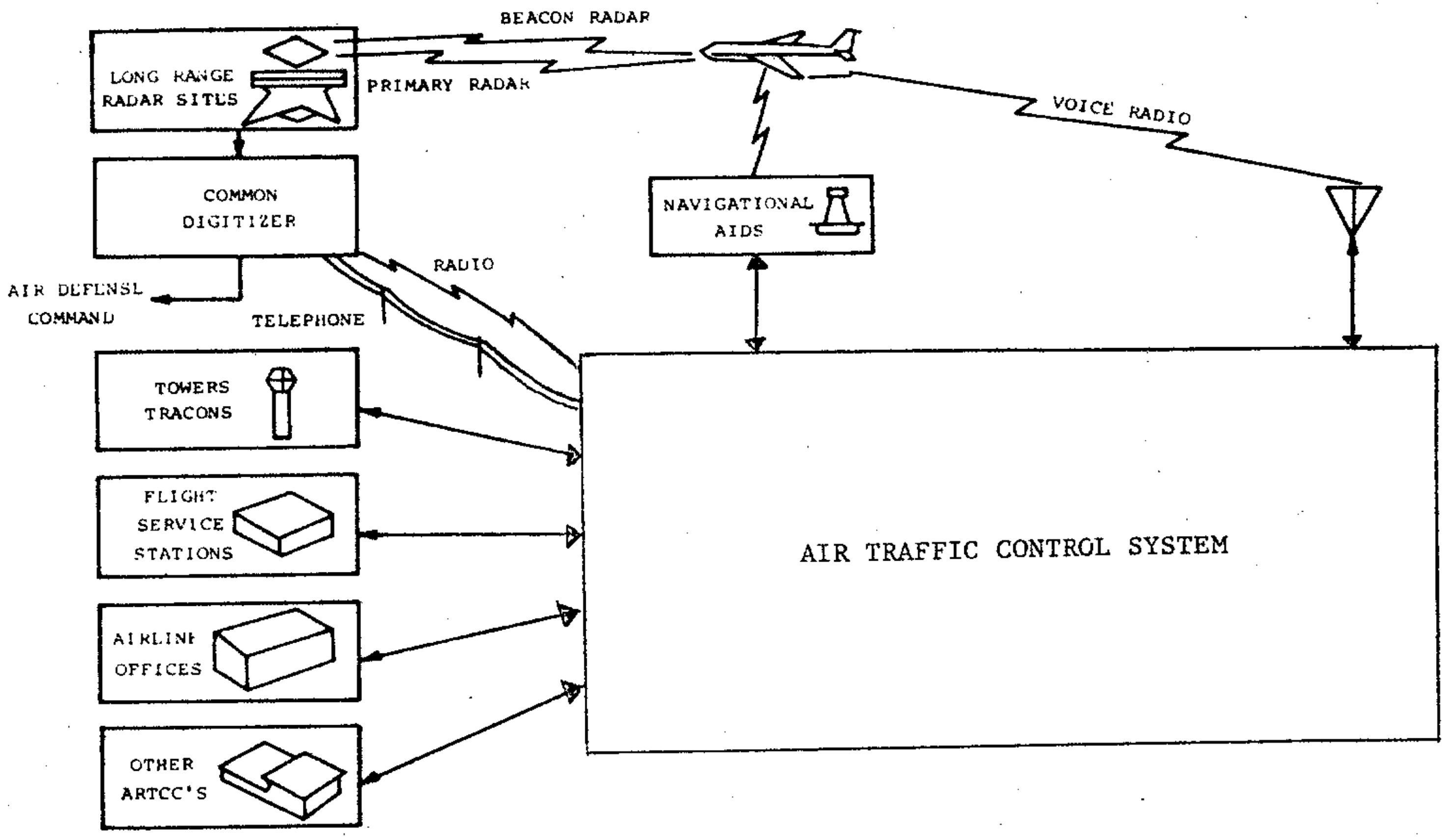


Figure 5 Closed System Configuration of a Air Traffic Control System

preferences of the users. The generation of these system requirements and attributes is the requirement specification and attribute formulation step.

(A) Requirement Specification

Based on the closed system configuration, the system requirements are specified in terms of the interfaces of the designed system and its enviroment. The interfaces specify the input (stimulus) and output (response) relations of the system. These input to output mappings can be expressed either vigorously in mathematical formulas [WYM76,] or less formally in a specification language [RAM78c,ROS77]. In real-world situations, the problems are so complex that pure mathematical formulation is usually impossible. Therefore in this methodology, the approach of using a specification language is chosen.

A specification language is a syntactically and semantically well defined language possibly intermixed with mathematical equations. Its whole purpose is to provide a efficient and effective medium for defining the system requirements. Many specification languages have been developed previously [CON77,ROS77,TEI77,BEL76,HAM76]. This paper does not intend to developing a new specification language. We will choose a specification language and express the system requirements in it. In choosing the specification language for a distributed system, the constructability and comprehensibility of the language must be evaluated carefully. The specification language must be able to express the functional requirements of the system efficiently and be easily understood by the customers (users) and the requirement engineers. It must be modular enough to express the inherent modularity of a distributed system. It must be equipped with powerful communication constructs in order to specify the interactions between system modules and provide capabilities of performance specifications in the case of real time systems. The language should be amenable to both static (hierarchical relationship, data definition, etc.) and dynamic (control flow and data flow) analyses [BEL76,HAM76,PET77]. It would be extremely helpful if system simulation can easily be generated from the system specifications as similation is oftenly used in predicting the characteristics and performance of a distributed system. Finally, the language should be backed up by a specification data base management system and powerful graphical supports to provide easy and efficient accesses to the designed system.

One great problem in the specification of requirements is the misinterpretation of the original system requirements. A plausible solution is to use multiple independent specification teams to develop the system specifications from the closed system configuration. This approach has been used in the development of critical real-time software for nuclear power plants [LON77]. In that project, two independent specification teams are used. Each specification is first "validated" against the original requirements and then "compared" against the other. Discrepancies are resolved to the satisfaction of both teams. By this multiple specification approach, most of the errors due to ambiguities and misinterpretations can be corrected before they can propagate into the next phase.

(B) Attribute Formulation

Attributes are the evaluation criteria for qualitative comparations of competing systems that meet the system requirements. For a distributed system, the attributes are cost, reliability, availability, flexibility, expandability, reconfigurability, etc. They are very difficult to be quantified. Usually, the situation is further complicated by the fact that the system attributes are interdependent on each other and they may compete and interact with each other. The degree and manner in which they interact are greatly dependent on the architecture of the system and will be discussed in detail in the design phase. What is to be done by the designers at this step is to list and rank the system attributes according to their importance. In this way, the designers are forced to consider design tradeoffs early in the development process.

3.1.3 Process Definition

The process definition step accepts inputs from the requirement specification and attribute formulation step and identifies major functions to be performed. First the input stimulus and the required responses are characterized. This involves stating the form of the input and output signals. They may be mechanical, electrical, optical, etc. From this, the required I/O processes can be defined. For example, in an air traffic control system, it contains the radar control process, the graphic display process and the interactive I/O process.

In parallel to the formulation of the I/O processes, the functional requirements can be decomposed into data processing requirements, communication requirements, precedence constraints, etc. Similarly, the performance requirements can be decomposed into resource requirements, scheduling requirements, etc. Based on these requirements and the attributes defined previously, the information flow and control flow of the system can be modelled and analyzed to identify the major operations to be performed. From these analyses, the system processes required to perform the above functions can be defined. These process definitions state precisely the function of the processes, the resources required by the processes, the interaction between the processes and the test experiments to be performed. At this stage, the virtual system is formed and is ready for verification. For example, based on the processing requirement of the air traffic control system (Table 1) the storage and performance requirements can be generated (Table 2 and 3).

ANNUAL OPERATIONS, 1968-1995 (IN MILLIONS)			
	1968	1980	1995
Air Carrier	11	21	31
General Aviation	84	167	448
Military	33	34	40
Total:	128	222	519

Table 1 Processing Requirement of the Air Traffic Control System

STORAGE REQUIREMENT (WORDS)			
	Enroute	Terminal	National
Data Aquisition	74000	56000	
Command & Control	6300	80000	
Present ATC Functions	1150000	260000	
Flow Control			62200
Collision Avoidance System	99400	75000	
Monitor	100000	75000	
Total:	1486400	546000	62200

Table 2 Storage Requirement of the Air Traffic Control System

MAXIMUM INSTRUCTION EXECUTION RATE (MILLION INSTRUCTIONS/SEC.)		
	Largest Enroute	Largest Terminal
1980	22.5	19.9
1995	32.5	28.4
Max. Sizing Model	48.5	42.7

Table 3 Performance Requirement of the Air Traffic Control System

3.1.4 Verification of Requirements

In this step, the processes of the virtual system is verified to meet the original users' requirements. As the system is developed hierarchically, the specifications of one level are the requirements of the next level. To verify the correctness of the virtual system, we only have to verify the consistency between the specifications and requirements between consecutive levels. This simplifies the verification process a lot and if an analysable specification language is used, the consistency can be verified automatically. In addition to this, test experiments generated automatically [RAM76b] or formulated from the enviroment in the requirement elaboration step can be used to check the quality of the virtual system. If the tests are not acceptable, necessary changes are incorporated into the requirement specifications. The affected processes will be updated and the corrected system will be tested again.

3.2 Design Phase

The design phase starts with the defined processes which are the output of the requirement and specification phase. The major steps involved in the design phase are decomposition and partitioning, functional specification and finally verification (Figure 6). Basically, the design phase requires a provision to trace the system requirements through all levels of design, and a means of accessing trade-offs at the functional level and comparing design alternatives.

3.2.1 Decomposition and Partitioning

After the requirement process, a well defined, complete, consistent, unambiguous and testable set of system process specifications are produced. These system process specifications insure that the system behavior will be satisfactory to the customer and that the required system can plausibly be designed and implemented. Usually the specified system at this stage is so complex that it is very difficult if not impossible for the hardware designers to start implementing the system. In order for the design process to be manageable, it must be decomposed and partitioned in such a way that most decisions can be made locally, based on data available within a local area of the developing system specifications. To achieve this, the system is decomposed into progressively more detailed components which are then grouped into partitions (subsystems) to minimize the

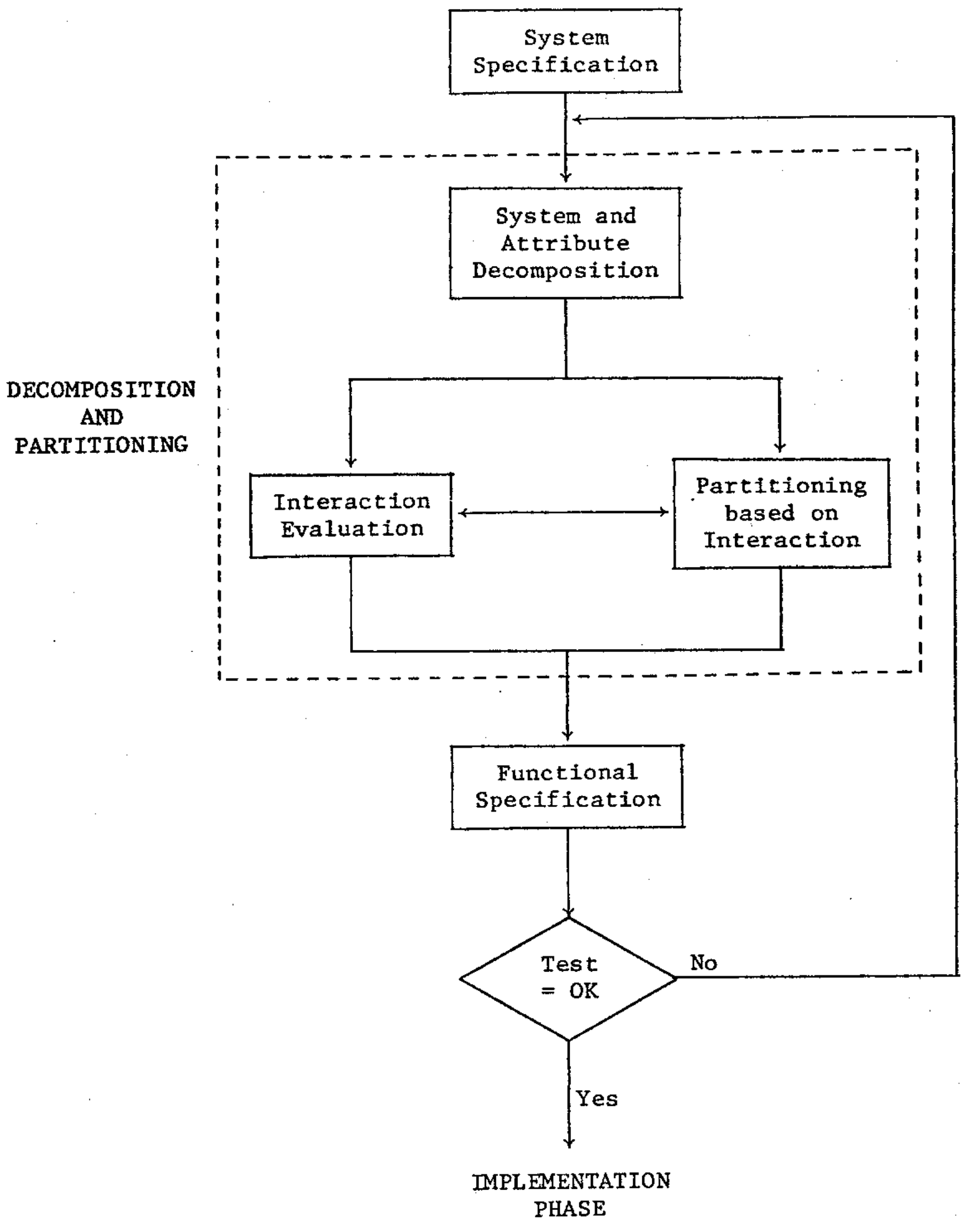


Figure 6. Design Phase

amount of interactions between partitions (Figure 7). The remaining steps of the development process can then independently (or nearly so) elaborate the design of each subsystem, maintaining the inter-subsystem interaction as design invariances. However, decomposition and partitioning must be carried out very carefully satisfying the following criteria:

(1) The decomposed (i.e. decomposed and partitioned) system must be well-defined -- they are consistent, complete, unambiguous and testable.

(2) The decomposed system must be capable of being integrated.

(3) The decomposed system must meet the resource allocation requirements, reliability requirements, performance requirements, etc.

(4) The decomposed system must satisfy the parametric logical specifications such that minor changes in the requirements would not require a redesign of the whole system.

(5) The decomposed system must be expandable so that future growth of the system can be easily incorporated.

In accomplishing the above criteria, a decomposition and partitioning procedure will be used to guide the designer to generate a satisfactory decomposition of the system. The objective here is not to produce an optimal

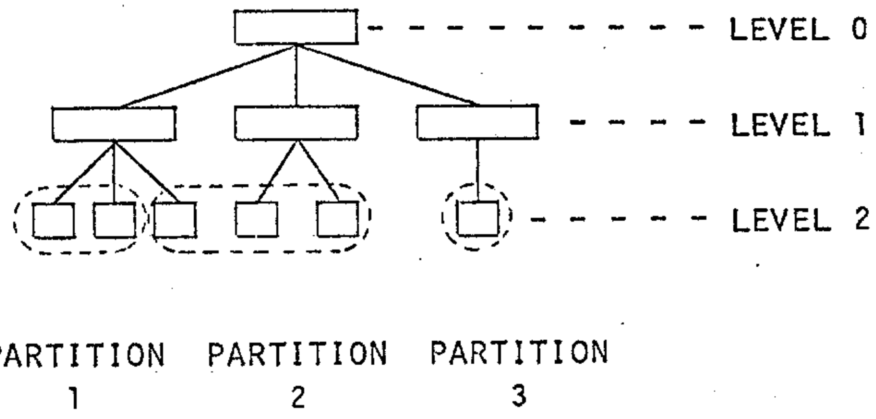


Figure 7 Reconfiguration and Partitioning

solution, but to develop a set of tools to help him to make his decisions. Most of the steps in the decomposition and partitioning process will be automated as to relieve the designer from tedious computations. However, some tradeoff decisions require human experience and interaction, and must be made by the designer. As a result, a close interaction of the designer is required.

(A) System and Attribute Decomposition

A large distributed system will have many processes that are only loosely coupled, for example, the communication subsystem and the data processing subsystem. These loose subsets of tightly coupled processes can easily be identified by the designer and can be used to partition the system into subsystems. Thus, the whole system is decomposed into smaller subsystems, each represents a different aspect of the original system.

After the preliminary decomposition, each subsystem can be decomposed further according to the data flow and control flow of the system. It can also be decomposed functionally into subfunctions. This step is greatly dependent on the characteristics of the system and the experience of the designers. After the system is decomposed, the attributes can be decomposed accordingly. For example, the tracking function in the air traffic control system is decomposed functionally into the radar control function, the position and velocity calculation functions, and the display function. The attributes of the tracking function can be passed to each of the decomposed function (Figure 8).

Another way that attributes can be decomposed is according to the architecture of the system. For example, in a distributed network, the reliability attribute can be decomposed into a payoff tree as shown in Figure 9. Other attributes can be decomposed similarly [MAR77]. Based on the payoff trees, the payoff interactions can be formulated (Figure 1Ø) and can be used to access tradeoff decisions made in the partitioning step.

(B) Partitioning based on Interaction

The objective of interaction partitioning is to reduce the amount of interactions between subsystems. This in turn reduces the complexity of the interfaces and the amount of communication between subsystems in a distributed system. The interaction partitioning process can be divided into two cooperating steps: (i) decision step which requires direct interaction from the designer, and (ii) solution finding step which can be highly automated (Figure 11). In the decision step, based on the processes definitions and their attributes, the designer assigns weights to the interaction between the processes. These weights represent the degree of coupling between the processes and will be a function of communication cost, amount of traffic flow, degree of synchronization, ranking of the attributes, etc. By using this weighting function, tightly coupled processes

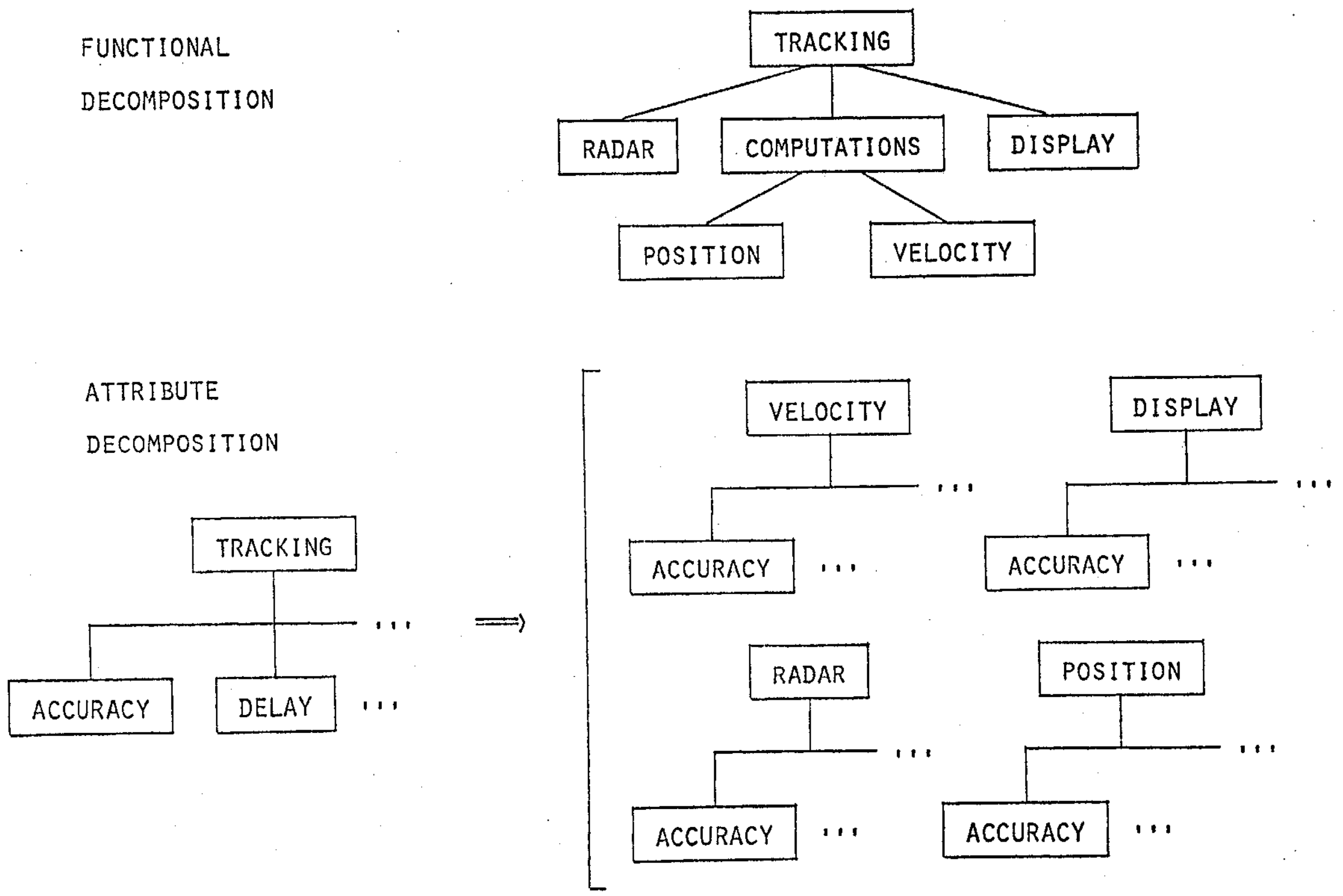


Figure 8 Functional and Attribute Decomposition

Example: Air Traffic Control System

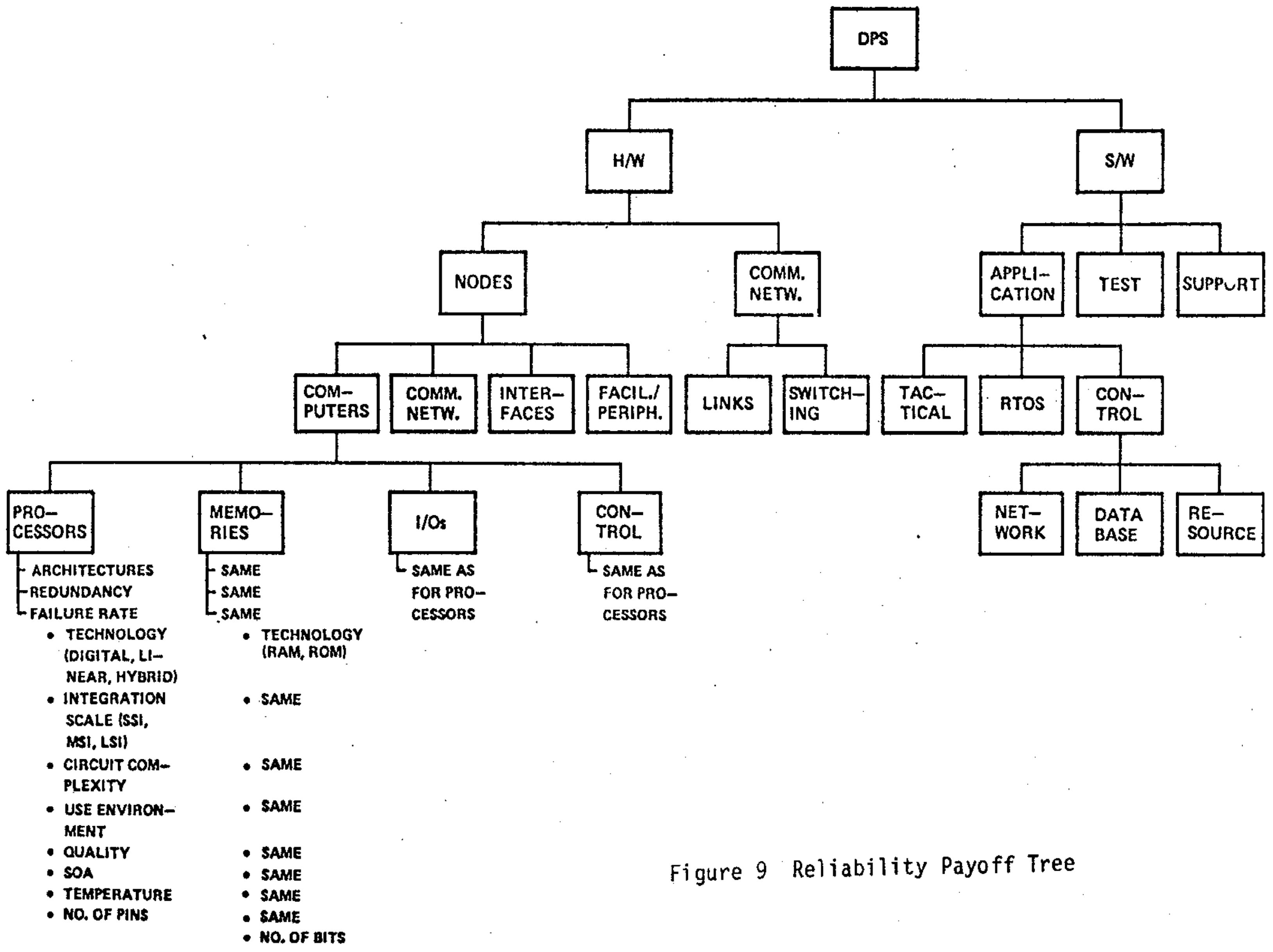


Figure 9 Reliability Payoff Tree

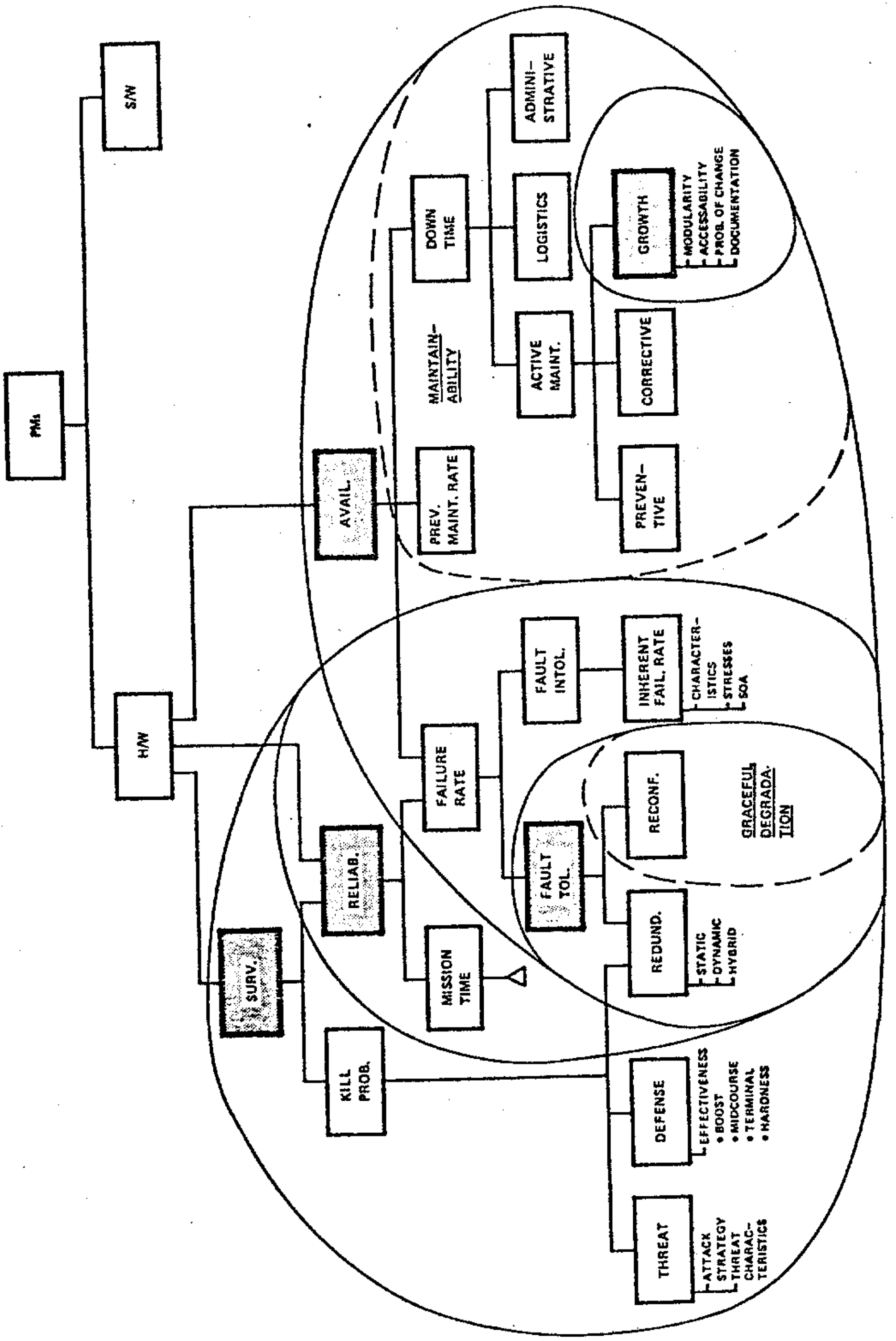


Figure 10 Payoff Measure Relationship

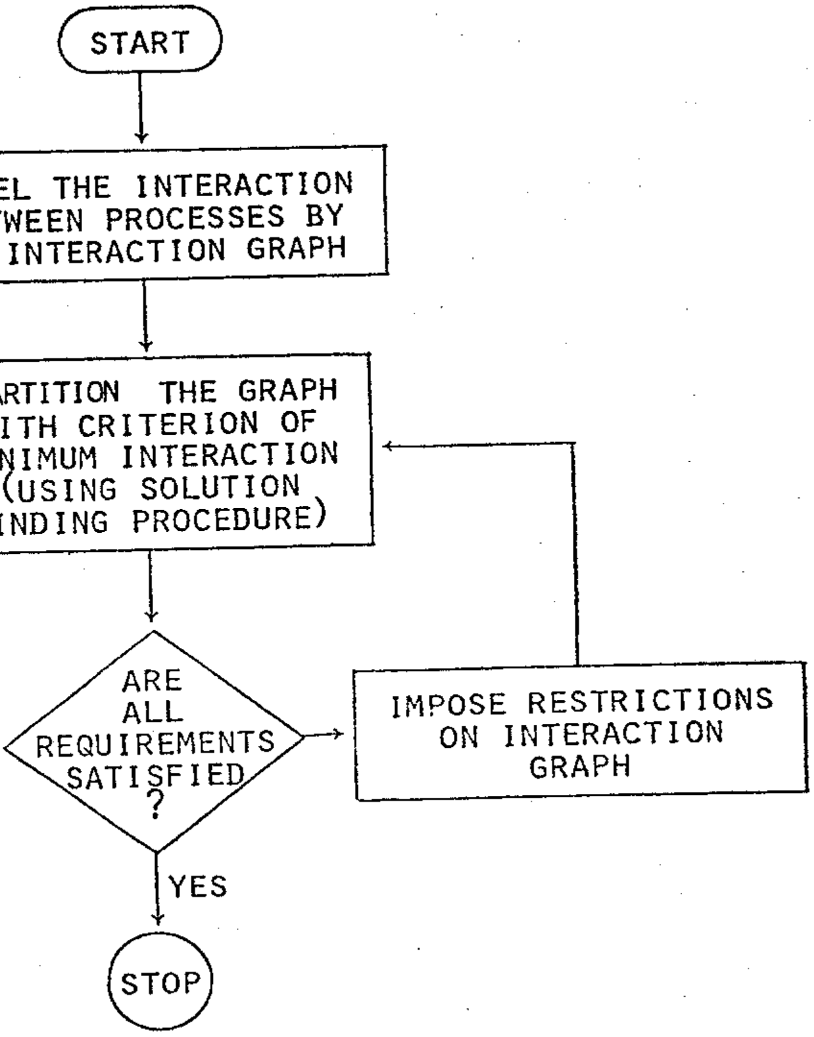

START
MODEL THE INTERACTION BETWEEN PROCESSES BY AN INTERACTION GRAPH
PARTITION THE GRAPH WITH CRITERION OF MINIMUM INTERACTION (USING SOLUTION FINDING PROCEDURE)
ARE ALL REQUIREMENTS SATISFIED ?
IMPOSE RESTRICTIONS ON INTERACTION GRAPH
YES
STOP

Figure 11 Flow Chart of Graphical Partitioning

will be assigned a high weight between them. Processes that should be in different modules (for example, due to reliability requirement) are assigned a very low weight between them. With this weight assigning function, we will then model the system by an interaction graph (Figure 12). In the interaction graph, nodes represent processes and the weight assigned to each arc corresponds to the amount of interaction of the two processes connected by the arc. In this way, the system is then partitioned into subsystems automatically by the solution finding step. After each iteration, the partitioned system is checked by the designer to determine whether all the requirements are satisfied. Discrepancies are identified and required changes are imposed onto the interaction graph which is analyzed by the solution finding step again. These two steps are iterated until all specifications are met.

Solution finding procedure: In this step, the system which is represented by an interaction graph is partitioned into loosely coupled subsystems such that the system requirements are satisfied. First, the interaction cut-tree (Figure 13) will be generated from the interaction graph by using the maximum flow minimum cut algorithm [FOR62]. It involves choosing two nodes arbitrarily in the interaction graph and finding the minimum cut-set of the two nodes by the maximum flow minimum cut algorithm. This minimum cut divides the interaction graph into two subgraphs. Two other nodes are then chosen arbitrarily in one subgraph and the

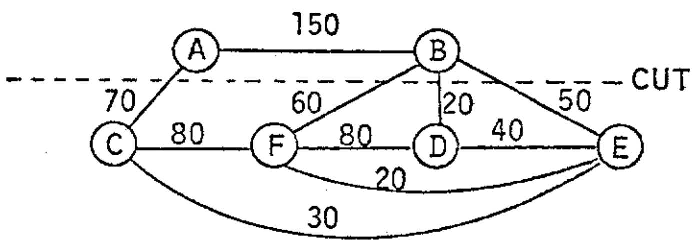


Figure 12 Interaction Graph

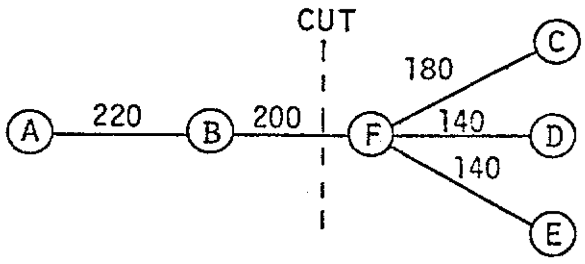


Figure 13 Interaction cut-tree

above procedure is repeated (with the other subgraphs being considered as macro nodes) until the interaction cut tree is generated. By this transformation, the interaction requirements between processes are represented very clearly by the interaction cut-tree. The minimal cut-set separating two nodes in the interaction graph is in one to one correspondence with the minimal cut-set of the corresponding two nodes in the interaction cut-tree. In addition to this, the values of the corresponding cut-sets in the two graphs are equal. For example, the minimal cut-set separating nodes a and f partitions the interaction graph and the interaction cut-tree identically into two subgraphs, {a,b} and {c,d,e,f}. The two cut sets both have values equal to 22. As a result, the decomposition can be performed on the interaction cut-tree rather than the interaction graph. This simplifies the computation greatly and displays clearly to the designer the condition of the system. The minimum cut can be identified easily in the cut-tree as it is the minimum weighted arc in the unique path connecting the two nodes.

After the interaction cut-tree has been generated, the system can be partitioned into loosely coupled subsystems such that the total weight of all the arcs in the cut-set is minimum. In addition to this, the partition should preserve the special configuration imposed by the requirements specified by the designer. For example, because of resource requirements, processes a and f, process a and d, and processes e and f have to be in different modules. In order for a and d to be on different modules, either arc (a,b), (b,f) or (f,d) has to be cut in the cut-tree. These requirements are then expressed in a table as shown in Table 4.

The decomposition can now be achieved by finding the set of arcs in Table 4 such that each row in the table contains at least on cross in the arcs chosen. This can be solved by the set covering algorithm [GAR72]. In our example, arcs (b,f) and (e,f) are chosen such that the interactions between subsystems is minimized (Figure 14). In general, this method will give a solution very close to the optimal solution and the computation complexity is very low when compared with that of generating the optimal solution.

3.2.2 Functional Specification

The next major step in the design phase is functional specification of the partitioned processes. This functional specification is different from the process specification described in the requirement specification phase. The objective of the process specification is to define the interactions of the the processes for the decomposition step. The objective of the functional specification here is to define the characteristics of the functions so to enable optimization in the functions to processors mapping.

In the functional specification, all the processes

ARCS SEPARATING	(A,B)	(B,F)	(C,F)	(D,F)	(E,F)
A,F	X	X			
A,D	X	X		X	
E,F					X

Table 4 Cut Table

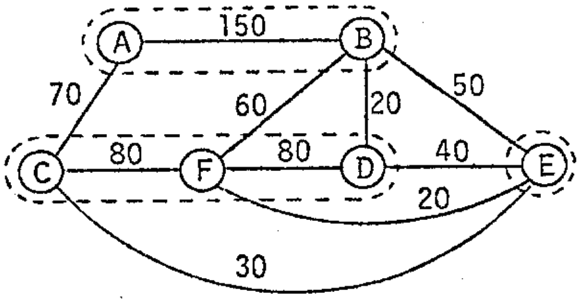


Figure 14 Decomposed System

in the same partition are considered as a single function. Similar to the process specification, the input and output relation, the precedent constraints and the interactions between different functions are determined. In addition to these, the characteristics of the function are defined. These include:

(i) types of operations to be performed -- matrix operations, floating point or integer operations, etc.

(ii) resource requirements -- storage requirement, processing power, etc.

(iii) speed requirements -- frequency and execution speed of the function.

Based on these information and the processors available, the functions are mapped onto the available processors. In [MUN69], an efficient mapping algorithm of two processors system is discussed. For the general case of n processors, no efficient algorithm is known at this time.

3.2.3 Verification of Design

In order to be able to verify the correctness of the design and to evaluate the effectiveness of the control, the system is represented in some abstract model. This abstract model should be capable of analysing the intercommunication, the synchronization, the performance and the coordination of the functions in a distributed system. However, no single model is powerful enough to have all the above features. The control flow of a distributed system can be modelled quite effectively by Petri net, UCLA graph model, E-net, etc. [PET77,GOS71,NOE73]. These models represent clearly the flow of information and control in a distributed system, especially those which exhibit asynchronous and concurrent properties. However, they do not provide any support for system data interpretation and therefore are unable to model data sensitive control flow. Various extensions have been proposed to include system data interpretation in the models, but the models become so complex after the extension that they are very difficult to analyse.

In order to predict and analyse the performance of the designed system, queueing models and simulation are oftenly used [FER78]. However, assumptions and abstractions usually have to be made to simplify the calculations and the results obtained may not be realistic. For example, when queueing models are being used, assumption like poisson arrivals and exponential service time are usually made. These assumptions may not be true especially when tasks are interdependent on each other. On the other other, simulations are almost expensive to run, and it is difficult to simulate all the possible features of the system. A unified model for the analysis of distributed systems is needed and much research should be done in this area.

3.3 Implementation, Evaluation and Validation

The implementation phase takes the virtual system and develops the system architecture. It then maps the system functions into either hardware or software functions. This step is greatly dependent on the technology, the architecture chosen and the physical constraints of the system. The final phase is the evaluation and validation of the system. This phase uses the bottom up validation approach. Both analytical modelling and simulation will be used. Because of the hierarchical decomposition, each subsystem to be analysed should be small and therefore complexity should be low. Mistakes or unfulfiled requirements found are traced back to the source of error. The system is then redesigned from that point.

In the following sections, the design issues of distributed data base systems are discussed. The distributed data base system is decomposed into three successive levels: DCS memory, nodal memory and data base machine. Design issues for each level are discussed in detail.

4. DISTRIBUTED DATA BASE SYSTEMS

In the remaining sections of the paper, we will discuss the design issues of a distributed data base system. A data base is a collection of stored operational data used by the application systems of some particular enterprise [DAT77], [FRY76], and a distributed data base (DDB) can be thought of as the data stored at different locations of a distributed system. It can be considered to exist only when data elements at multiple locations are interrelated and/or there is a need to access data stored at some locations from another location. The rise of DDB's is the result of two trends in data processing - growing need for large data base systems and recent advances in data communications. The prime concept of the generalized data base system is based on the definition of a data format to store the data and on a generalized data base management software for accessing the data. Due to the ever-increasing demand for on-line processing, there is a need for decomposing very large data bases into physically/geographically dispersed units and/or integrating existing data bases held in physically isolated nodes into a single, coherent data base that will be available to each of the distributed nodes. The DDB is not only an important application of distributed systems, but it is a good example as well as an important application of the methodology proposed in this paper.

There are three major logical components in a data base [BRA76]. First, there is the structured information or schema which describes the data structures and the validity criteria of the data. Second, there is the data itself. Finally, there are various programs or processes which control the operations of the data base. A level below the schema is another structural component, the sub-schema. It describes the data base as user applications see it. The control programs, together with the sub-schema, collectively form the Data Base Management System [FRY76], [BAC75]. The Data Base Management System allows data sharing among a community of users, while insuring the integrity of the data over time, and providing security against unauthorized access. It also provides the transparency of the data, in order to allow the data to be stored in different formats in different parts of the system. Finally, it provides an interface between the users and the system. The above logical components of a data base are built on a distributed computer memory system (DCS memory system), which consists of nodes interconnected by a network of channels. At each of these nodes, an uni-processor or a multiprocessor system exists and is supported by the storage sub-system whose function is to provide the nodal processor sub-system and users with fast retrievals and accesses of the stored data. The storage sub-system usually consists of a memory hierarchy which is divided into levels. These levels are made up of memory elements of varying speeds and the fastest level is interfaced to the processor sub-system. The file system on the memory hierarchy are usually broken into pages and they act as an unit of transfer among the levels. Further, intelligence have also been distributed to various levels of the hierarchy. One such design is the data base machine [HSI77].

The data base can be classified according to how these components are being put together. In [ASC74], two classifications are proposed, the first is based on the number of Data Base Management Systems in the network and the second is based on the centralization or decentralization of the file directory and the data. In [BOO76], the DDBs are classified into two structures, partitioned data bases and replicated data bases. A partitioned data base is one that has been decomposed into physically separate units distributed across multiple nodes of a computer network. The partitioning will normally be based on the distribution of access requirements. In a replicated data base, all or part of the data base is being replicated at multiple processing nodes. The amount of partitioning and replication will depend on the architecture of the distributed system, the amount of traffic anticipated and other requirements such as reliability, security, etc.

5. ISSUES IN DISTRIBUTED DATA BASE SYSTEMS

The issues associated with the design of a DDB can be classified according to users' point of view or from system designers' point of view. From users' point of view, the users are concerned with the type of organization and controls which can give efficient and reliable operations and can satisfy their requirements. The users usually do not relate very closely other factors such as technology and architecture in their considerations. On the other hand, from designers' point of view, the designers are more concerned with the architecture of the system and its limitations due to technology dependency. However, the issues considered from both views are not independent and must be investigated jointly in the design of a DDB. We have therefore taken an integrated approach and have classified these issues into four categories. The classification are diplayed in Figure 15.

5.1 Architectural Issues

5.1.1 DCS Memory Design

The DCS memory system is made up of nodal memories interconnected together through an interconnection network. There are many issues associated with the design of network memories in addition to the design issues of efficient nodal memories. Among these are: the selection of network topology; the selection of the channel type; the design of network control strategies; the design of communication

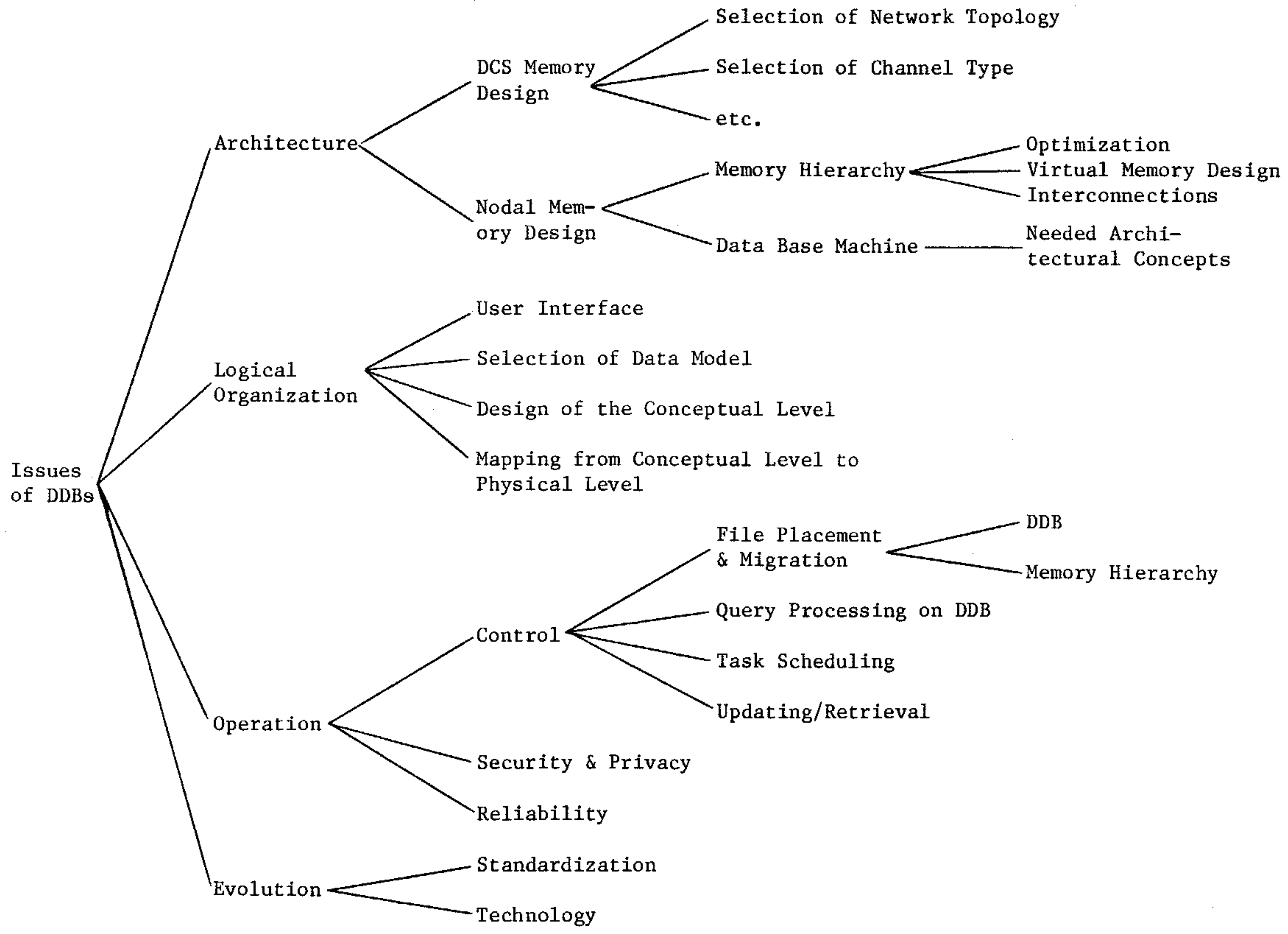


Figure 15. Classification of Issues in Distributed Data Base Systems

processors, etc. These issues have been examined carefully in [RAM76], and therefore will not be repeated here.

5.1.2 Nodal Memory Design

The nodal memory design is concerned with the design of the memory hierarchy and the data base machine which can efficiently support data base operations. Issues like the selection of the number of levels and the size of each level of the memory hierarchy; the utilization of gap-filler technology; the hardware design for supporting data base operations in a data base machine; the interconnection structure between memories and processors; etc. must be considered in the design.

5.2 Issues in Logical Organization

These issues are related to the user-system interface and can be classified as:

5.2.1 User Interface

An important issue in the design of a DDB is the design of user interface for efficient and easy users' access to the data base. This interface basically consists of the communication interface and a query language. It should help the users to find the exact location of the reqired data. The complexity of this interface depends on the required ease with which users access the data and it directly governs the design of communication processors.

5.2.2 Data Base Organization

A data base is generally organized in one of the data models: relational, hierarchical or network [DAT77]. There are other models like the binary association model and the external set model which are not quite popular. The efficiency of a DDB is very much dependent on the type of organization since it affects the storage organization, access mechanisms and the communication requirements. The criteria for designing and selecting a model has not yet been well understood or established, nor is it likely to be established in the near future. The designers of a DDB are therefore encountered with two decisions: which data model to utilize and how to structure the data for a chosen model [SIL76].

5.2.3 Design of the Conceptual Level

The conceptual level is a level of indirection between the external level which consists of different data models and language interfaces and the internal level which consists of the physically stored data. The conceptual level actually maps the users' view onto physical data and is intended to provide a solid and enduring foundation for the total operation of the DDB. Its design depends on how the data are being stored, the physical storage media, the number of different data models, the way that data are being distributed on the DCS and other user requirements. It is important to construct a conceptual schema at a suitable level of abstraction in the design stage [DAT77]. Many of the techniques in artifical intelligence have been applied successfully in this design.

5.3 Operational Issues

5.3.1 Control

These issues are concerned with the efficient, correct, reliable and secure operations of the data base. They can be classified as:

(1) File Placement and Migration

This relates to the distribution and migration of data base components, namely, schema, data and control programs on the DDB and the files and programs on a memory hierarchy with the objective of minimizing the overall storage, migration, updating and operational costs on the system. A related issue in DDB is how to decompose the request so that they can be processed in parallel by the different nodes of the DCS.

(2) Task Scheduling

This is to schedule jobs and requests on the DCS so that high parallelism and overlap can be achieved. This parallelism is important on a DCS because in order to attain high throughput, the parallel hardware and resource must be efficiently utilized. The control of task scheduling can be distributed or centralized. In distributed control, each node may act independently and coordinate with each other so that equilibium can be achieved. In centralized control, there is a primary node in which all scheduling control will be performed there. The decision of which is the better control mechanism depends very heavily on the interconnection structure and the communication overhead involved.

(3) Updating/Retrieval

In a DDB where several users share the same data, there are several problems associated with multiple accesses and updates. When several users try to access the common data, there would be interference among the users, and the communication protocol should be designed to minimize this interference. Another problem related to consistency arises when data elements with multiple copies at different locations are to be updated. Simple locking mechanisms cause excessive delays and may cause throughput degradation in the distributed system. Efficient updating schemes are needed and the architectures would be very much influenced by such schemes [ESW76].

5.3.2 Security and Privacy

Another important issue in the design of a DDB is security and privacy. Security refers to the protection of data against deliberate or accidental destruction, unauthorized access or modification of data. On the other hand, privacy refers to the right of an individual user to determine for himself what personal information to share with others as well as what information to receive from others. As the size of the data base increases, the threat to security and privacy increases. In addition, it is increasingly difficult to implement effective measures in a DDB. Additional techniques such as data encryption would influence the transmission efficiency and the communication mechanisms [BAD78], [DOW77].

5.3.3 Reliability

The determination of the needed hardware and data redundancy and the reconfiguration strategies is another major issue in the design of a DDB. Multiple copies of data base realm offer fast recovery; checkpointing of realms, dumping and journal rollback and roll-forward offer a slower but cheaper recovery. The effect of any recovery mechanism and reconfiguration strategy on the response time and the associated overhead must be weighed against the reliability requirements [KRI78].

5.4 Evolution

In order for the system to be able to adapt to new application requirements and technology advancements, Evolutionary measures must be designed into the system at the design stage. Two of the contradicting issues for evolution are:

5.4.1 Standardization

One of the major inhibiting factors in the development and evolution of DDBs is the lack of standardization in the areas of programming languages, user interface commands, data models, concurrency control mechanisms, hardware components (e.g. disks, tapes), data formats, network protocols, etc. Standardization of hardware and software components allow modular expansion of the system. On the other hand, with a highly evolving technology on hand, standardization may cause costly refitting later and may even hinder acceptance of new ideas.

5.4.2 Technology Dependence

Technology is one of the most important driving force for the success of a computer system. There are a variety of device manufacturing technologies [MOE78]. These different technologies are able to offer a variety of semiconductor memories with different access times and prices [THE78], [UPT78], [FET76]. In Table 5, the typical access time and power consumption for several semiconductor memory types are shown. Given these diverse types of memories available on the market, the designer must therefore decide at the design stage which one is the most suitable.

Table 5 Typical values for LSI Semiconductor RAMs (1978) (Price is shown for quantities of 100)

Memory Type	Access Time (nsec)	Power Consumption (mw)	Approx. Price (¢/bit)
16K MOS dynamic	125-300	400-600	0.30
4K NMOS dynamic	150-350	460	0.33
4K ECL static	30	1000	0.85
4K I^2L dynamic	120	450	0.59
4K TTL static	50-70	600-900	0.80-1.00
4K MOS static	55-170	30-500	0.61-0.92
1K CMOS static	150	4	1.02
1K TTL static	40-100	500-800	0.95
1K ECL static	35-60	500-800	1.30

On the other hand, magnetic device technologies have also improved significantly. With the improvement of disks, fixed-head disks, drums and tapes, the invention of the bubble memories [BOB71], and the Electron Beam Access Memories (EBAMs) [HUG75], it is now possible to provide inexpensive secondary and archival storage to the computer system.

With these evolving technologies, there are three significant impacts on the design of computers. First, new technologies add extra design alternatives to the designers. With more freedom in the alternatives, the designer may be able to design a system with improved performance and decreased system complexity. An example is shown by the recent developments of bubble memories, CCD memories and EBAMs which have emerged to fill the "access gap" between the two traditional memory technologies. The access gap is the region characterized by an access time between 10^{-6} sec. (MOS memories) and 10^{-3} sec. (fixed head magnetic disk) (see Fig. 16). Much time and effort is expended in computer systems in finding efficient ways to accomplish at minimum cost the necessary transfers of information across the access gap. With the utilization of "gap-filler" technologies, improved performance and less complex transfer algorithms can be envisioned. But before a successful utilization can be achieved, some questions have to be answered. These questions include where, in what sizes and what type of memories to use. The utilization of these memories is likely to change the current design and control algorithms. Therefore, the design of a DDB must be developed jointly with the availability of gap-filler technologies in mind.

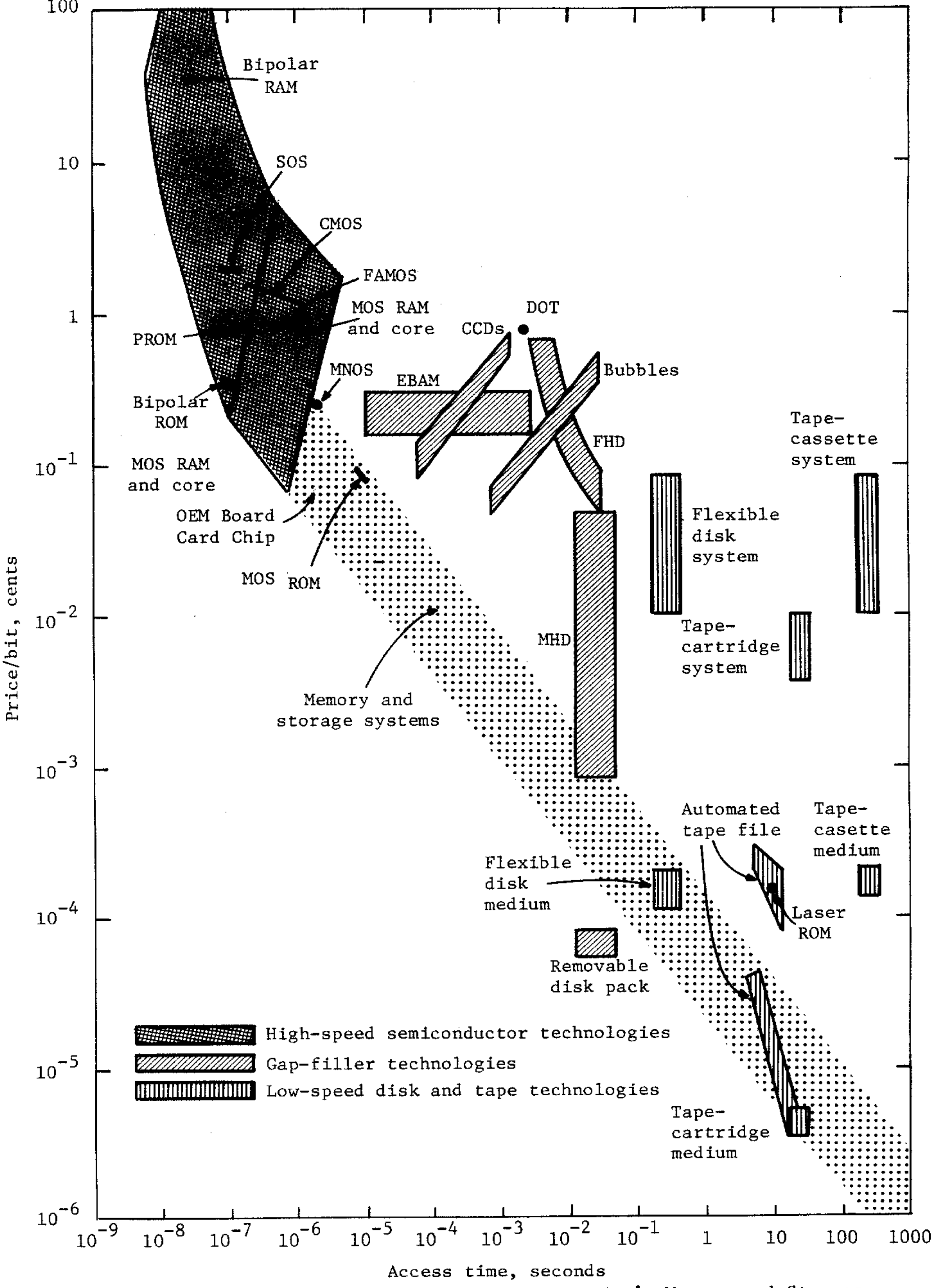


Figure 16 The Price-Performance Spectrum of Today's Memory and Storage Technologies for Large Systems [FET76]

Second, increasing logic on a chip allows the designer to incorporate more logical capabilities into the storage sub-system in addition to the storage capabilities. These logical capabilities include abilities to execute arithmetic operations like summation, finding averages, as well as logical operations like maximum/minimum searches, equality search, etc. The designer has to decide what logical capabilities are needed in the system and how they should be designed. The last impact of changing technologies on computer system design is that there is an increasing speed mismatch among the elements of the computer system. With the development of high speed processors such as the CRAY-1 and multi-processor system such as C.mmp, there is an increasing need of higher bandwidth from the supporting memory sub-system. In order to improve the bandwidths of memories, it is necessary to have intelligent architectural designs and efficient access algorithms for supporting retrieval operations in addition to the utilization of faster memory components. Special emphases should therefore be placed on the utilization of new technologies, the design of new memory architectures and the study of efficient access algorithms.

Evolving technology allows the users more freedom in specifying and operating the system. More stringent requirements can be specified and many of the system's functions can be designed in hardware. However, the dependence of the system on evolving technology usually is a severe constraint on the designer, and the evolutionary capabilities of a system depend very heavily on how well the designer can predict the future technologies.

We have outlined some of the issues in the design of a distributed system supporting a DDB. These issues are by no means complete and other issues, both design and operational, have to be considered. Alternative solutions to these issues provide the options to be decided upon in the design phase. In the next two sections, we will discuss some of these issues and their solutions in detail. The issues that we will discuss include the architectural issues and the operational issues. We have also integrated the discussion so that the operational issues are discussed with respect to the memory architecture, namely, nodal memory and DCS memory system. Issues relating to the logical organization and evolution are not studied and they are beyond the scope of this paper.

6. DCS MEMORY DESIGN

The memory system on a DCS is made up of nodal memories connected together by a network and communicates via the connected processors (Fig. 17a). Each node in the system, which consists of a set of processing elements and the supporting storage sub-system, may be active or passive. If the node is active, it acts as a requesting source and can access the memories at other nodes via the communication sub-system. Each of the active nodes in the system has the following functions in addition to the local file accesses.

(1) Remote access control

This module detects all remote access requests from this node and is responsible for processing them. When a remote request is detected, this module will look up the network directory, and assess the file status. If the file exists on the network and is accessable by the request, this request will then by transmitted.

(2) Local access control

This module is responsible for processing all remote requests received from other nodes in the network. It acts as a security filter and determines whether the file is accessable. If so, the local file is accessed and the data will be transmitted.

(3) Redundant file maintenance control

This module coordinates all the local and remote updates

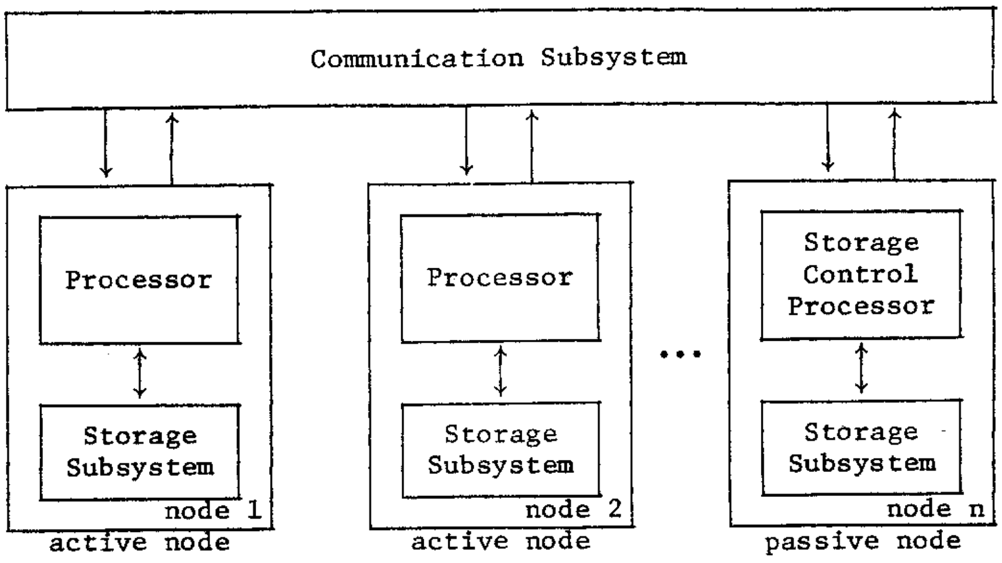
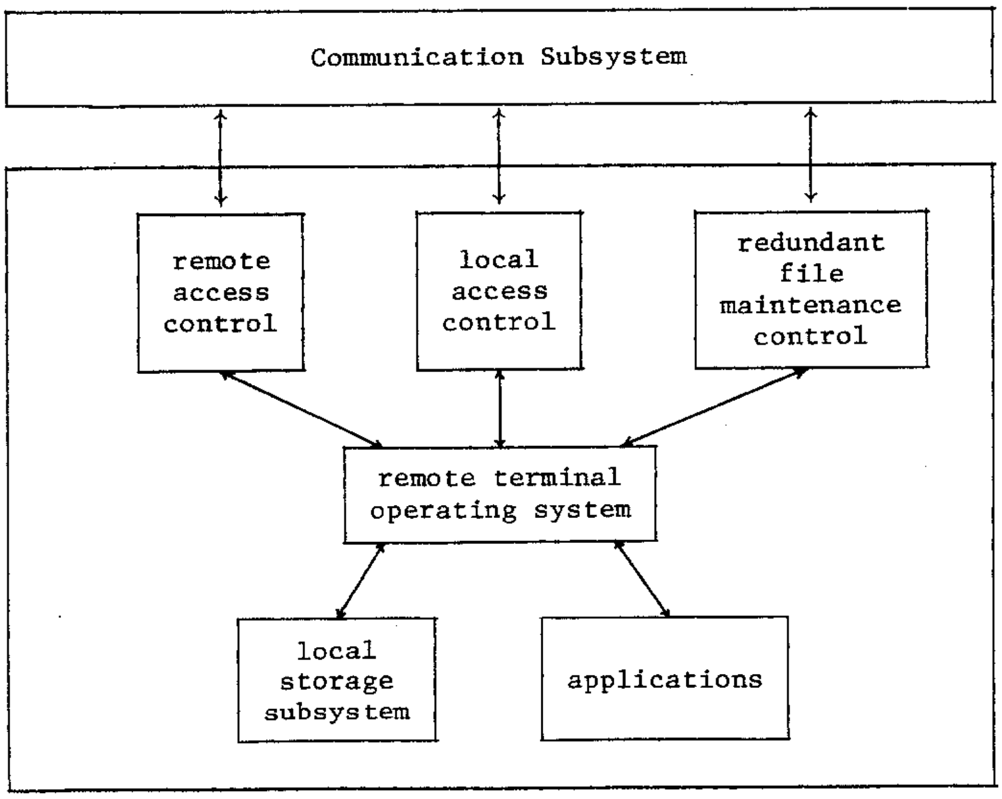


Figure 17(a) A DCS Memory System

Figure 17(b) Functional Design of an Active Node



at this node and the management of multiple copies of a file on the system. In coordinating updates, if the update is originated from a remote node, the status of the file is checked. In case that a conflict occurs and the data cannot be updated, a status message is being sent. On the other hand, if no conflict occurs, the file will be updated. If the update originates at this node, this module will look up the network directory and send out all the requested updates to all redundant copies on the system.

The relation of these modules in an active node is shown in Fig. 17b.

Due to the information explosion and the need for more stringent requirements, the design of an efficient coordination scheme for the local memories is a very critical problem in DCS. To indicate the amount of data processed, we shown in Table 6 the typical data base processing requirements for a ballistic missile defense system [DDP78], operating in a centralized environment. For this system to be distributed, several issues have to be resolved. Many of these issues are inherent in the DCS. Details of these issues have been presented in an earlier paper [RAM76] and will not be repeated here. We discuss in this section the operational issues of a DDB. These issues, which include the program/file placement/migration problem and the resilient multiple update problem, are related to the correct

Table 6 Typical Ballistic Missile Defense Data Base Processing Requirements in a Centralized Environment [DDP78]

Task	Number of independent tasks	20
Files	Number of dynamic files	117
	Local	30
	Global	87
	Dynamic file storage requirement	431K, 60 bits
	Local	26K, 60 bits
	Global	405K, 60 bits
	RTOS storage requirements	10K, 60 bits
Processing Environment	Processing speed	13.9 MIPS
	No. of reads/sec	7×10^6, 60 bits
	No. of writes/sec	3.4×10^6, 60 bits
	RTOS events/sec	24K

and efficient operations of a DDB. The issues concerned with the design of the nodal memories will be discussed in the next section.

6.1 Program/Data Placement/Migration Problem

The problem is defined as follows: given a number of computers that process common information files, how can one allocate the files so that the allocation yields minimum overall operating costs. This problem has been called the File Allocation Problem (FAP) [ESW74]. A more general problem is the Dynamic File Allocation Problem (DFAP) in which the files are allowed to migrate over time so as to adapt to changing access requirements. The FAP and the DFAP deals with the placement of files. On the other hand, the Query Processing Problem (QPP) deals with the partitioning of the queries. The query is an access request made by an user or a program in which one or more files have to be accessed. For a given distribution of files on a DCS and a query which is decomposable into sub-queries, the QPP is concerned with the order in which the sub-queries should be processed. Depending on the ways in which the sub-queries are processed, QPP can further be classified into Sequential Query Processing Problem (SQPP) and Parallel Query Processing Problem (PQPP). In SQPP, the sub-queries are processed in sequential order. Using the results produced by the processing of the previous sub-query, the processing of the present sub-query will produce some results to be used by the next sub-query in sequence. If the files used by the sub-queries are separated geographically, intermediate results have to be transferred over communication lines. The objective is to minimize the amount of communications required. In PQPP, the files to be used by the different sub-queries are transferred to a single location. The queries are then processed there and the results are sent back to the requesting location. The amount of communications required is usually greater than sequential query processing, but the response time is smaller because all the communications are done in parallel (it is assumed that the major overhead is in communications and not in processing). For a compromise between the amount of communications and the response time, a combination of sequential and parallel query processing should be used. The QPP has been studied in [WON76], [WON77]. The FAP and the QPP are not independent. The solution to the QPP depends on the assumption that the files have been distributed optimally. On the other hand, the solution to the FAP depends on the strategies of decomposing the queries. A general problem of file distribution and query processing is therefore a combination of the FAP and the QPP.

The major reason for allocating multiple copies of a file to certain part of the system at certain times and the unnecessariness of keeping a copy of every file at every node all the time is because users have localities of access in any time interval. At any particular time, a file may be used by a group of users and it will continue to be used by the same group for a certain length of time. For a particular user, the file that he wants to access may be available locally, in which case, he can access the file with very little cost. If the file is not located at the node that he is working, he would have to pay a cost in terms of delay in accessing the file and also introducing congestion in the DCS before he can make the access. Under this situation that we should consider moving a copy of the file to his node. Introducing a new copy would also increase the cost in terms of storage space and the additional overhead in locking and concurrency control. Therefore, the decision of whether to introduce a new copy of a file involves a balance of the cost between the two cases. The costs are a function of the topology of the system, the type of communication protocols used and most importantly, the extensiveness of usage at a particular node. The last factor is measured over a particular interval and is actually the area under the access frequency (versus time) curve. The access frequency for the future may not be known and therefore a priori predictions may have to be made in advance. The inaccuracy by which we make these a priori predictions can be a major source of error.

In Fig. 18a, the utility gained through migration is plotted against the staticity of the request patterns for different ratios of the cost of migration to the cost of making the access without migration. The staticity of the request pattern is the rate of change of the request pattern with time. Since it is difficult to measure this for a DCS, we can only classify it as dynamic, semi-dynamic, static, etc. We see that when the request pattern is static, no utility is gained through migration. However, when the request patterns change with time, we get higher and higher utility through migration. The curve is convex because of the increasing desirability to migrate the files as the request patterns become more dynamic. We also see that as the ratio of the costs is higher, the gain in utility is lower for the same staticity of request patterns. The relation with the ratio of costs and the staticity of the request patterns is further displayed in the three dimensional graph in Fig. 18b. From Fig. 18, we conclude migration of files may not be a suitable solution to all file distribution system. Only when the request patterns are dynamic and the costs of migration are low can we consider helpful to migrate the files.

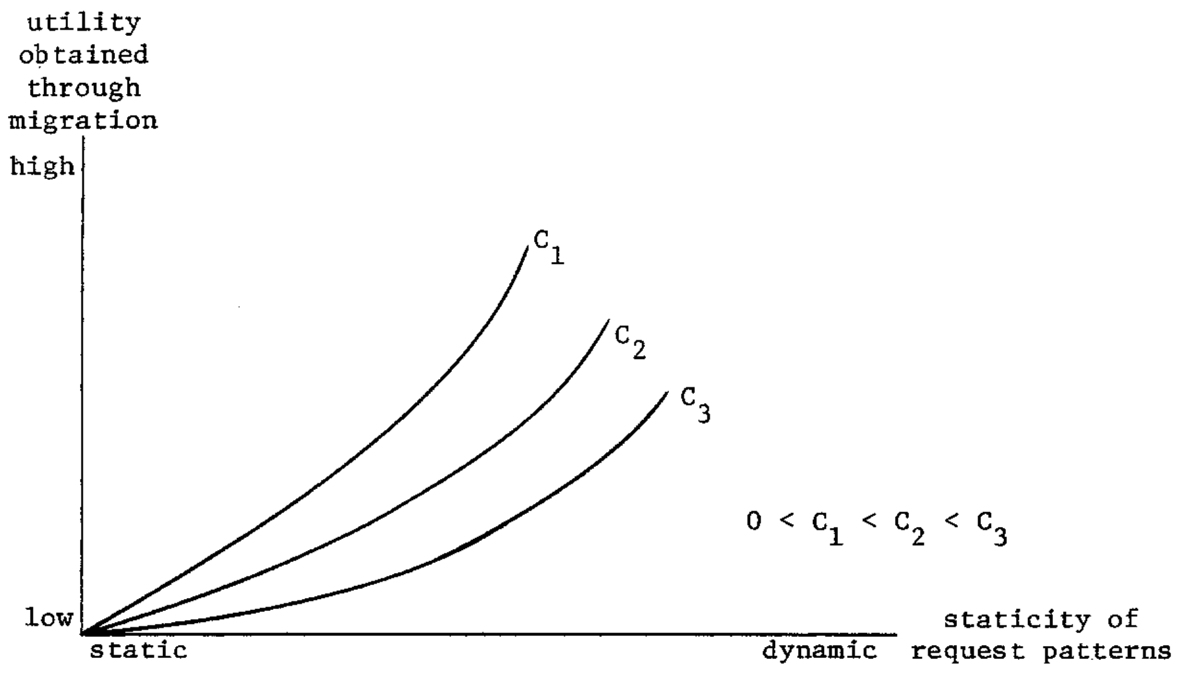


Figure 18(a) 2-dimensional diagram showing the utility obtained through migration for different ratios of (cost of access after migration + cost of migration)/(cost of making the access without migration).

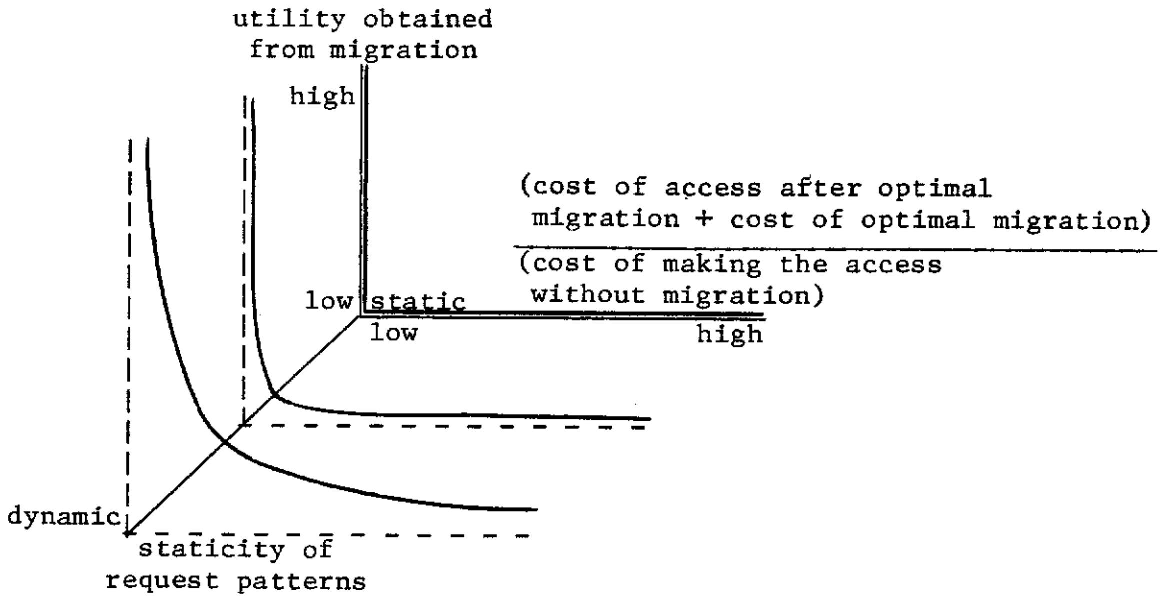


Figure 18(b) 3-dimensional diagram showing the utility obtained through migration.

The problem of file placement/migration can be classified in three dimensions as follows:

(1) The level of sharing

The entities that exist on a data base can be classified into objects and agents. An object refers to a file, a program, a subroutine or any piece of information considered as an unit. An agent is an object which accesses another object. An agent can therefore be an user, a program or the operating system itself. The objects in the data base can therefore be classified as agents and non-agents. The level of sharing refers to the degree of inter-dependence between the agents and the objects. When no sharing takes place, there is no allocation problem since each node can carry the objects that might be requested at that node. When there exists sharing, an object may be requested by several agents at different nodes, and we have to consider the alternatives of moving either the agents to the unique object or to place identical copies of the requested object at different nodes.

(2) The behavior of access patterns

The behavior of access patterns can be classified as static which does not change with time and dynamic which changes with time. Most real world situations fall into the latter category.

(3) The level of information available to each node

This refers to the extent by which changes in one part of the system is propogated to another node in the system. If every node has complete information of the system, then significant overhead is needed to propogate all the changes in status to other parts of the system. For example, when an user logs off from a terminal, it may be necessary to propogate this piece of information to other nodes so that the objects that this user was using can be released. However, in most cases, incomplete information about the status of the system is available at each node and it may be too expensive or impossible or unnecessary to propogate all changes in status to every part of the system.

Most of the previous studies on optimization is based on static distribution, that is, the allocation does not change with time. Some variation of dynamic distribution involves the application of static algorithms whenever need arises. These algorithms are very expensive to run in real time. A particular solution to this problem involving a 30 site network required about an hour on an IBM 360/91 computer [GRA77]. The difficulty in optimization is also exemplified in [SIC77]. Moreover, most of the algorithms are shown to be NP-complete[1] [ESW74,], [KAR72]. Although polynomial algorithms could exist for some special cases of the problem, e.g. the allocation of files in a two processor system [MUN69], their use in practical applications is very

[1] NP-complete problems is a class of problems for which there are no known optimal algorithms with a computation time which increases polynomially with the size of the problem. The computation times for all known optimal algorithms for this class of problem increase exponentially with problem size, i.e., if n represents the size of the problem, then the computation time goes up as k^n where $k>1$.

limited. This result suggests that the distributed system designer should focus his attention to efficient heuristics.

Heuristics for file distribution on a DDB are usually interactive algorithms. A feasible solution can be generated. Users or some decision algorithms then have to decide whether to improve the solution or not and how to improve it. The disadvantages of these types of algorithms are that they usually find a local optimum instead of a global optimum and the validation of the algorithm is very difficult. For most cases, the heuristics can be shown to perform satisfactorily for some example values, but its worst case behavior is very difficult to determine. Four of the most common heuristics are[2]:

(1) Hierarchical designs

This is a heuristic procedure in which attention is first restricted to the more important features of a system. In a file allocation problem in a DDB, attention can first be restricted to geographical regions. After analysis has been performed and the files have been distributed to different geographical regions, attention can be directed to the less important details such as allocating files within a geographical region or within a memory hierarchy. This stepwise refinement procedure can continue down many levels. At each level

[2] The first three heuristics are applied and controlled by the redundant file maintenance module. The last heuristic is applied at the interface between the applications and the network operating system (Fig. 17b).

of optimization, it is hoped that the effects on the optimization of the current level from the levels above and the levels below are very small. Nonetheless, iterations and design cycles may exist to refine the solution.

(2) Clustering algorithms

Clustering algorithms are horizontal design processes which have a similar objective as hierarchical algorithms, namely, to reduce the complexity of the analysis in a large system. Clustered file organization have been studied in centralized data bases in which related or similar records are grouped together into classes, or clusters of items in such a way that all items within a cluster are jointly retrievable, e.g. [SAL77]. Such algorithms are developed for single file searches. In a DDB, the files can be clustered according to criteria different from those of a centralized data base, e.g. geographical distribution of access behavior [LOO75, [LOO76]. If the usage of files can be clustered according to geographical regions, then we can design network of each region independent of each other.

(3) Add-drop algorithms

In applying this algorithm, a feasible distribution of files is first found. The total cost of the system can be improved by successive addition or deletion of file copies. When a feasible solution with a lower cost is found, it is adopted as a new starting solution and the process continues. Eventually, we come to a point in which addition or deletion does not reduce the cost. We therefore reach a local optimum. The whole procedure can be repeated with a different starting feasible distribution and several local optima can be obtained. By taking the minimum of all the local minima obtained, it is hoped that we can get very close to the global optimum [MAH76].

(4) Query decomposition

The approach using query decomposition has been developed for relational data bases. In this approach, optimization is performed on the processing of a single query originated at a node. The objective is to minimize the cost of data (relations) movements from one node to another. It is assumed that the major cost lies in the communication overhead and not in the local processing. This approach is proposed for the design of the centralized version of INGRES [WON76] and is extended to the design of SDD-1, a distributed data base [WON77].

The above techniques are by no means complete. The choice of which type of algorithms to use depends very heavily on the topology and the architecture of the DCS, and should be resolved by the designer at the design stage.

6.2 Resilient Concurrency Control

This area is concerned with the problem of providing control mechanisms that allows concurrent accesses to a DDB, such that each transaction, apart from timing, sees and proceeds as if it is served by a dedicated data base system. In particular, if a transaction is consistency preserving and terminates in finite time when runs alone, it should also do so when runs concurrently with other transactions. Further, since one of the requirements of a DDB is high reliability, the underlying concurrency control must also be robust to failures. In summary, the main objectives of a concurrency control mechanism is to preserve consistency of the data base and to resolve deadlocks when they occurs.

6.2.1 Requirements for the design of a concurrency control mechanism

The requirements for the design of a concurrency control mechanism are:

(1) Correctness

(a) Consistency

There are two types of consistency: mutual consistency, which states that all copies converge to the same state and would be identical should update activity ceases; and internal consistency, which states that each distinct copy of the data base must remain consistent within itself, as in a conventional centralized data base [ESW76], [BER77], [BAD78].

(b) Deadlock free

Another problem associated with synchronization is deadlock. Basically, a mechanism may choose to avoid or detect deadlock. Appropriate rollback procedure must be tailored for the specific data base enviornment. The proof that a system is deadlock free is usually not easy [BER77].

(2) Reliability

(a) Robustness

In order for the DDB to be reliable, the underlying concurrency control mechanism must be robust when a node fails or when the network is partitioned due to the failure of a set of nodes. It is important to incorporate robustness at the initial design stage.

(3) Cost-Effectiveness

(a) Efficiency

The final system must be efficient and satisfy the response time requirements. In order to improve the efficiency of the system, we can try to reduce the communication delays and the control overhead while on the other hand try to maximize the degree of concurrency.

(b) Low cost

The cost of concurrency consists of: communication cost, storage cost for control (lock) tables, cost of re-initiation when a query has to be re-initiated due to conflicts of access in the last initiation and lastly, the cost of developing and executing control algorithms. The cost of control algorithm can be reduced by using homogeneous solutions, that is, all nodes would use the same control algorithm. Besides easier to develop and maintain, this can greatly facilitates the proof of correctness [ELL76].

(4) Evolution

(a) Independence of network model

It is desirable that the control mechanism can function independently of the topology of the network. There may exists different routes from one point of the network to another. If the mechanism is independent of the topology, it would function correctly even though the order of information received is different from the order sent [ELL77].

(b) Modular growth

The control mechanism must be able to adapt to technology and design evolution.

In designing the concurrency control mechanism, the design should be balanced between performance and reliability. In some systems, this is the duty of the data base administrator. In other systems, the user can have the freedom of choosing the desired performance-reliability ratio for himself. For example, in System R, the user is allowed to have several levels of consistency and therefore he can trade consistency for efficiency [AST76].

6.2.2 Types of Controls

Concurrency control mechanisms can be classified according to whether it is centralized or distributed. When a distributed algorithm is used, we can further classify it according to whether primary sites[3] exist among multiple copies and the means of synchronization (e.g. locks, timestamps, etc.). As an example, Table 7 characterizes seven existing concurrency control methods. These mechanisms are believed to represent a typical sample of current research interests in concurrency control mechanisms.

Table 7 Classification of Seven Existing Concurrency Control Methods.

Reference	Controlling Sites	Primary	Synchronization Method
[MEN78]	centralized	-	Locking with fully redundant lock tables
[STO78]	distributed	yes	Locking with localized lock tables
[THO78]	distributed	no	Majority consensus on the validity of the base variables based on time stamp values. Each site maintains a localized table of pending requests.
[ELL77]	distributed	no	Conflict test. No lock tables. Each site remembers one and only one pending update (internal) request.
[ROS77]	distributed	no	Locking with localized lock tables
[ROT77]	distributed	no	A time stamp mechanism with varying degrees of synchronization. Each site maintains a table of classes of pre-analyzed transactions.
[ALS76]	distributed	yes	unspecified

[3] A primary site in a data base is a node at which all updates from user processes will be initiated there.

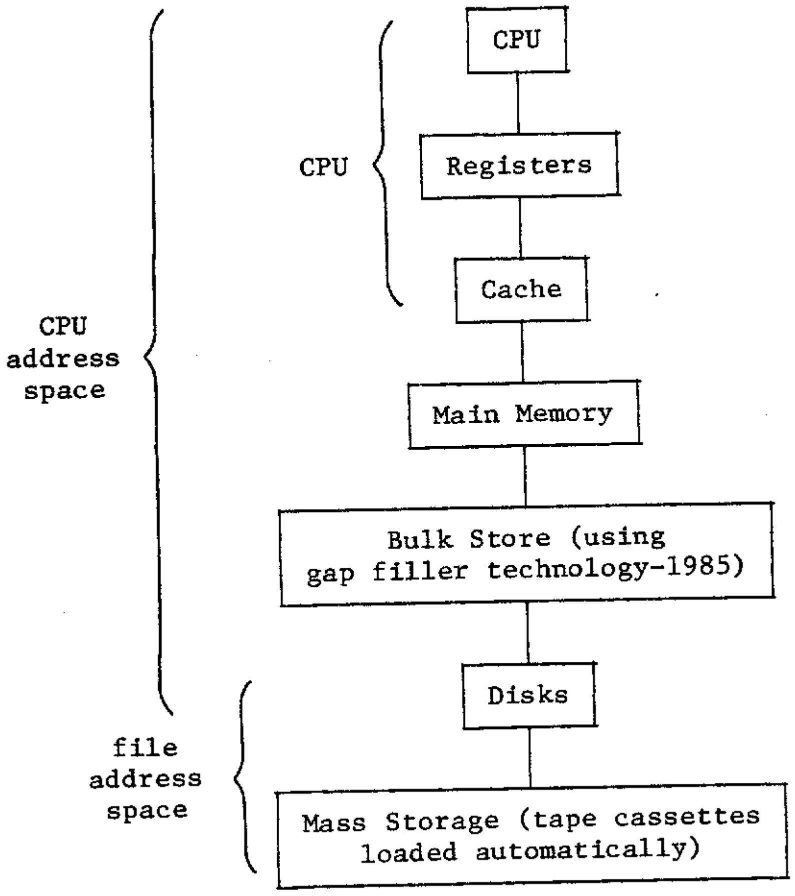
Design of concurrency control mechanisms for DDBs has been ad hoc in nature. This is probably due to the large number of factors one has to consider and it is difficult to identify all of them at the design stage. Further, efficient methods are usually difficult to find even when the parameters can be identified. In practical designs, most of the requirements presented cannot probably be satisfied. The final algorithm is usually tailored towards the particular network and applications.

7. NODAL MEMORY DESIGN

The storage sub-system of a data processing system at a node comprises a memory hierarchy that stores programs and data. Its spectrum ranges from bulk store and magnetic tape on one end to the fast register storage and cache memory in the CPU on the other end (Fig. 19). Further, there is an increasing tendency to distribute the processing of the CPU to the various levels of the storage sub-system. One successful implementation of this is the data base machine (Fig. 20) [HSI77]. The data base machine may be a separate member of the storage sub-system or it may represent a level of the memory hierarchy with extra intelligence. The optimization of performance of a storage sub-system is very important because the storage sub-system is very expensive and can be more than 50% of the total system cost [SCH78]. In this section, we will discuss some issues and solutions concerning the memory hierarchy and the data base machine.

7.1 Memory Hierarchy

It has been realized for a long time that the conflicting requirements for high performance and low cost storage sub-system at a node can be satisfied by a combination of expensive high performance devices with inexpensive low performance devices which results in a memory hierarchy. However, before a "good" memory hierarchy can be designed, many issues have to be resolved. Some of these issues have


CPU
Registers
Cache
Main Memory
Bulk Store (using gap filler technology-1985)
Disks
Mass Storage (tape cassettes loaded automatically)
CPU
CPU address space
file address space

Year 1975	Year 1985
20 MIPS	40 MIPS
500B	10^3B
10^4B	10^6B
10^6B	10^8B
0B	10^9B
10^9B	10^{11}B
10^{11}B	10^{13}B

Figure 19 Storage Hierarchy (with typical sizes shown for years 1975 and 1985)

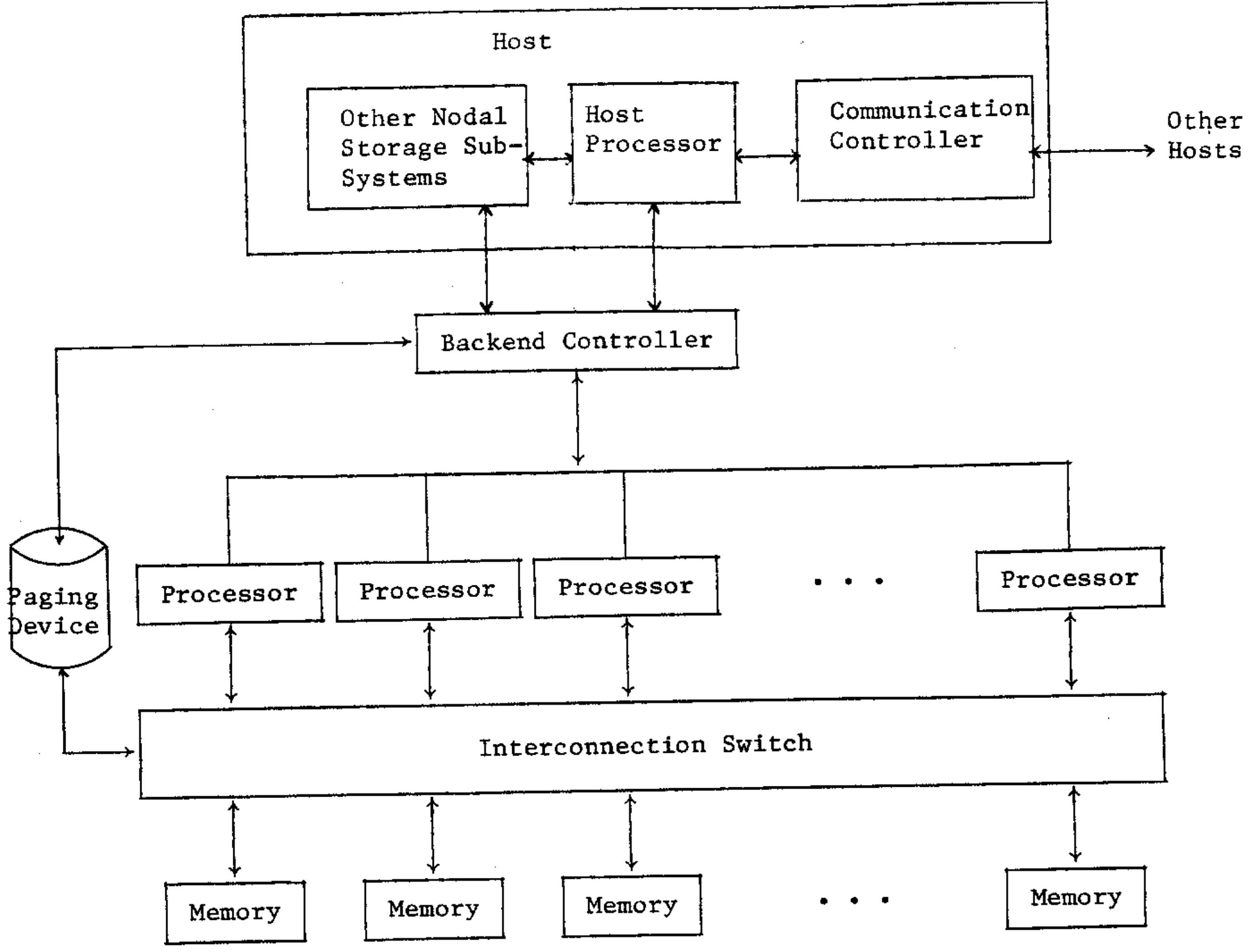


Figure 20 Architecture of a Data Base Machine

been studied extensively in the past. Others are currently under research. Among the most important issues are the following.

7.1.1 Architectural Issues

(1) Virtual Memory Support of an Automatic File Management System [TUE76], [POH75], [DEN70], [BAS70]

The problems studied in virtual memories include:

(a) The evaluation of replacement and retrieval algorithms [BEL66], [MUN74], [EAS77], [EAS78];

(b) The effects of page size and primary memory allotment on page fault rate [DEN68], [FAG76], [SMI76];

(c) The optimization of program distribution and block transfer size [ROB71], [HIR73], [BOY74], [FER76], [CHO77].

The success of a data base is very much dependent on the efficiency of the virtual memory. A file on a data base is likely to be large and cannot reside entirely in the main memory. The use of virtual memory can relieve the users from the laborious task of storage management. However, past researches have concentrated on machine instruction executions and data accesses on a virtual memory. On a data base, the characteristics of the accesses are usually different. The requests made on a data base access single or multiple files as an unit. This characteristics may change for different applications. We are therefore faced with the following research tasks related to a data base.

(a) Characterize file access patterns;

(b) Relate file access patterns to query access behavior in order to make projections;

(c) Study restructuring of a data base based on the access pattern so that the page fault rate can be reduced;

(d) Evaluate different file access and replacement algorithms;

(e) Study the relation between block transfer size and page size and the page fault rate;

(f) Study the effects of fragmentation due to the virtual memory on the data base.

(2) <u>Optimization of Memory Hierarchies</u>

In optimizing the memory hierarchy, we have to select the major components of the memory hierarchy and to investigate control algorithms when certain behavioral (statistical and application) information is known about the programs being run, under certain specifications. These specifications will have been defined after the requirement phase and include: (a) the effective access time of a request which is the time between the initiation of the request and the delivery of the result; (b) the throughput, which is the average number of words accessed by the CPU per unit time; (c) the word width, which defines the word size of the CPU and (d) the cost. Further, there are two application dependent requirements: (e) the request rate or a distribution of the request rate and (f) the correlation between the address of the current request and the address of the next request in sequence. This correlation will determine the sequentiality and locality of the memory accesses. Tradeoffs must be made so that the final design satisfies these requirements. The designer is faced with the decision of optimally selecting the following design alternatives.

(a) Physical parameters of the memory

These include the number of levels of the hierarchy and the size and the speed of each level. Traditionally, this problem falls into the class of NP-complete problems. The optimization of physical parameters of the memory hierarchy can be solved by integer programming [RAM70], [WAR76]. However, their use in practical situations is rather limited. The designer should therefore search for heuristics whose execution time is proportional to a polynomial of the problem size for the optimization.

(b) Interconnection mechanisms

This is the problem of designing the interconnection paths among the different levels of the memory hierarchy. A simple design would allow communications between adjacent levels. However, it is restricted when accessing or updating data not residing in an adjacent level of the hierarchy. The data to be accessed or updated have to pass through all the intermediate levels and therefore cause unnecessary traffic and overhead. However, it would be less expensive than a design which allows each level to communicate directly with a number of other levels. In the latter design, it is necessary to decide which are the essential paths. Accessing and updating algorithms are also more complex. Questions like whether the data is made available to all the connected levels when they are accessed or updated should be answered [SMI76b], [POH75]. The major advantages of the latter design are faster response time and less overhead in propogating the accesses and updates.

7.1.2 Operational Issues - Control Algorithms

(1) Scheduling algorithms

Memory requests made by the processor are scheduled so that some overall objective like the completion time is minimized. Investigations show that most of the optimal scheduling algorithms for ordering requests in memory hierarchy are NP-complete. However, in some special cases, polynomial algorithms exist and can be implemented. An example of this is shown in the scheduling of requests for an interleaved memory with a fixed buffer size and data path width [RAM78b]. It is found that an algorithm which initiate the memory module with the maximum number of queued requests will minimize the expected completion time of the requests. For most of the other NP-complete scheduling problems, it is necessary to find good approximation algorithms or heuristics.

(2) Record/file distribution and migration algorithms

This is a similar problem we have discussed earlier on file allocation in a DCS. The location of a record/file in a memory hierarchy is an important consideration in minimizing response time and meeting real-time constraints. Further, as the locality of accesses changes when the processor switches processes, it may be necessary to reorganize the files at different levels of the hierarchy. The objectives for our optimization are multi-folds. We would like to minimize the storage cost for a file in a level and transfer cost of migrating a file to another level while trying to make available as much as possible, all the files accessed by the processor currently and in the future on the fastest level of the hierarchy. Further, since the locality of execution changes from time to time and it is difficult to anticipate exactly what locality the system would change to in the future, the migration algorithms are necessarily dynamic, that is, the files in the system are reorganized whenever the locality changes.

So far, several studies have been made on the problem of file distribution and dynamic migration. The areas studied include replacement algorithms [MUN74], [FRA74]; characterizing the access behavior [STR77], [REV75]; the fragmentation problem [CON76], [WAR76] and distributing files on a distributed data base [CHU69], [CAS72], [LEV75], but very few studies have been made on an unified approach to study file placement and migration in a memory hierarchy. Research is therefore urgently needed in this aspect.

Most of the control algorithms for the memory hierarchy belong to the class of NP-complete algorithms. The designer therefore has to look for good heuristics which can be executed within the real time constraints. However, the evaluation of these heuristics are usually difficult. Evaluation methods and techniques are usually of three kinds, analytical techniques, simulations and approximation algorithms. Analytical techniques generally have to make some simplifying assumptions about the system parameters and the results obtained are usually not accurate. For example, when using queuing theory [KLE75] to evaluate the performance of a system, assumptions like poisson arrivals and exponential service time have to be made in order for the solution to be tractable. On the other hand, simulations are almost always expensive to run, and it is difficult to exhaust all the possible cases of the system. A third type of evaluation algorithms are approximation algorithms [WEI77]. There are two classes of these approximations, one guaranteeing a near-optimal solution always, and the other producing an optimal or near-optimal solution "almost everywhere". These types of algorithms are still in the research stage and a unifying approach in designing algorithms of this type is still lacking. The future trend is in the direction of investigating good approximation algorithms for memory hierarchies.

7.2 Data Base Machines (DBMs)

7.2.1 Objectives for DBMs

The objectives for the design of DBMs are:

(1) Parallelism

As the size of information processing grows, it becomes increasingly difficult to use an uni-processor to achieve the system's requirements. One alternative is to exploit the possibility of using multiple, less expensive and less powerful processors to form a conglomerate of parallel processors which can usually achieve the system's requirements in a more cost-effective way.

(2) Communication overhead

Processing on large file systems are often I/O bound. Many of the file operations are quite simple and a significant communication overhead is incurred in transferring the file to a level of the memory hierarchy where the processor can process it. By distributing the intelligence to the different levels of the memory hierarchy, the DBM can allow parallel processing with very little communication overhead.

(3) Hardware or firmware realization of data base functions
The complexity of data base system software is largely due to the processing of memory mapping operations. Memory mapping operations convert the file accesses by a query into actual memory addresses and must be highly optimized if they are to perform well. These operations often utilize complex data structure to achieve efficiency. On the other hand, data base software are divided into modules which perform specific tasks. For example, modules may exist for query parsing, directory access, directory processing, data retrieval and update, and data security. These modules usually have diverse capabilities and bottlenecks exist if these modules are executed on the same processor. The system performance is consequently degraded. The DBM solves the above two problems by eliminating the complex address mapping operations and utilizing hardware/firmware to replace the software. The query is transferred directly from the processor to the DBM without address mapping. Then hardware/firmware will procss the query and realize the data base functions.

7.2.2 Issues in the design of DBMs

Although DBMs have been successfully designed or implemented, e.g. Data Base Computer (DBC) [BAU76a], Context Addressed Segment Sequential Storage (CASSM) [LIP78], Relational Associative Processor (RAP) [OZK77], Rotating Associative Memory for Relational Data Base Applications (RARES) [LIN76], Datacomputer [MAR75], etc., the design of DBMs are still plagued by many issues.

(1) Architectural Issues

(a) Parallelism - kind and degree

The designer has to decide what functions can be processed in parallel and in what degree. These functions include address mapping operations, and the data base functions itself. With parallel processors, the designer also has to consider the efficient scheduling of tasks on these processors.

(b) Technology dependence

The design of the DBM must take into account the available technology. Using disk technologies, there is a large overhead in translating the signal available from a disk head to an useable form by the DBM. With bubble memory, it is very difficult to implement extra logic and hardware onto the same memory chip. A good candidate now is the CCD memory where the logic and the memory cells can be implemented together on the same chip. The design must also be able to adapt to future technologies as they are made available.

(2) Issues in Logical Organization

(a) Interface where and in what form:

The problem is to design a good interface between the DBM and the host processor. This interface may be implemented in hardware/firmware or software or a combinations of both. This interface translates queries from the host processor to data base functions processable by the DBM. Important questions like where to put this interface and its capabilities must be considered. Should it be a part of the host, or should it be a part of the DBM? Should the interface be able to access the memory hierarchy? How should the interconnection network be between the DBM and the storage sub-system? What type of language primitives should be used? These questions have to be considered carefully by the designer.

(b) Storage structure

The kind of storage structure is very important. If keyed accesses, that is, accessing data via a key, are allowed, then additional hardware capabilities like associative memory or extra pointers are necessary to support it. Further, questions like whether storage structure is dynamic in order to adapt to applications should also be considered.

(c) Backend primitives

The designer has to trade the availability of backend primitives (which include functions like sorting, file merging, etc.) with the cost and the difficulty of implementing it.

(3) Operational Issues

(a) Control algorithms

Because the memories of a DBM are usually slow (of the order of 100 μsec access time), much overlap and parallelism are necessary in order to achieve a high throughput. Control algorithms like scheduling and file placement and migration algorithms are therefore very important. The problem is very similar to that of designing control algorithms for the memory hierarchy which is discussed earlier and will not be elaborated here.

7.2.3 Key Architectural Concepts in the Successful Design of DBMs

To overcome the issues discussed, a few of the key architectural concepts must be followed.

(1) Associative memory

The associative memory is an important architectural component for the design of a DBM. Using the associative memory, logical operations like equality searches, extremum searches, and arithmetic operations like sum and difference can be performed on the records of the file associatively. In [RAM78a], a design for a sequential associative memory which is capable of performing logical operations like equality and extremum searches is shown (Fig. 21). Research is now under way to include arithmetic operations in the associative logic. This design can be generalized further where each word is made up of a memory block. Note that the associative logics are the blocks representing the parallel processors in Fig. 20.

(2) Function partitioning

The design of a DBM requires thorough analysis of the requirements and properly delegating (partitioning) the functions to various processors. The basic idea of partitioning and analyzing the application is to tailor the system architecture to suit the application and to utilize the system resources efficiently. In a DBM, the data base functions are partitioned so that each of the functions can be implemented in hardware/firmware. The decomposition and partitioning technique we have discussed earlier in Section 3 is useful in this context. In Fig. 22, we have shown an example where the user request handling are partitioned into different modules [PHI78]. In this functionally specialized system, the components are individually designed to adapt to their functions optimally. Since all the major functions and capabilities are well specified, estimation of the required processing power and memory capacity is much easier. Further, as each of the unit is a hardware component, their speeds can be designed to avoid

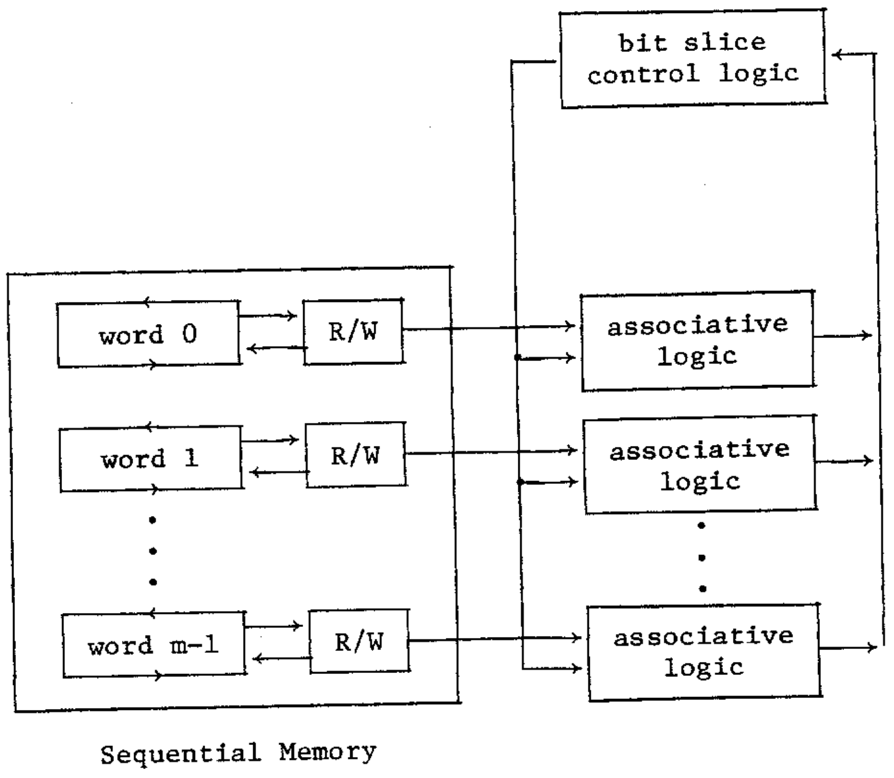


Figure 21 Associatve Logic for Associative Sequential Memory

bottlenecks. An example of such hardware partitioning is shown in the design of the Data Base Computer [BAU76b], [HSI76a], [HSI76b], where the design is partitioned into seven major functionally specialized components: the keyword transformation unit, the structure memory, the mass memory, the structure memory information processor, the index translation unit, the data base command and control processor and the security filter processor (Fig. 23).

(3) Lookaside buffers

The memories used in a DBM are usually slow, e.g. CCD memory, and the access time is rather long. To avoid unnecessary waiting in updating the files, a fast memory called the lookaside buffer is used to store these updates. Normal processing can resume after the lookaside buffer is updated because the data in the lookaside buffer now represents the most current copy. Updates can now be performed in the main memory without interference to the execution of other data base operations.

The concepts listed here are only a few of those that occur in the design. New concepts will be developed during the design and implementation phase for which the designer will have to make judicious decisions and innovative designs.

Finally we conclude that the DBM is a very powerful

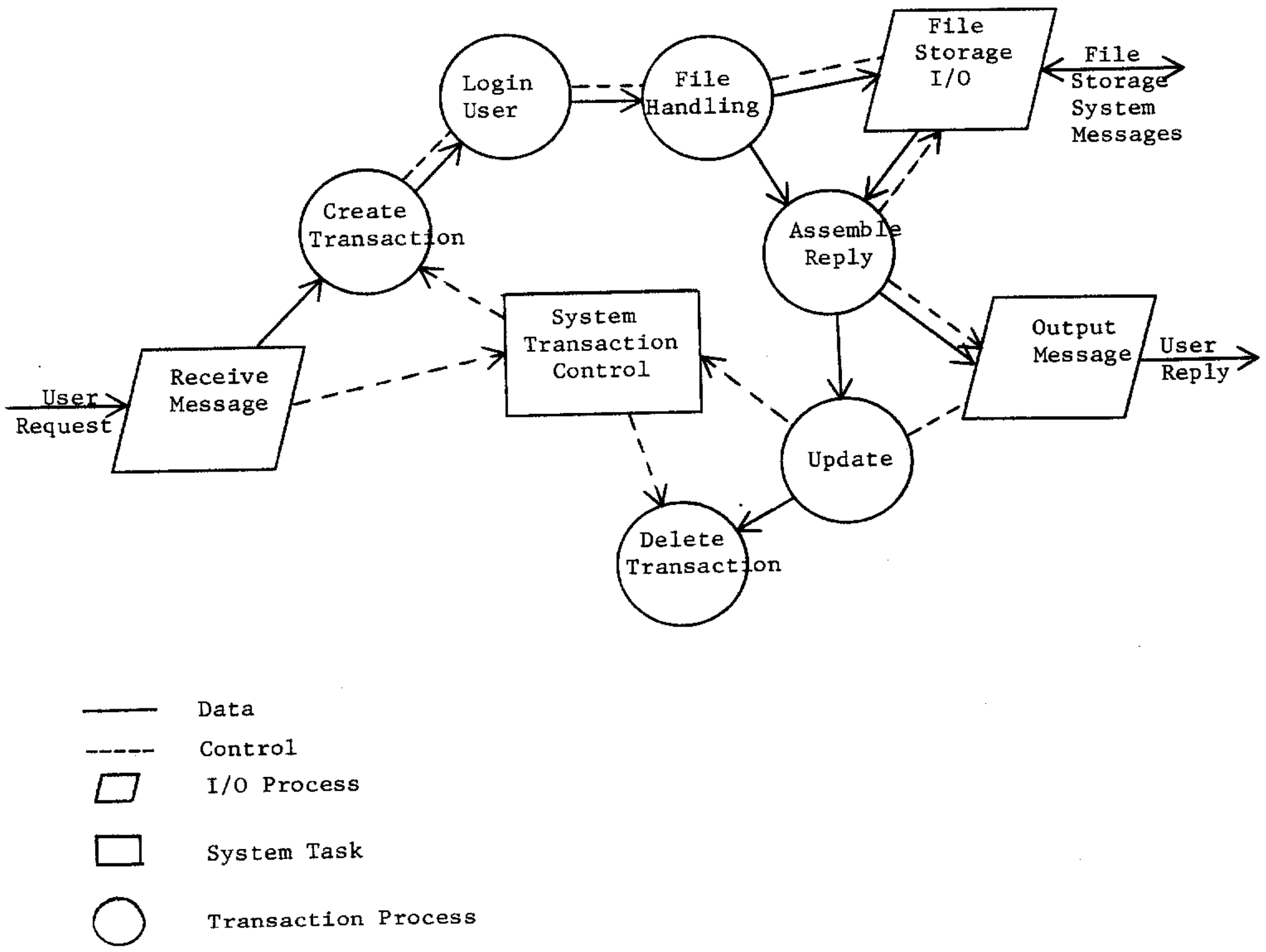
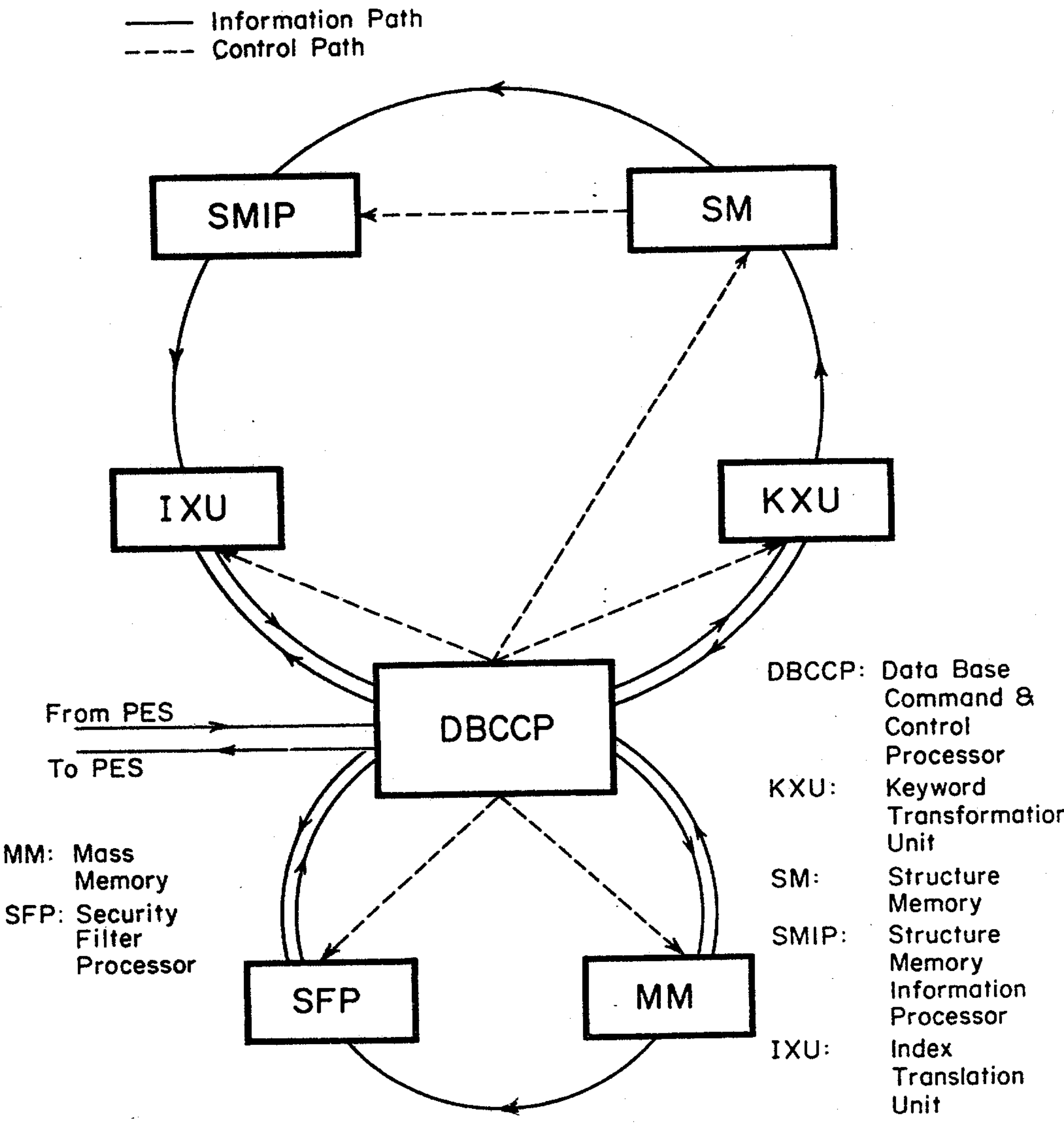


Figure 22 Function Partitioning for User File Request Handling [PHI78]



Information Path
Control Path
SMIP
SM
IXU
KXU
DBCCP
From PES
To PES
SFP
MM
MM: Mass Memory
SFP: Security Filter Processor
DBCCP: Data Base Command & Control Processor
KXU: Keyword Transformatior Unit
SM: Structure Memory
SMIP: Structure Memory Information Processor
IXU: Index Translation Unit

Figure 23 Architecture of Data Base Computer (DBC) [BAU76b]

architectural concept. We visualize that the nodal memory sub-system for a future computer system will consist of a memory hierarchy with each level being replaced by a data base machine or with distributed intelligence to handle local processing requirements. At the present time, only the bulk storage level (see Fig. 19) have been designed into a data base machine. However, with the availability of inexpensive processing units, distributed intelligence memory hierarchy will become a reality in the future.

8. CONCLUSION

This paper has discussed a systematic procedure for a distributed computer system and the design issues of a distributed data base. The main objective of the methodology is to develop reliable, effective, modifiable systems with low cost and lead time. The methodology uses the concept of abstraction, step-wise refinement and modularity to design a DCS. By following the design guidelines of the methodology, the system can be developed systematically in a hierarchical manner.

The design methodology is divided into four successive phases: (1) requirement and specification phase, (2) design phase, (3) implementation phase and (4) evaluation and validation phase. The first two phases have been explored in detail in section 3. In the requirement and specification phase, the requirements and the characteristics of the system are elaborated and expressed in a formal specification. The basic steps involved, the primitives to be specified, and the choice of specification language are discussed. In the design phase, the system specifications are analyzed to generate the virtual system. A systematic decomposition procedure is proposed. The design issues of a distributed data base are then investigated in sections 4, 5, 6 and 7. The issues that have been discussed are divided into the architectural and the logical aspects and the architectural aspect are examined in detail. The distributed data base is systematically divided into the network memory level and the nodal memory level. On the network memory level, two of the important operational issues, namely, the file allocation-query processing and the resilient concurrency control are examined. The nodal memory level is further divided into the design of virtual memory and data base machines. Issues concerning the design and optimization of the virtual memory are discussed. Some of the key architectural concepts for a data base machine have also been proposed. In particular, it is noticed that the data base machine concept can be extended to every level of the memory hierarchy. Lastly, the last two phases of the methodology are only outlined briefly. They are too technology and architecture dependent and are beyond the scope of this paper.

BIBLIOGRAPHY

[ALS76] Alsberg, P. A., and Day, J. D., "A Principle for Resilient Sharing of Distributed Resources", Proc. of 2nd Int'l Conf. on Software Engineering, 1976, pp. 562-570.

[ASC74] Aschim, F., "Data-Base Networks - An Overview", Management Information, Vol. 3, No. 1, 1974.

[AST76] Astrahan, M. M., et. al., "System R: A Relational Approach To Data Base Management", ACM Trans. on Data Base, Vol. 2, No. 2, June 1976.

[BAC75] Bachman, C., "Trends in Data Base Management", Proc. of AFIPS National Computer Conference, 1975, Vol. 44, AFIPS Press, Montvale, NJ, 1975, pp. 569-576.

[BAD78] Badal, D. Z., "Data Base System Integrity", Digest of Papers, Compcon Sp. 78, pp. 356-359.

[BAS70] Baskett, F., Browne, J. C., and Raike, W. M., "The Management of a Multi-level Non-paged Memory System", Spring Joint Computer Conference, 1970, pp. 459-465.

[BAU76a] Baum, R. I., Hsiao, D. K., "Data Base Computers - A Step Towards Data Utilities", IEEE Trans. on Comp., Vol. C-25, No. 12, Dec. 1976.

[BAU76b] Baum, R. I., Hsiao, D. K., and Kannan, K., "The Architecture of a Database Computer, Part I: Concepts and Capabilities", Ohio State University, Tech. Rep. OSU-CISRC-TR-76-1, 1976.

[BEL66] Belady, L. A., "A Study of Replacement Algorithms for a Virtual Storage Computer", IBM Sys. J., Vol. 5, 1966, pp. 78-101.

[BEL76] Bell, T. E. and Thayer, T. A., "Software Requirements: Are they Really a Problem?" Proceedings of the 2nd International Conference on Software Engineering, October, 1976.

[BEL77] Belady, L. A. and Lehman, M. M., "The Characteristics of Large Systems", IBM Research Report, RC6785, September 1977.

[BER77] Bernstein, P. A., et. al., "The SDD-1 Redundant Update Algorithm (The General Case)", Tech. Rep.

No. CCA-77-Ø9, Computer Corporation of America, 575, Technology Square, Cambridge, Mass, Ø2139.

[BOB71] Bobeck, A. H., and Scovil, H. E. D., "Magnetic Bubbles", Scientific American, Vol. 224, No. 6, pp. 78-9Ø, June 1971.

[BOE75] Boehm, B. W. and McClean, R. K., "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software", Proceedings of the International Conference of Reliable Software, 1975.

[BOO76] Booth, G. M., "Distributed Data Bases - Their Structure and Use", Infotech State of the Art Report on Distributed Systems, 1976.

[BOY74] Boyse, J. W., "Execution Characteristics of Programs in a Page-On-Demand System", CACM, Vol. 17, No. 4, April, 1974, pp. 192-196.

[BRA76] Bray, O. H., "Distributed Data Base Design Considerations", Trends and Applications, Computer Networks, 1976.

[CAS72] Casey, R. G., "Allocation of Copies of a File in an Information Network", AFIPS, SJCC, 1972, pp. 617-625.

[CHO77] Chow, W. M., and Chin, W. W., "A Program Behavior Model for Paging Systems", IBM Research Rep. RC-6452, 1977.

[CHU69] Chu, W. W., "Multiple File Allocation in a Multiple Computer System", IEEE Trans. on Computers, Vol. C-18, No. 1Ø, Oct. 1969, pp. 885-889.

[CON76] Considine, J., "A Computable Measure of Fragmentation for Direct Access Volumes", IBM Research Rep. RC-6241, Oct. 1976.

[DAT77] Date, C. J., "An Introduction to Data Base Systems", 2nd Edition, Addison-Wesley, 1977.

[DDP78] Distributed Data Processing Workshop, Stanford University, Feb. 15-17, 1978.

[DEN68] Denning, P. J., "The Working Set Model for Program Behavior", CACM, Vol. 11, May 1968, pp. 323-333.

[DEN7Ø] Denning, P. J., "Virtual Memory", Computer Surveys, Vol. 2, No. 3, Sept. 197Ø, pp. 62-97.

[DOW77] Downs, D., and Popek, G. J., "A Kernel Design for a Secure Data Base Management System", Proc. Very

Large Data Base, Oct. 1977, pp. 5Ø7-514.

[EAS77] Easton, M. C., and Bennett, B. T., "Transient Free Working Set Statistics", CACM, Vol. 2Ø, No. 2, Feb. 1977, pp. 93-99.

[EAS78] Easton, M. C., and Fagin, R., "Cold Start vs Warm Start Miss Ratios", CACM, to appear.

[ECK76] Eckert, J. P., "Thoughts on the History of Computing", Computer, 9, 12, Dec. 1976, pp. 58-65.

[ELL76] Ellis, C. A., "The Duplicate Data Base Problem", MIT Report RFC-112, May 1976.

[ELL77] Ellis, C. A., "A Robust Algorithm for Updating Duplicate Data Bases", Proc. of 2nd Berkeley Conf. on Distributed Data and Computer Networks, May 1977, pp. 146-161.

[ESW74] Eswaran, K. P., "Placement of Records in a File and File Allocation in a Computer Network", Information Processing, 74, IFIPS, North Holland Publishing Co., 1974.

[ESW76] Eswaran, K. P., et. al., "The Notions of Consistency and Predicate Locks in a Data Base System", CACM, Vol. 19, No. 11, Nov. 1976, pp. 624-633.

[FAG76] Fagin, R., and Easton, M. C., "The Independence of Miss Ratio on Page Size", JACM, Vol. 23, No. 1, Jan. 1976, pp. 128-146.

[FER76] Ferrari, D., and Law, E., "An Experiment in Program Restructuring for Performance Enhancement", Proc. of 2nd Int'l Conf. on Software Engineering, 1976.

[FER78] Ferrari, D., Computer Systems Performance Evaluation, Prentice-Hall, Inc., Englewood Cliffs, 1978.

[FET76] Feth, G. C., "Memories: Smaller, Faster, and Cheaper", IEEE Spectrum, June, 1976, pp. 36-43.

[FOR62] Ford, L. R. Jr. and Fulkerson, D. R., Flow in Network, Princeton University Press, Princeton, N. J., 1962.

[FRY76] Fry, J. P. and Sibley, E. H., "Evolution of Data Base Management Systems", Computer Surveys, Vol. 8, No. 1, March 1976, pp. 7-42.

[GAR72] Garfinkel, R. S. and Nemhauser, G. L., "Integer Programming", Wiley - Interscience, 1972.

[GOS71] Gostelow, K. P., "Flow of Control, Resource Allocation, and the Proper Termination of Programs", Ph. D. Dissertation, School of Engineering and Applied Sscience, University of California, Los Angeles, Dec. 1971.

[GRA77] Grapa, E., Belford, G. G., "Some Theorems to Aid in Solving the File Allocation Problem", CACM, Vol. 2Ø, No. 11, Nov. 1977, pp. 878-882.

[HAM76] Hamilton, M. and Zeldin, S., "Higher Order Software Methodology for Defining Software", IEEE Trans. on Software Engineering, Vol. SE-2, No. 1, March 1976.

[HIR73] Hirschberg, D. S., "A Class of Dynamic Memory Allocation Algorithms", CACM, Vol. 16, No. 1Ø, Oct. 1973, pp. 615-618.

[HSI76a] Hsiao, D. K., and Kannan, K., "The Architecture of a Database Computer, Part II: The Design of Structure Memory and its Related Processors", Ohio State University, OSU-CISRC-TR-76-2, 1976.

[HSI76b] Hsiao, D. K., and Kannan, K., "The Architecture of a Database Computer, Part III: The Design of the Mass Memory and its Related Components", Ohio State University, OSU-CISRC-TR-76-3, 1976.

[HSI77] Hsiao, D. K., and Madnick, S. E., "Database Machine Architecture in the Context of Information Technology Evolution", Proc. Very Large Data Base, Oct. 1977, pp. 63-84.

[HUG75] Hughes, W. C., et. al., "A Semiconductor Nonvolatile Electron Beam Accessed Mass Memory", Proc. IEEE, Vol. 63, No. 8, Aug. 1975, pp. 123Ø-124Ø.

[JEN75] Anderson, G. A. and Jensen, E. D., "Computer Interconnection Structures: Taxonomy, Characteristics and examples", Computing Surveys, Vol. 7, No. 4, December 1975.

[KAR72] Karp, R. M., "Reducibility among Combinatorial Problems", Complexity of Computer Computations, R. E. Miller and J. M. Thatcher eds., Plenum Press, New York, 1972, pp. 85-1Ø4.

[KLE75] Kleinrock, L., "Queuing Systems, Vol. I: Theory", John Wiley and Sons, 1975.

[KRI78] Krishnarao, T., A Systematic Design and Analysis of Reconfigurable Distributed Computer Systems", Ph.D. Dissertation, University of California, Berkeley, June 1978.

[LEH76] Lehman, M. M. and Parr, F. N., "Program Evolution and its impact on Software Engineering", Proceedings of the 2nd International Conference in Software Engineering, October 1976.

[LEV75] Levin, K. D., and Morgan, H. L., "Optimizing Distributed Data Bases - A Framework for Research", Proc. NCC, 1975, pp. 473-478.

[LIN76] Lin, C. S., et. al., "The Design of a Rotating Associative Memory for Relational Data Base Applications", ACM Trans. on Data Base, Vol. 1, No. 1, March, 1976.

[LIP78] Lipovski, G. J., "Architectural Features of CASSM: A Context Addressed Segment Sequential Memory", Proc. 5th Ann. Symp. on Comp. Arch., ACM-SIGARCH, pp. 31-38.

[LON77] Long, A. B. et al, "A Methodology for the Development and Validation of Critical Software for Nuclear Power Plants", Proceedings of the COMPSAC, November 1977.

[LOO75] Loomis, M. E. S., "Data Base Design: Object Distribution and Resource Constrained Task Scheduling", Ph.D. Dissertation, Comp. Sci. Dept., UCLA, 1975.

[LOO76] Loomis, M. E. S., and Popek, G. J., "A Model for Data Base Distribution", Symp. on Trends and Applications, 1976, Computer Networks, IEEE, 1976, pp. 162-169.

[MAH76] Mahmoud, S., Riordon, J. S., "Optimal Allocation of Resources in Distributed Information Networks", ACM Trans. on Data Base, Vol. 1, No. 1, March, 1976, pp. 66-78.

[MAR75] Marill, T., and Stern, D., "The Datacomputer - A Network Data Utility", AFIPS Conference Proceedings, 44, 1975, pp. 389-395.

[Mar77] Mariani, M. P., "The Use of Payoff Trees in the Distributed Data Processing Design Process", Distributed Data Processing Technology FY 77 Research Conference Publications, 1977.

[MEI77] Meindl, J. D., "Microelectronic Circuit Elements", Scientific American, Sept. 1977.

[MEN78] Menasce, D. A., et. al., "A Locking Protocol for Resource Coordination in Distributed Systems", Proc. 1978 ACM-SIGMOD Conf. on Management of Data, Austin, Texas, May 1978.

[MOE78] Moeller, A., "Fabrication Technology and Physical Fundamentals of Components used for Semiconductor Memories", Digital Memory and Storage, W. E. Proebster Ed., Braunschweig: Vieweg, 1978.

[MUN69] Muntz, R. R. and Coffman, E. G. Jr., "Optimal Preemptive Scheduling on Two-Processor Systems", IEEE Trans. on Comp., Vol. C-18, No. 11, Nov. 1969, pp. 1014-1020.

[MUN74] Muntz, R. R., et. al., "Stack Replacement Algorithms for Two Level Directly Addressable Paged Memories", SIAM J. on Computing, Vol. 3, No. 1, March, 1974, pp. 11-22.

[NOE73] Noe, J. D. and Nutt, G. J., "Macro E-Nets for Representation of Parallel Systems", IEEE Trans. on Computers, Vol. C-22, No. 8, Aug. 1973.

[OZK77] Ozkarahan, E. A., et. al., "Performance Evaluation of a Relational Associative Processor", ACM Trans. on Data Base, Vol. 2, No. 2, June 1977, pp. 175-195.

[PET77] Peterson, J. L., "Petri Nets", Computing Surveys, Vol. 9, No. 3, Sept. 1977.

[PHI78] Philips, B. J., "The File Store - A Distributed Modular File Handling System", Ph.D. Dissertation, University of California, Berkeley, June 1978.

[POH75] Pohm, A. V., "Cost/Performance Perspectives of Paging with Electronic and Electro-mechanical Backing Stores", Proc. of the IEEE, Vol. 63, No. 8, Aug. 1975, pp. 1123-1128.

[RAM70] Ramamoorthy, C. V., and Chandy, K. M., "Optimization of Memory Hierarchies in Multi-programmed Systems", JACM, Vol. 17, No. 3, July, 1970, pp. 426-445.

[RAM76a] Ramamoorthy, C. V., and Krishnarao, T., "The Design Issues in Distributed Computer Systems", Inftotech State of the Art Report on Distributed Systems, 1976, pp. 375-400.

[RAM76b] Ramamoorthy, C. V. and Ho, S. F., "Testing large software with Automated Software Evaluation Systems", IEEE Trans. on Software Engineering, March 1976.

[RAM78a] Ramamoorthy, C. V., Turner, J. L., and Wah, B. W., "A Design of a Fast Cellular Associative Memory for Ordered Retrieval", IEEE Trans. on Computers, Vol.

C-27, No. 9, Sept. 1978.

[RAM78b] Ramamoorthy, C. V., and Wah, B. W., "A Model of Interleaved Memory for a Pipelined Processor", Research Report, University of California, Berkeley, 1978.

[RAM78c] Ramamoorthy, C. V. and So, H. H., "Software Requirements and Specifications: Status and Perspectives", Memorandum No. UCB/ERL M78/44, University of California, Berkeley, June 1978.

[RAV76] Ravi, C. V., "The Structure and Characteristics of Distributed Systems", Proceedings of the 2nd International Conference on Software Engineering, October 1976.

[REV75] Revelle, R., "An Empirical Study of File Reference Patterns", IBM Research Report, RJ-1557, Apr. 1975.

[ROB71] Robson, J. M., "An Estimate of the Store Size necessary for Dynamic Storage Allocation", JACM, Vol. 18, No.3, July, 1971, pp. 416-423.

[ROS77a] Ross, D., "Structured Analysis (SA): A language for Communicating Ideas", IEEE Trans. on Software Engineering, Vol. SE-3, No. 1, Jan. 1977.

[ROS77b] Rosenkrantz, D. J., et. al., "A System Level Concurrency Control for Distributed Data Base System", Proc. of 2nd Berkeley Conf. on Distributed Data Management and Computer Networks, May 1977, pp. 132-145.

[ROT77] Rothnie, J. B., and Goodman, N., "A Survey of Research and Development in Distributed Data Base Management" Third Int'l Conf. on Very Large Data Bases, 1977, pp. 48-62.

[SAL77] Salton, G., and Bergmark, D., "Clustered File Generation and its Application to Computer Science Taxonomies", Information Processing 77, North Holland Publishing Co., pp. 441-445.

[SCH78] Schunemann, C., and Spruth, W. G., "Storage Hierarchy Technology and Organization", Digital Memory and Storage, W. E. Proebster ed., Braunschweig: Vieweg, 1978.

[SIB77] Sibley, E., "Standardization and Data Base Systems", Proc. Very Large Data Base, Oct. 1977, pp. 144-155.

[SIC77] Sickle, L. V., and Chandy, K. M., "Computational

Complexity of Network Design Algorithms", Information Processing 77, IFIPS, North Holland Publishing Co., 1977.

[SIL76] Siler, K. F., "A Stochastic Evaluation Model for Data Base Organization in Data Retrieval Systems", CACM, Vol. 19, No. 2, Feb. 1976, pp. 84-95.

[SMI76a] Smith, A. J., "A Modified Working Set Paging Algorithm", IEEE Trans. on Computers, Vol. C-25, No.9, Sept. 1976, pp. 907-914.

[SMI76b] Smith, A. J., "Characterizing the Storage Process and its Effects on the Update of Main Memory by Write-Through", Research Report, University of California, Berkeley, 1976.

[STO78] Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES", Mem. No. UCB/ERL M78/24, University of California, Berkeley, May, 78.

[STR77] Stritter, E., "File Migration", Stanford Linear Accelerator Center Report, SLAC-200, Jan. 1977.

[TEI77] Teichroew, D. and Hershey, E. A. III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Trans. on Software Engineering, Vol. SE-3, No. 1, Jan. 1977.

[THE78] Theis, D. J., "An Overview of Memory Technologies", Datamation, Jan. 1978, pp. 113-131.

[THO78] Thomas, R. H., "A Solution to the Concurrency Control Problem for Multiple Copy Data Bases", Digest of Papers, Compcon 78, March 78, pp. 56-62.

[TUE76] Tuel, W. G., "An Analysis of Buffer Paging in Virtual Storage Systems", IBM J. of Research and Development, Sept. 1976, pp. 518-520.

[UPT78] Upton, M., "Price/Performance Game Rules Change", Computer World, Jan. 23, 1978, p. 61.

[VIC76] Davis, C. G. and Vick, C. R., "The Software Development System", Proceedings of the 2nd International Conference on Software Engineering, 1976.

[WAN72] Wang, G. Y., "Advanced Computer Architecture for Large-Scale Real-Time Applications", Ph.D. Dissertatiion, University of Texas at Austin, June 1972.

[WAR76] Warren, H. S. Jr., "Static Main Storage Packing

Problems", IBM Research Report, RC-63Ø2, Nov. 1976.

[WEI77] Weide, B., "A Survey of Analysis Techniques for Discrete Algorithms", Computing Surveys, Vol. 9, No. 4, Dec. 1977.

[WON76] Wong, E., and Youssefi, K., "Decomposition - A Strategy for Query Processing", ACM Trans. on Data Base, Vol. 1, No. 3, Sept. 1976, pp. 223-241.

[WON77] Wong, E., "Restructuring Dispersed Data from SDD-1: A System for Distributed Data Bases", Comp. Corp. of America, Tech. Rep. CCA-77-Ø3, 1977.

[WYM76] Wymore, A. W., "System Engineering Methodology for Inter-Disciplinary Teams", Wiley - Interscience, 1976.

Benjamin W. Wah was born in Hong Kong on September 7, 1952. He received the B.S. and M.S. degrees in electrical engineering and computer science from the Columbia University, New York, NY, in 1974 and 1975, respectively, and the M.S. degree in computer science from the University of California, Berkeley, in 1976.

In 1973-74, he received the Sloan Foundation Fellowship, and in 1974-76 , he was on the University Fellowships. He is currently a Research Assistant in the Electronics Research Laboratory, University of California, Berkeley, where he is working toward the Ph.D. degree in computer science. His current research interests include distributed data bases, design methodology, distributed systems and computer networks, associative memory, memory hierarchy, scheduling theory, artifical intelligence, and parallel processing.