A design methodology for user oriented computer systems

by C. V. RAMAMOORTHY and G. S. HO
University of California
Berkeley, California

INTRODUCTION

A user oriented computer system is a computer system specifically designed to meet the users' requirements. With the advances in solid state technology and the emergence of microprocessors, system designers are able to design specialized systems to satisfy the users' requirements. This trend has led to an era of user oriented design. However, current approaches to the design of computer systems and their evaluation, unfortunately, are based primarily on experience and intuition. The specification, design, implementation, and evaluation of large embedded computer systems, such as the air traffic control systems, the patient monitoring systems and the ballistic missile defense systems (BMD), are very expensive, difficult to test adequately, slow to deploy, and difficult to adapt to changing requirements. The major cause of these problems 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 ever changing system environment. When the system application changes or the technology changes, the system has to be modified to adapt to the changes. 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 unstructureness (or the entropy) of the system increases enormously and leads to a regenerative, highly non-linear, increase in the effort and cost of system maintenance. 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 nuclear reactor control systems, 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 has to be paid (for example, large loss of life due to a nuclear explosion). However, these systems cannot be tested in its real operational environments. 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 an orderly way so that both validation and testing can be done efficiently.

In this paper, a systematic design and development methodology for user-oriented data processing systems (DPS's) is presented. The methodology will provide guidelines for the systematic design and construction of DPS's so that the users' requirements are satisfied if there exists a feasible design under the given constraints. However, it must be emphasized 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. The design methodology will provide design laws and analysis tools to help the designers in making design decisions, trade-offs and predicting the consequences. By following the guidelines given by the design methodology, the very complex design process will be simplified and we will be able to develop reliable, effective, modifiable systems with low costs and lead times.

Characteristics of the methodology

The philosophy behind the methodology is based on hierarchical modelling of a DPS. 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 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. In summary, the design methodology will:

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

Overview 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) users' require-
ments 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 for up to 85 percent of requirement errors. In order to avoid the above mistakes, the concept of a closed system is used. The users and their environment are considered together during the requirement and specification process. Requirement elaboration is done jointly in the environment and system models. By doing this, more complete and consistent requirements can be produced.

The design phase starts with the requirement specifications and finishes when the system specifications are produced. The objective is to organize and optimize the system in a well-formed structure. It involves a hierarchy of decompositions 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 amount of interactions and to satisfy the performance and reliability constraints of the system. After each decomposition and partition 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 come 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 modeling and simulation. Mistakes or unfulfilled 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 feedback path from each development process back to the previous one as indicated in Figure 2.

DESIGN METHODOLOGY

The design methodology is aimed at satisfying a broad spectrum of data processing applications including real-time applications. Its primary objective is to develop reliable, effective, modifiable systems with low cost and lead time. In the following sections, the requirement and specification phase and the design phase of the methodology will be discussed in detail. The implementation and validation phases are too technology and architecture dependent and are beyond the scope of this paper.

REQUIREMENT AND SPECIFICATION PHASE

The requirement and specification phase starts with informally specified users' needs and elaborates on them to generate formal system requirement specifications. These specifications are used for two purposes: (1) as a problem definition for the design process, (2) as a means against which an implementation can be validated. The success in the development of the data processing system greatly depends on the correct interpretation of the requirements in the specification phase. However, these requirement specifications usually suffer from many ills: they are often designs rather than a statement of need; they are usually incomplete and are expressed in an ambiguous language (English); they are difficult to verify and hence are often incorrect, conflicting within themselves (inconsistencies); and they are difficult to test and if one wants to modify them, it is difficult to locate and accurately modify all affected areas. In order to rectify the above problems, a systematic procedure is used to generate correct, consistent, complete, traceable, design-free and feasible requirements. Formal definitions of the above terms can be found in Reference 6.

In this methodology, the requirement and specification phase consists of four major steps, Figure 3: (i) requirement elaboration, (ii) requirement specification and attribute formulation, (iii) process definition, and (iv) verification of requirements.

![Figure 3—Requirement and specification phase](from the collection of the Computer History Museum (www.computerhistory.org))
Requirement elaboration

The requirement elaboration step can be considered as a problem understanding stage. The requirement engineers investigate in great detail the users' needs and develop clear and precise requirements of the system. In this step, the closed system approach is chosen. In a closed system, the functional, logical or precedence relationships. For example, problem understanding stage. The requirement engineers and precise requirements of the system. In this step, the engineers have fully understood the operations of the closed system, the data processing requirements and the attributes of the computer system can be derived from the behavior of the computer system. These overall system requirements can then be decomposed into finer detail according to their functional, logical or precedence relationships. For example, the BMD management system can be decomposed functionally as shown in Figure 4. Test experiments can also be generated from the behavior of the environment to verify the proper operation of the designed system. In this way, more complete and consistent requirements can be generated.

Requirement specification and attribute formulation

From the required behavior of the system, the users' original objectives can be formally expressed in the system requirements and system attributes. 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 preferences of the users. The generation of these system requirements and attributes is the requirement specification and attribute formulation step. As pointed out previously, one of the greatest problems in requirement specifications is the misinterpretation of the original system requirements. A plausible solution is to use dual specification teams to develop the system specifications from the requirements independently as in the development of critical real-time software for nuclear power plants. The two specifications are then compared and discrepancies are resolved to the satisfaction of both teams. By this dual specification approach, most of the errors due to ambiguities and misinterpretations can be corrected before they can propagate into the next phase.

Requirement specification

The system requirements can be broken down into two categories: (i) functional requirements, and (ii) performance requirements.

Functional requirements specify the input (stimulus) and output (response) relations of the system. These input to output mappings can be expressed rigorously in mathematical formulas or less formally in a specification language. Mathematical formulation defines explicitly the ranges of input and output domains and exhibits the required system functions by tabulation or a set of mathematical expressions. This allows formal consistency and correctness proofs of the system. However, this method is greatly limited by the ability of the requirement engineers in formulating the mathematical functions. In real-world situations, the problems are usually so complex that pure mathematical formulation is 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 an efficient and effective medium for defining the functional requirements. Several specification languages have been developed previously. This paper does not intend to develop a new specification language. We will choose a specification language and express the functional requirements in it. In choosing the specification language, the constructability and comprehensibility of the language must be evaluated carefully. The specification language must be able to express the functional requirements efficiently and be easily understood by the customers and the requirement engineers. It must be able to specify the system requirements unambiguously 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. Finally, it should be backed up by a specification data base management system and powerful graphical supports to provide easy and efficient access to the designed system.

Performance requirements specify the functional effectiveness of the system. They include the input and output rates, response time, accuracy etc. There is essentially no notion of completeness as far as performance requirements are concerned. The requirement engineers have to work closely with the users to ensure that all the important performance aspects of the system are captured in the specifications. At this point, it can be noted that there are still some uncertainties and vagueness in the methodology. However, these are unavoidable as design in general is a wicked problem. There is no stopping rule and definite test to the solution of a wicked problem. One stops only because one has run out of resources, patience, etc. Therefore, the best
we can do is try to provide guidelines in the development process to improve the quality of the product.

Attribute formulation

Attributes represent the customers' preference on the designed system, e.g., cost, reliability, availability, flexibility, expandability, reconfigurability etc. They are the evaluation criteria used to determine which characteristics make one system desirable over another system even though they may both meet the requirements. Usually, the system attributes are interdependent on each other and they may compete and interact with each other. In order to clearly specify the preferences of the customers, the system attributes are weighted or ranked according to the customers' preferences. They are then expressed in a utility function or in a payoff tree. Utility function has been used extensively in economics and in business management. It measures formally the degree of satisfaction of the customers in terms of the utility of the system. Once the utility function of a system has been formulated, the optimal system configuration can be determined by optimizing its utility. However, due to the inherent complexity of a large computer system and the high interdependency of its attributes, the formulation of its utility function can be very difficult. In such cases, the less formal approach of using payoff tree is preferred. It involves first generating the payoff trees of the system attributes. For example, the reliability payoff tree is shown in Figure 5. The payoff measure interrelationship can then be generated, Figure 6. These payoff trees show clearly to the designers the tradeoffs involved in the attributes. Based on these payoff trees, trade-off decisions in later stages of the development process can be facilitated. This approach is being used in the development of the BMD system.

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 and their locations of occurrence. 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, and the interaction between the processes. At this stage, the virtual system is formed and is ready for verification. For example, in the formulation of a tangent function, it can be decomposed functionally into a sine function, a cosine function and a division operation, Figure 7a. The attributes of the function (like accuracy, delay, output range, cost, etc.) can then be extended to become the attributes of the component functions, Figure 7b. In this manner, the virtual system is formed.

**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 or formulated from the environment 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.
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 process decomposition and partitioning, functional specification and finally verification, Figure 8. Basically, the design phase requires a provision to trace the system requirements through all levels of design, and a means of assessing trade-offs at the functional level and comparing design alternatives.

In the process partitioning phase the designer will decompose the system into a network of interacting tasks. The motivation for such partitioning is to increase modularity and testability and to decrease the interferences between processes. This partitioning is based on the philosophy of hierarchical decomposition of the system into progressively more detailed networks of processes at different levels of abstraction. The hierarchical approach permits partial ordering and allows the designer to isolate interacting and non-interacting parts. The partitioning criteria and approaches depend on the level at which the partitioning is carried out and upon the nature of the logical aspects involved, for example functional decomposition, algorithmic decomposition and attributes decomposition. Several techniques can be adopted to perform partitioning at various levels. Our approach to this problem of partitioning is discussed in the following subsections.

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 in such a way that most decisions can be made locally, based on data available within a local area of the developing system specifications.

The decomposition process can be accomplished by identifying the tightly coupled processes of the system and factoring the system into subsystems with respect to this criterion. The remaining steps of the development process can then independently (or nearly so) elaborate the design of each subsystem, maintaining the inter-subsystem interactions as design invariances. However, the decomposition must be carried out very carefully satisfying the following criteria:

1. The decomposed system must be well-defined—they are consistent, complete, unambiguous and testable.
2. The decomposed system must be capable of being integrated.
The decomposed system must meet the resource allocation requirements, reliability requirements, performance requirements, etc.

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.

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

In accomplishing the above criteria, a decomposition 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 solution, but to develop a set of tools to help him to make his decisions. Most of the steps in the decomposition process will be automated 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 in the decomposition process is required.

Preliminary decomposition

A large system will have many processes that are only loosely coupled, for example, the communication process and the data processing process. 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 representing some aspects of the original system from a different point of view. The number of such decomposed subsystems is application-dependent and sensitive to the identification of loosely coupled subsets of processes. By this preliminary decomposition, the system is decomposed into a structure according to its requirements and is ready for further analyses.

Decomposition based on interaction

The objective of interaction decomposition 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. The interaction decomposition process can be divided into two cooperating steps: (i) decision process which requires direct interaction from the designer, and (ii) solution finding process which can be highly automated, Figure 9. In the decision process, the designer looks at the requirement specifications and assigns weights to the interaction between processes. These weights 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 will be assigned a high weight. Processes that should be in different modules (for example, due to reliability requirement) are assigned a very low weight. With this weight assigning function, we will then model the system by an interaction graph, Figure 10. 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 can then be decomposed into subsystems automatically by the solution finding process. After the decomposition, the decomposed system is then checked by the designer to determine whether all the requirements are satisfied. The discrepancies are identified and the required changes are imposed onto the interaction graph which is analyzed by the solution finding step again. These two steps are iterated until all the 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 will be generated from the interaction graph by using the max. flow

From the collection of the Computer History Museum (www.computerhistory.org)
min. cut algorithm. It involves choosing two nodes arbitrarily in the interaction graph and finding the minimum cut set of the two nodes by the max. flow min. cut algorithm. This minimum cut divides the interaction graph into two subgraphs. Two other nodes are then chosen arbitrarily in one subgraph and the 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, Figure 11. The minimal cut-set separating two nodes in the interaction graph is one to one corresponding to the minimal cut-set of the same 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 1.

The decomposition can now be achieved by finding the set of arcs in Table 1 such that each row in the table contains at least one cross in the arcs chosen. This can be solved by the set covering algorithm. In our example, arcs (b,f) and (e,f) are chosen such that the interactions between subsystems are minimized, Figure 12. 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.

**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 processes for the decomposition step. The objective of the functional specification here is to define the characteristics of the functions so as to enable optimization in the functions to processors mapping.

In the functional specification, all the processes in the same partition are considered as a single function. Similar to the process specification, the input and output relation, the precedence 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 Reference 21, an efficient mapping algorithm of two processors system are discussed. For the general case of n processors, no efficient algorithm is known at this time and more research should be done in this area.

**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 must be modelled in some abstract model. This abstract...
model should be amenable to analyze the intercommunication, the synchronization, the performance and the coordination of the functions. In this methodology, the Petri net model is chosen.

Petri nets display clearly the flow of information and control in systems, especially those which exhibit asynchronous and concurrent properties. However, in order to model the time constraints of real-time systems, the Petri net model is extended to include the execution time aspect of a system. In the following sections, the extended Petri net model and the analysis techniques will be discussed. In general, the designed system is modelled by a Petri net at all levels in the design phase. The Petri net is analyzed to guarantee the proper behavior of the system. For example, the Petri net models of the system before and after the decomposition are analyzed to ensure the consistency of decomposition.

**Analyses of the system**

Throughout the whole design phase, the system is modelled by a Petri net. By analyzing the liveness, the boundness and the proper termination properties of the Petri net, the properties of the designed system can be unveiled. A Petri net is live if for each transition in the net, there always exists a firing sequence to fire it. By the liveness property of the Petri net, the designed system is guaranteed to be dead-lock free. A Petri net is bounded if for each place in the net, there exists an upper bound to the number of tokens that can be there simultaneously. By the boundness property of the Petri net, the number of buffers required between asynchronous processes can be determined and therefore information loss due to buffer overflow can be avoided. A Petri net is properly terminated if the Petri net always terminates in a well-defined manner. By the proper termination property, the designed system is guaranteed to function in a well behaved manner. (This point will be elaborated later.)

All the above three properties can be analyzed by constructing the reachability graph. (The method for constructing the reachability graph can be found in Reference 22.) In Figure 13, the Petri net models a concurrent system with five processes: r, s, t, u and v. The reachability graph is shown in Figure 14. Since the reachability graph is strongly connected and all the entries are finite, the concurrent system is live and bounded. By these two properties, the concurrent system is guaranteed to be dead-lock free and to have enough buffers.

**Extended Petri net**

However in real world problems, especially in real-time systems, the execution time of a process is a very important aspect of a system. In order to model this aspect, the Petri net model is extended to include the notion of time. In the extended net, each transition is associated with two times, $t_1$ and $t_2$, where

$t_1$ = minimum time a transition can begin firing after being enabled (it can be zero)

and $t_2$ = maximum time a transition can remain not fired after being enabled ($t_2 > t_1$)

Whenever a transition is enabled, it has to fire between times $t_1$ and $t_2$. In the case that a transition has $(t_1,t_2)$ as defined and an execution duration of $u$, it can be modelled by cascading two transitions with times $(t_1,t_2)$ and $(u,u)$ associated to the transitions, Figure 15. By using this ap-
Figure 16—Non-live condition of a timed Petri net

The problem of losing tokens during the firing of a transition is avoided.

By using the extended Petri net, the concurrent system discussed previously is analyzed again. The Petri net model of the concurrent system and its time constraints are shown in Figure 16. A careful study of the system will show that transition \( u \) is dead (it will never be fired). This is due to the long delay produced by transition \( r \). After firing transition \( r \), the token in place \( C \) will move into place \( E \) and then move into place \( A \) before the token from place \( D \) can move into place \( F \).

This situation is shown in the reachability graph in Figure 17. The two times associated to each arc are the earliest and the latest transition times of the arc. The two arcs \( t_1 \) and \( t_2 \) are blocked because their earliest transition time is longer than the latest transition time of the other transition emitting from the same node. Therefore by using the extended Petri net, the dead-lock situation due to time constraints can be detected. In addition to that, if the Petri net is a marked graph and if the execution time, \( T \), of a transition is taken to be \( t_2 \), the maximum computation rate, \( \rho \), of each of the transition can be computed by

\[
\rho = \min \left\{ \frac{N_k}{T_k} | k = 1, 2, \ldots, n \right\}
\]

where

\[
T_k = \sum_{t \in C_k} t_1
\]

is the sum of the times associated with transitions of circuit \( C_k \) and

\[
N_k = \sum_{t \in C_k} M_k
\]

is the number of tokens in places of circuit \( C \).

In finding the dead-lock situation in the extended Petri net, the algorithm to find the earliest and latest transition times in the reachability graph is quite lengthy and complicated. For example, the arc \( t^2 \) is associated with times \((3,5)\) rather than \((5,6)\) because transition \( t \) is enabled before transition \( s \) is fired. This type of interdependency can be resolved by tracing back the reachability graph to find the time when the transition is enabled. The computation can be done by a computer. Although it may be very lengthy, it is worth the effort to guarantee the designed system to be dead-lock free.

Proper termination

A Petri net is properly terminated if a token is injected into the input place, the net produces a finite number of tokens and terminates with all the tokens in its output place. This implies that if the corresponding system is initiated, it will always execute to completion without any intermediate results or pending conditions left in the system.

The proper termination property of a Petri net can be studied by its reachability graph. For example, the Petri net and its reachability graph are shown in Figure 18 and Figure 19. Since the only maximal node in the reachability graph is the state that place \( E \) contains a token, the Petri net is properly terminated.

The notion of proper termination can be used to verify the consistency of decomposition. For example, in Figure 20, transition \( s \) in global net \( N_1 \) is decomposed into the subnet \( S \). Virtual nodes \( A \) through \( G \) are then added to the subnet to form \( N_2 \) which is then analyzed for proper termination.
If N2 is properly terminated, the decomposition is consistent. This guarantees that no data stay in the decomposed process and no extra data are generated when the decomposed process is substituted for the original process in the system. If time constraints, \((t_1,t_2)\), are added to each of the transitions, the time constraints of transitions can be verified by finding the minimum and maximum path lengths in the decomposed net.

The verification techniques discussed above are by no means complete. They are just some of the approaches for analyzing the synchronization and consistency of interacting

---

**GLOBAL NET** \( \rightarrow \) **N1**

**N2**

---

Figure 19—Reachability graph

Figure 20—Consistency verification by Petri net
asynchronous processes. It is hoped that through these analysis techniques, synchronization errors in the designed system can be detected before implementation.

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, 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 analyzed should be small and therefore the complexity should be low. Mistakes or unfulfilled requirements found are traced back to the source of error. The system is then redesigned from that point.

CONCLUSION

This paper has discussed a systematic design procedure for a user oriented computer system. The main objective is to develop reliable, effective, modifiable systems with low cost and lead time. The methodology uses the concept of abstraction, stepwise refinement and modularity to design a DPS. 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: (i) requirement and specification phase, (ii) design phase, (iii) implementation phase, and (iv) evaluation and validation phase. The first two phases have been explored in detail in the previous sections. 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. Analysis and modelling techniques used in this phase are also discussed. By modelling the virtual system in an extended timed Petri net, the performance, the consistency and the liveness of the system can be determined. 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.

ACKNOWLEDGMENTS

This research was supported by the Ballistic Missile Defense Advanced Technology Center (BMDATC) under contract DASG60-77-C-0138. We would also like to thank Mr. C. R. Vick, J. E. Scalf, D. Palmer and M. Mariani for many helpful discussions related to this work.

REFERENCES