〈研究論文〉

Journal of the Korean Institute of Industrial Engineers
Vol. 13, No. 2, Dec. 1987

A Software Design Methodology for Designing APS (Activity Planning System)

Doo-Kwon Baik*

Abstract

The main purpose of this paper is to design a simple interactive software system for planning activities, called APS. We discuss the importance of informal and formal specifications of the system design, and survey a number of promising design methodologies. To design APS, we present a methodology based upon the hierarchical decomposition of the system, and illustrate the methodology as applied to the design of an activity planning system.

요약

본 논문의 주된 목적은 APS (Activity Planning System) 라고 불리우는 활동계획을 위한 간단한 대화식 소프트웨어 시스템을 설계하는데 있다. 여기에서는, 시스템 설계에 있어서 비형식적인 명세와 형식적인 명세의 중요성을 논하였고, 또한 몇가지 저명한 시스템 설계 방법론을 조사하였다. 그리고 APS를 설계하기 위하여 시스템의 계층구조적 분해에 의한 하나의 설계방법론을 제시하였고, 이 방법론을 응용하여 APS를 설계하였다.

1. Introduction

The activity planning system (APS) is an interactive software system that lets users use computers to create and schedule their activities over a period of time. In this basic form the program has a very limited, but convenient, interface for entering a textual description of an upcoming activity, its importance and a deadline by which it should be done. Such activity descriptions are suggested by the program according to an urgency measure that takes account of both proximity to deadline and importance. Since a future deadline becomes more imminent as time passes, the urgency ordering of activities changes daily so that an activity rises to the top of the order as its deadline approaches.

At any time, the user may request a listing of the most urgent impending activities. When an activity has been accomplished, this can be easily communicated to the program. Such an activity will no longer appear in the urgency listing, but will be retained in a calendar and marked as

*Department of Computer Science, Korea University

done. By viewing the calendar, the user can see the past or future activities that have been done or are still pending. Thus an automatic record of activity status is also maintained. This is helpful when one has forgotten whether an activity has ever been done, or desires to know the date that it was done, etc.

The concept of a scheduling calendar, in manual or electronic form is not new. What distinguishesAPS is its concept of urgency, which relieves the user of the burden of deciding what to do next, yet does not remove the flexibility in performing activities that a strict scheduling concept does. A second feature of APS is the incorporation of a user friendly interface, that makes the few operations required to communicate with the program extremely simple to perform.

There are currently available scheduling systems which do not have the concept of urgency and a user-friendly interface required few operations to communicate with the systems. For example, the project planning system(PPS) [12] was designed to aid project supervisors and to level management in the design of a work schedule during the planning phase of a large project. It is one of the PERT/CPM oriented scheduling system [1, 8], but it is not convenient to handle the system, because it does not have a user-friendly interface to communicate interactively with the system. Another type of the PERT/CPM oriented scheduling systems in the ManageMint[3] which is a modularized, menu-driven system, and tracks schedules, budgets and resources. It is designed to provide an integrated project management system. The Softrend company introduced the Time Scheduler [3] which is a comprehensive time management system. It may be used for appointment or project scheduling. It has some special features such as automatic time conflict checking for any activity and menu-driven function selection.

To produce a model or representation of an entity, we shall use the existing system design methodologies, which introduce techniques for the systematic design of well-structured and reliable system architecture. Zeigler [16] encourages conceptual development of the model prior to program writing. The conceptual development starts with an informal model specification and proceeds in stages to more formal specifications. Wymore [13, 14] introduced the process of system design in the context of the life history of system. The design process takes place between the client and the environment. Using this method, we shall focus on the external system design specification in terms of an input/output specification, a technology, and merit orderings, and to decompose the subsystems with coupling recipes of the system.

One of the major concepts of system design methodologies is the hierarchical system decomposition [2, 5, 6] which is associated with the principle of complexity decomposition. It is supported by the concepts of abstract operations, abstract data types, and abstract machines. Parnas has developed a technique and notation for writing implicit specifications by describing the states of an abstract state machine. The basic idea is to separate the operations into two groups: those which do not cause a state change but allow some aspect of the state to be observed and those which cause a change of state. Using the state machine model, we shall describe the internal specification of the system in order to know how the state of the object changes and a result of the applications of some of the operations.

2. Software Design Methodology

Software design is a process through which reguirements are translated into a representation of software. Pressman [9] presented the guideline for establishing a good design criteria. That is, a design should (1) exhibit a hierarchical organization, (2) be modular, (3) lead to modules that exhibit independent functional characteristics, and (4) be derived using a repeatable method.

These characteristics of a good design can be obtained through the advanced software design methodologies. Myers [4] recognized the need for advanced softare methods. His observations demonstrate problems with conventional software design practices, reflect the implications of changing technology on requirements for future design methods, and motivate the development of advanced software design methodologies.

The major cause of design errors is a lack of communication between the client and the designer concerning the requirements and between the designer and the implementor on what is expected of the implementation. It is important to detect design errors as early as possible, because design errors are costly to rectify, often requiring major changes to the system organization. Formal specifications can be useful for exposing design errors in the system development process[15].

In this section we shall introduce the advanced software design methodologies such as Wymore's system theoretic approach and Parnas' abstract model approach which will be used to develop the design. This formal specifications are superior to informal ones as a communication medium among the designers and the implementors, while the informal specifications are used for communication among the clients and the designers.

2.1 System Theoretic Approach

The system theoretic methodology derived from the tricotyledon theory of system design [13, 14] is useful for interdisciplinary teams at the mathematical level. Using the methodology one can define the problem as precisely and completely as possible, because the methodology is powerful enough to state any system design or analysis problem. Wymore presented the process of system design, in which there are seven steps called problem definition, preliminary design, final design, implementation, acceptance testing, operation, and retirement. The ultimate goal of the problem definition is the definition of the system design problem in terms of an input/output specification, a technology, merit orderings, which are over the input/output cotyledon, the technology cotyledon, and the feasibility cotyledon, and the system test plan. These are representing the questions asked of the client as the following:

"...What is the system supposed to do basically? How is the systems performance to be judged? What can be used to build the system? How is the use of resources to be judged? How are conflicts between performance and resource utilization to be resolved? How is the system to be tested?..."[13]

The parts of the problem definition are applied to define APS design problem. To build a model of a system, it is much easier to identify and model simpler system components, and identify input/output relations among the components, and then deduce the resultant system model, than to develop a model of the overall system directly, The basic manipulation of systems is that of putting components together to build a more complex model with the input/output coupling recipe.

2.2 Abstract Model Appoach

In the past few years some techniques for the systematic design of well-structured software architecture have been developed. Most techniques use the concept of modularity. The notion of abstraction has emerged for the construction of useful modules. To reduce the complexity, the principle of abstraction is concerned with selecting essential properties and omitting inessential ones.

Parnas [5, 6] has developed a technique and notation for writing implicit specifications by describing the state of an abstract machine. In this approach, the machine is identified with a single object, and the specification describes how the state of machine changes. An abstract machine specification characterizes the value returned and the new state for each possible operation invocation and each possible state of the machine. The specification describes the functional behavior of a system. A machine can sometimes be decomposed into smaller modules. Modules have operations and a state. A module specification is similar to that of a machine execpt that its specification can reference the state of other modules in its machine.

Depending on whether its invocation returns a value or causes a state change, an operation is declared to be one of the two functions, V-function (returns a value, but causes no change) or O-function (causes a state change, but does not return a value).

The specifications are given by indicating the effect of each O-function on the result of all the V-functions. This determines the smallest class of states necessary to distinguish the observable variations in the values of the V-functions, and also determines the transitions among these states caused by the O-functions. Each V-function's specification contains a definition of its initial value and exception conditions under which it may not be successfully called. Each O-function's specification contains exception conditions, and a description of the effects of a call to the O-function. If an exception condition is satisfied, no value is returned in the case of a V-function, and no effects are executed in the case of an O-function. In both cases control is returned to the calling program.

3. Design of APS

A busy executive would like to have his/her work schedule planned so that he/she is freed of the annoying problem of deciding which activity must be completed next. An even larger problem can accrue if the individual overestimates or underestimates the amount of time required for each activity. This may lead to a situation where it is impossible to complete tasks on schedule. In the ideal case such scheduling of activities could be automated so that a busy executive is freed of the cumbersome job of scheduling activities. To relieve the executive of this task, we can design a computer-aided activity planning system, which will organize time more effectively. Such a system is a natural extension of the current trends toward increased use of computers in management areas.

In any activity planning system, there are certain minimal requirements specified by the potential user. For example, (1) The system must be effective. It should provide a schedule of user activities that can be completed at the times specified by the user. (2) The system must be simple to use. At worst, it should require no more effort to use than the current system. (3) The system and the user should communicate readily. It means that there is a good user-system interface, which will minimize potential confusion.

The purpose of a design is not only to create the desired system but also to provide such results that fully satisfy or compromise both clients and designers' requirements. Having been given the set of requirements, the designers produce the overall optimal system. Such a design is chosen after the input/output (I/O) and utilization of resources (U/R) merit orders are specified and an overall merit order is defined in such a way that one is able to compare any two proposed systems with respect to the tradeoff between I/O performance and U/R performance.

In the next two sub-sections, we present a certain level which is suitable to a simple interactive system for planning activities. At this level the user can store and update his activities, display them in a chronological order or by urgency, and search a desired activity.

3.1 Informal Specification

From the given problem description we can build a model which satisfies the system requirements. In this process, we present the essentials but not the details, which will help one to maintain a clear gestalt of the model as it develops. The informal specification[16] should help both the users and the designers to see the basic outlines of the model and to visualize it within the framework of their prior conceptions about how things work. Having achieved an informal specification with the model structure, we can describe the formal specification of the model.

The way to informally present a model as shown in Figure 1 is to describe its components, descriptive variables, and component interactions as the following:

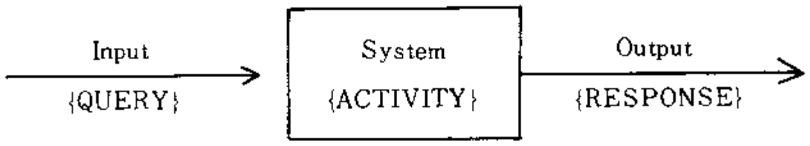

Figure 1 The model of APS

(1) COMPONENTS are the parts from which the model as shown in Figure 1 is constructed.

- QUERY.1, QUERY.2, ..., QUERY.l
- RESPONSE.1, RESPONSE.2, ..., RESPONSE.m
- ACTIVITY.1, ACTIVITY.2, ..., ACTIVITY.n
- SYSTEM

(2) DESCRIPTIVE VARIABLES serve as tools to describe the conditions of the components at points in time.

- For QUERY.i,
  - Q.ARR.TIME with range the positive reals; Q.ARR.TIME=y means the QUERY.i arrives at y time units.
  - COMMAND with range {‹add ACTIVITY.k›, ‹done ACTIVITY. k›, ‹modify ACTIVITY.k›, ‹list›, ‹calendar (time1, time2)›, ‹search A.IDENTITY›} where (time1, time2) is a period from time1 to time2; It indicates the command type the QUERY.i is containing.
- For RESPONSE.j,
  - OUTPUT with range {‹not found›, ‹no activity›} or the subsets of {ACTIVITY.1, ..., ACTIVITY.n}.
- For ACTIVITY.k,
  - A.IDENTITY with range the positive integers; A.ACTIVITY.k=q indicates that the identification of the ACTIVITY.k is q.
  - A.PRIORITY with range the positive reals; A.PRIORITY=r indicates that the importance of the ACTIVITY.k is r.
  - A.DEADLINE with range the positive reals; A.DEADLINE=s means the deadline of the ACTIVITY.k is s.
  - A.LEFTTIME with range the positive reals; A.LEFTTIME=t means A.DEADLINE$-$Q.ARR.TIME$\longrightarrow$ t.

– A.TEXT with range the set of character strings; A.TEXT =u indicates the description of the ACTIVITY.k is u.
– A.TRADEOFF with range the positive reals; A.TRADEOFF =v means that the tradeoff of the ACTIVITY.k is v which is computed by a tradeoff function with inputs(A.PRIORITY and A.DEADLINE of the ACTIVITY.k).

• For the SYSTEM,
– TRADEOFF.QUEUE with range the subsets of {ACTIVITY.1, ..., ACTIVITY.n}.
– DEADLINE.QUEUE with range the subsets of {ACTIVITY.1, ..., ACTIVITY.n}.
– PRIORITY.QUEUE with range the subsets of {ACTIVITY.1, ..., ACTIVITY.n}.

(3) COMPONENT INTERACTIONS are the rules by which components exert influence on each other, altering their conditions and so determining the evolution of the model's behavior over time.
– When QUERY.i arrives at the time Q.ARR.TIME with the COMMAND, the ACTIVITY.k can be added, done, modified, or displayed. If the COMMAND is ‹add ACTIVITY.k›, the SYSTEM adds ACTIVITY.k to the DEADLINE.QUEUE ordered by A.DEADLINE and the PRIORITY.QUEUE ordered by A.PRIORITY, and to the TRADEOFF.QUEUE ordered by A.TRADEOFF by computing the A.TRADEOFF of the ACTIVITY.k. If the COMMAND is ‹done ACTIVITY.k›, the SYSTEM takes ACTIVITY.k out from the DEADLINE.QUEUE, the PRIORITY. QUEUE and the TRADEOFF.QUEUE. If the COMMAND is ‹modify ACTIVITY.k›, the SYSTEM replaces ACTIVITY.k in the DEADLINE.QUEUE, the PRIORITY.QUEUE, and the TRADEOFF.QUEUE by computing the A.TRADEOFF of the ACTIVITY.k again. If the COMMAND is ‹search A.IDENTITY›, ‹list›, or ‹calender (time1, time2)›, the SYSTEM displays a subsets of {ACTIVITY.1, ..., ACTIVITY.n}.

3.2 Formal Specification

In this part, two major methods are applied to describe the formal specification of APS. While the following external design specification is based on Wymore's approach, the following internal design specification is based on Parnas' approach.

3.2.1 External Specification

The external specification of the system design can be done by using the input/output (I/O) specification of the system and the decomposition of the components input/output specification[14, 17].

(1) System I/O Specification.

The system has inputs, outputs, and an internal structure as shown in Figure 2. I/O specification comprises two parts: I/O interface and I/O constraints. Using a finite state machine model I/O constraints can be specified as the following:

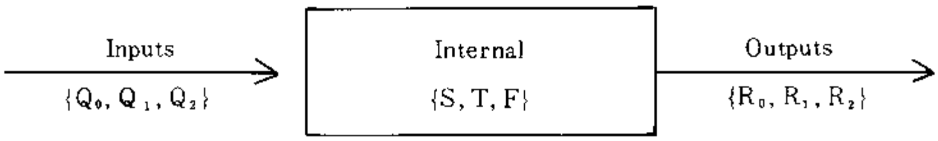

Figure 2 Input/Output of the System

M =‹Q, R, S T, F›, where

Q is the input set,

R is the output set,

S is the set of states: {Si i=1, ..., n},

T is the transition function:$Q \times S \rightarrow S$,

F is the output function: $Q \times S \rightarrow R$.

a) I/O Interface

Input ports

Q_0= {c_0}

Q_1= {c_1, c_2, c_3}

Q_2= {c_4, c_5, c_6}

c_0=[signon ‹time›]

c_1=[list]

c_2=[calendar ‹d1, d2›]

c_3=[search ‹a›]

c_4=[add ‹d, p, text›]

c_5=[done ‹id›]

c_6=[modify ‹b›]

where

time, d, d1, d2, p are positive reals,

text is a character string, id is a positive integer,

a is in {id, d, p, text},

b is a subset of {d, p, text}.

Output port

R_0= {signon.mess}

R_1= {‹id, d, p, text, status›}

R_2= {'added', 'done', 'already done', 'modified'', 'not found', 'no activity'}

where

signon.mess is a character string,

status is in {0, 1}

b) I/O Constraints

Let ID= {id of activity in the system},

E= {e_i | i=1, ..., n}, where e_i represents all information referring to activity$_i$,

M(E)=a set of subsets over E,

then $S=M(E) \times PN \times$ {time},

where PN=a set of tradeoff values.

Transition Function (T):$Q \times S \rightarrow S$

$T(S_i, c_0)=S_i$

$T(S_i, c_1)=S_i$

$T(S_i, c_2)=S_i$

$T(S_i, c_3)=S_i$

$T(S_i, c_4)=S_i$ if id(c_4) is in ID,

$=S_i'$ otherwise

where S_i = [M(E)+ {‹id, d, p, text, status›}] $\times$ PN $\times$ {time}

$T(S_i, c_5)=S_i''$ if id(c_5) is in ID,

$=S_i$ otherwise

where S_i''=M'(E) $\times$ PN $\times$ {time}, where M(E) has been changed to M'(E)

$T(S_i, c_6) = S_i''$ if $id(c_6)$ is in ID,

$= S_i$ otherwise

Output Function(F):$Q \times S \rightarrow R$

$F(S_i, c_0) = \{signon.mess\}$

$F(S_i, c_1) = \{\langle id, p, d, text, status \rangle\}$ if $id(c_1) = 0$

$=$'no activity' otherwise

$F(S_i, c_2) = \{\langle id, p, d, text, status \rangle\}$ if $d1 \quad d(c_2) \quad d2$

$=$'no activity' otherwise

$F(S_i, c_3) = \{\langle id, p, d, text, status \rangle\}$ if $id(c_3)$ is in ID,

$=$'not found' otherwise

$F(S_i, c_4) =$'already added' if $id(c_4)$ is in ID,

$=$'added' otherwise

$F(S_i, c_5) =$'done' if $id(c_5)$ is in ID and $status(c_5) = 0$,

$=$'already done' if $id(c_5)$ is in ID and $status(c_5) = 1$,

$=$'not found' otherwise

$F(S_i, c_6) =$'modified' if $id(c_6)$ is in ID

$=$'not found' otherwise

(2) Components I/O Decomposition.

The essential components of the system are shown in Figure 3.

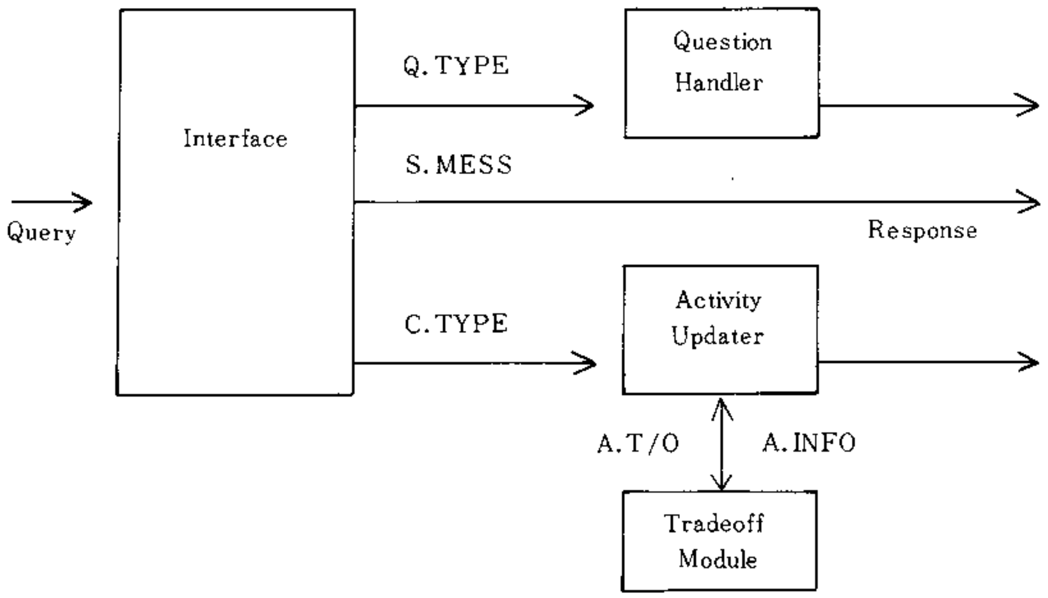

Figure 3 Input/Output of the Components

Interface

- Input ports:
  TIME - with range the positive reals
  QUERIES - with range $\{Q_0, Q_1, Q_2\}$
- Output ports:
  SIGNON.MESS - with range the set of strings

QUESTION.TYPE - with range $\{c_1, c_2, c_3\}$
COMMAND.TYPE - with range $\{c_4, c_5, c_6\}$

Qestion.Handler

- Input ports:
  ΓIME - with range the positive reals
  ACTIVITY.SET - with range the subset of $\{a_1, a_2, \ldots, a_n\}$
  QUESTION.TYPE - with range $\{c_1, c_2, c_3\}$
- Output port:
  ACTIVITY.LIST - with range the subset of $\{a_1, a_2, \ldots\ a_n\}$

Activity.Updater

- Input ports:
  ACTIVITY.SET - with range the subset of $\{a_1, a_2, \ldots, a_n\}$
  COMMAND.TYPE - with range $\{c_1, c_2, c_3\}$
- Output ports:
  UPDATE.RESPONSE - with range the set of character strings
  ACTIVITY.INFO - with range $\{a_1, a_2, \ldots, a_n\}$

Tradeoff.Module

- Input ports:
  TIME - with range the positive reals
  ACTIVITY.INFO - with range $\{a_1, a_2. \ldots, a_n\}$
- Output port:
  ACTIVITY.TRADEOFF - with range the nonnegative reals

3.2.2 Internal Specification

The internal design specification of the system can be written by the abstract machine model [2, 6, 15] which describes the function behavior of the system. The machine is decomposed into several smaller modules. Here we choose the two key modules which are the question .handler and the activity.updater modules. While the activity.updater module has three O-functions, the question.handler module has three V-functions as the following:

Global Function

- V-function:HAS
  argument:activity
  output:BOOLEAN
  purpose:to reveal whether a particular activity is in the set of activities
  initial:FALSE

Activity.Updater Module

- O-function:ADD
  argument:activity
  purpose:to add an activity to the set of activities
  exception:HAS(activity)=TRUE
  effect:HAS(activity)='HAS(activity)' if it is not the activity being added; otherwise TRUE
- O-function:DONE
  argument:activity
  purpose:to delete an activity from the set of activities
  exception:HAS(activity)=FALSE

effect:HAS(activity)='HAS(activity)' if it is not the activity being done;otherwise FALSE

- O-function:MODIFY

  argument:activity

  purpose:to modify an activity being added the set of activities

  exception:HAS(activity)=FALSE

  effect:HAS(activity)='HAS(activity)' if it is not the activity being added; otherwise FALSE

Question.Handler Module

- V-function:LIST

  argument: none

  output the subsets of the set of activites

  purpose:to display the activities in order of tradeoff

  exception:HAS(activity)=FALSE

  initial:none

- V-function:CALENDAR

  argument:(date1, date2)

  output:the subsets of the set of activities

  purpose:to display the activities in chronological order within the period(date1, date2)

  exception:HAS(activity)=FALSE

  initial:none

- V-function:SEARCH

  argument:activity.id

  output:activity

  purpose:to display a desired activity to be searched by an activity.id

  excepton:HAS(activity)=FALSE

  initial:none

4. Conclusions

The activity planning system(APS) is a user-friendly interactive software system that lets users create and schedule their activities by communicating readily with the system, and is a decision support system [10] that help users to decide which activity should be done next by showing an urgent list of pending activities.

To design APS,

1) we introduced the informal and formal specifications for the system design,
2) we presented a software design methodology based upon a hierarchical decomposition of the system, combining the promising design methodologies, and
3) we applied the method to the design of APS in forms of the hierarchical specifications such as informal/formal and external/internal specifications.

There are a number of possible extensions to the system that will increase its capabilities, including:

1) adding a graphical display of daily/monthly calendar that would allow users to easily adjust their schedules,
2) an ability to organize related groups of user-specified activities into projects that would allow users to have their projects planned,

3) an ability to allow the users to define the format of projects in the form of templates that can be activated to relieve the users of the need to redefine similar projects repeatedly, and
4) an ability to deal with various kinds of activities:deadlined task, undeadlined task, deadlined expectation, undeadlined expectation, once-only event, etc, so that activities may be strong together in units.

References

1. Elmaghraby, S., *Activity Network*, Wiley, 1977.
2. Guttag, J. V., E. Horowitz, D. R. Musser, "Abstract Data Types and Software Validation", *CACM*, Vol.21 ♯12, Dec. 1978.
3. INTERFACE AGE magazine, "New Products", *Interface Age*, Vol. 8 ♯4, April, 1983.
4. Myers, W., "The Need for Software Engineering", *Computer*, Vol. 11 ♯2, 1978, pp 12-26.
5. Parnas, D. L., "On the Use of Transition Diagrams in the Design of A User Interface For an Interactive Computer System", *Proceedings of the ACM National Conference*, 1969, pp 379-386.
6. Parans, D. L., "A Technique for Specification of Software Modules",*CACM*, Vol. 15 ♯5, May 1972, pp 330-336.
7. Parans, D. L., "On the Criteria To Be Used in Decomposing System into Modules", *CACM*, Vol. 15 ♯12, Dec. 1972, pp 1053-1058.
8. Phillips, D. T., A. Garcia-Diaz, *Fundamentals of Network Analysis*, Prentice, 1981.
9. Pressman, R. S., *Software Engineering*, McGraw-Hill, 1982.
10. Sprague, R. H., E. D. Carlson,*Building Effective Decision Support System*, Prentice, 1982.
11. Vansteenkiste, G. C, J. Spriet. *Computer-aided Modelling and Simulation*, Academic Press, 1982.
12. Wayne State University, "Project Planning System", *MTS manual*, WSU Computing Services Center, July 1973.
13. Wymore, A. W., *Systems Engineering Methodolgy for Interdisciplinary Teams*, Wiley, 1976.
14. Wymore, A. W., "Applications of Mathematical System Theory to System Design, Modelling and Simulation", *Proceedings of the Winter Simulation Conference*, 1981, pp 209-219.
15. Yeh, R. T.(editor), *Current Trends in Programming Methodology*, Vol. 1, Prentice, 1977.
16. Zeigler, B. P., *Theory of Modelling and Simulation*, Wiley, 1976.
17. Zeigler, B. P., *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, 1984.