A SPACE-CONSTRAINED RESOURCE-CONSTRAINED SCHEDULING SYSTEM FOR MULTI-STORY BUILDINGS

by

Walid Y. Thabet

Dissertation Submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in Partial Fulfillment of the Requirements for the Degree
of

DOCTOR of PHILOSOPHY

in

Civil Engineering

APPROVED

[illegible]

Y. J. Beliveau, Chairman

[illegible]

M. C. Vorster

[illegible]

J. M. De La Garza

Terry Rakes

T. R. Rakes

S. Jayaram

S. Jayaram

December 9, 1992
Blacksburg, Virginia

A SPACE-CONSTRAINED RESOURCE-CONSTRAINED SCHEDULING SYSTEM FOR MULTI-STORY BUILDINGS

by
Walid Y. Thabet

Yvan J. Beliveau, Committee Chairman

Civil Engineering

(ABSTRACT)

Current planning and scheduling techniques ignore the requirements of activities for work area or space. Any task or activity requires a specific work space for its execution. This demand is based on the space requirements of each resource allocated to the activity. When such required demand becomes unavailable, the activity or task can not be executed or, in some cases, is performed with a lower productivity rate. This is because performance and maneuvering of either crew or equipment may become difficult and sometimes not possible.

This research provides a structured methodology to deal with the problem of limited work space availability. The research's domain focuses on multi-story building construction with particular focus on the repetitive portion of the facility (i.e. typical floors). The research studies the issues of work space as a new decision factor for schedule generation in this type of construction.

A scheduling model is developed to define and incorporate work space availability in the scheduling process. The model includes a method to quantify work space parameters

(space demand and space availability) for any activity. The model also presents a procedure to compare space demand to availability and provide scheduling decisions to sequence each activity based on these two parameters.

The model allows for availability limits to be placed on resources required for the work by implementing limited resource scheduling techniques in the scheduling procedures. Horizontal and vertical logic constraints associated with repetitive work are also incorporated in the scheduling process of the model. Work continuity issues and varying productivity rates are used as scheduling decision options. The model adopts a procedure to schedule non-continuous activities using variable length segments along the typical floors.

In addition, the model allows for adjusting the initial defined resource demand pools for different activities to account for any modifications that may occur to the activity duration during scheduling. Loss of productivity as a result of the combined effect of travel time and learning curve phenomena is also incorporated in the generation of the schedule by the model.

The model is taken to a prototype proof of concept by developing **SCaRC** (Space Constrained and Resource Constrained) scheduling system. The system is implemented using a knowledge-based approach.

Acknowledgements

I would like to thank my advisory committee for their contribution to my research and their willingness to review my dissertation.

My deepest appreciation and gratitude goes to my advisor, Dr. Yvan Beliveau, for his continuous support and guidance throughout the course of this research. Special and sincere thanks to Dr. Michael Vorster for his support. His thoughts and comments in all the committee meetings made it challenging, interesting and pleasant.

I would like to thank my friend and colleague, Dr. Ayman Morad, for his friendship and support that helped me complete this research. Special thanks to my friends Joe Lubow, and Dr. Wala Mogawer for their moral support and friendship.

My thanks and love to my sister Maha, my brother Amro and Haja Halima for their care and love throughout the years and standing next to me to finish this work. My thanks and love also to my best friend Janice Petrone for her sincere friendship and for helping me in the printing and copying of this document.

Finally, my thanks and love go to my parents; my father Yehia and my mother Afaf who supported me with their love and care and made it important and possible for me to complete this work. To them I dedicate this dissertation.

Table of Contents

List of Figures

List of Tables

1.0 INTRODUCTION

1.1 Problem Statement

In many different areas of the industry including construction, project planning and scheduling is considered to be a vital element to the successful execution of the work. Project planning consists of identifying the work tasks or activities needed to achieve project completion and to determine the logical sequence among these activities based on different constraints and requirements. Project scheduling can be defined as introducing time parameters for these tasks to determine their start and finish dates and the overall completion time of the project.

The planning and scheduling process of activities would have been considered simple if such activities could be arranged in a sequence determined only by their technological dependency. That the construction of ground columns can only proceed after the completion of the ground footings is a typical example of such dependency constraint. Unfortunately, other constraints in addition to the technological dependency control and determine the logical relationships among the different activities making the planning and scheduling process more complex. Such constraints include: limited availability of resources, limited availability of work space, weather and climatic conditions, material delivery dates, etc. Unless these constraints are suitably provided for in the network plan and considered by the scheduling technique, the generated schedule will not be realistic and will not reflect the actual conditions that affects the execution of the work.

Current network techniques and linear scheduling methods examine and develop the sequence of activities based mainly on technological relationships among the activities (time-based schedules), and their requirements for labor, material and equipment (resource-based schedules).

Time-only or time-based project scheduling techniques, such as PERT and CPM procedures, rely on precedence logic defined by technological requirements among the activities (i.e. hard logic) to perform basic scheduling computations. Inherent to these techniques is the assumption that resource requirements for each activity is unlimited for the duration of the project and that only technological requirements constrain activity start and finish times.

One consequence of using such techniques is that the schedules produced may not be realistic when resource constraints are not considered. For this reason, these basic PERT/CPM procedures have been called:

"a feasible procedure for producing non feasible schedules."
[Moder, Phillips and Davis, 83]

Resource-based project scheduling techniques consider resource availability as an important factor for activity scheduling. Such techniques recognize the conflict between the resources that can be made available to a project and those apparently needed as determined from the network model. This recognition improves the schedule as it makes it more feasible and realistic.

Although these scheduling routines (time-based and resource-based) are widely used in the scheduling process, they ignore the requirements of activities for work area or space. Any task or activity requires a specific work space, termed *work space demand,* for its execution. This demand is based on the space requirements of each resource (i.e manpower, equipment, and material) allocated to the activity. When such required demand becomes unavailable, the activity or task can not be executed or, in some cases, is performed with a lower productivity rate. This is because performance and maneuvering of either crew or equipment may become difficult and sometimes not possible.

A very traditional example that demonstrates the work space problem is the sequencing of tasks for the construction of an airplane cockpit. Many of the tasks involved in this operation such as: electrical wiring and piping, installation of control panels and computer units, mechanical works, etc., may be technically independent and can be concurrently executed. However, because of the limited space availability inside the cockpit, it is not possible to fit all the crews of these tasks at the same time and it may be necessary to schedule the sequence of such tasks to be executed sequentially rather than in parallel.

Another problem resulting from the limited space inside the cockpit is the difficulty in increasing the production rate of these tasks. In general, production rate of any activity is directly proportional to the amount of resources allocated to that activity. Increasing the number of resources should increase the production. In an aeroplane cockpit, the number of resources that could be allocated to each task is not, by any means, determined by the availability of these resources, but rather is controlled by how many can fit in the cockpit space at any one time. As a result, the rate of production can not be continuously increased by increasing such crews, but becomes limited to a value that corresponds to the maximum number of technicians/engineers that can fit into the work area.

In multi-story building projects, the problem of limited space availability in the work area is present as a result of many factors. Due to the nature of multi-story buildings where construction is mostly in crowded downtown locations, storage of construction materials on site premises is not always possible because the available space is usually limited. Instead, construction floors are used as a main storage area for materials required by many activities. The space consumed by material stored in the work areas affects the overall availability of space required for activities manpower and equipment operating in these areas. In some cases, it might not be possible for crews to perform until the stored material is used. The scheduling procedure should take into account the space consumed by the material stored. The current scheduling techniques do not account for this problem during the generation of schedules.

Another factor contributing to the work space problem in the floor is that resources of some activities require a large amount of work space during execution. For example, in mechanical duct installation, materials are first spread and assembled partially on the floor area before they are installed in place. Available work space for other activities sharing the same work area is considerably reduced. The current scheduling techniques do not consider verifying the space required by the resources of such activities with the available space of the work area.

A third factor that adds to the limited space availability problem is the work area allocation policy among crews of different trades working on the floor. Based on such policy, crews of some activities that are logically independent are assigned solely to the work area. Although space may be available to accommodate crews of other trades concurrently, these crews may not be allowed to occupy or store their equipment and materials in such space. This policy among the different trades is implicitly understood by the crews and, in some cases, explicitly stated by the main contractor or construction manager. The policy is also adopted in many projects to prevent mixing of crews which may lead to interference and quarrels resulting in reduced productivity and project delays. As a result, work space is considered theoretically unavailable for concurrent activities until the activity assigned to such space is entirely completed. The current scheduling techniques do not account for allocating space entirely to an individual trade that is logically independent and can concurrently work in the same work area with other crews of different trades.

The limited space availability is sometimes remedied by manually checking the areas that might become congested and attempting to maintain a specific area per resource for the construction operation. However, such manual procedures are not always applied to all work areas during the initial schedule and are not repeated as the schedule is unavoidably updated.

If quantifying and incorporating work space is not part of the scheduling process, it will have no influence on the schedule. As a result, it is highly desirable to integrate work space availability as a construction constraint along with other constraints in developing schedules. At times, it may be even advantageous to develop a critical path simply based on congestion when all other influencing variables seem to have remained unchanged.

The integration of work space availability as a constraint in the scheduling process will provide a number of benefits such as: generation of a more realistic schedule, reducing delays and improving productivity.

1.2 Research Scope

Meeting the space needs of resources in scheduling multi-story building projects thus requires a new scheduling model that incorporates work space as an additional constraint in the schedule generation process. The model should take into consideration the space demand of all resources involved in the project versus the available space for these resources. Based on space availability, the model should provide scheduling decisions to account for limited space conditions. In addition, the model should allow to integrate work space constraints with other scheduling constraints (e.g. resource constraints) to insure that all constraints can be considered in the generation of the schedule.

This research provides a structured methodology to deal with the problem of limited work space availability in scheduling multi-story buildings. The research studies the issues of work space as a new decision factor for schedule generation. The objective is to improve the feasibility of the generated schedule by incorporating work space availability as an additional influential constraint in the schedule generation process.

A scheduling model is developed to define and incorporate work space availability in the scheduling process. The model includes a method to quantify work space parameters (space demand and space availability) for any activity. The model also presents a procedure to compare space demand to availability and provide scheduling decisions to sequence each activity based on these two parameters.

The model also allows for limited availability limits to be placed on resources required for the work. The model's domain focuses on multi-story building construction with particular focus on the repetitive portion of the facility (i.e. typical floors). Because of the repetitive nature of this type of construction, emphasis is also given to continuity issues and varying production rates as main decision factors in the scheduling procedures of the model.

The model is taken to a prototype proof of concept by developing **SCaRC** (Space Constrained and Resource Constrained) scheduling system. The system is implemented using a knowledge-based approach. The SCaRC system demonstrates the use of construction scheduling knowledge about limited resources and work space availability along with other constraints to generate a construction schedule for the repetitive floors. The system takes as input a logical network plan for a single typical floor. The system produces as output a space-constrained and resource-constrained schedule for all the typical floors of the facility.

1.3 Research Methodology

To accomplish the objective of the research, the SCaRC prototype scheduling system is developed to quantify and integrate work space constraints with resource constraints and other constraints to generate schedules for the typical floors of multi-story buildings.

Figure 1.1 depicts an overall basic picture of the scheduling process employed by the developed system. The process comprises of two basic steps. The first step is a **Pre-scheduling Step** and includes different data preparation procedures to define and generate necessary scheduling data. This data is processed in the second step to generate the schedule. The second step is a **Scheduling Step** and constitutes the actual production of the schedule. In this step the input data defined in the first step is processed to verify resource constraints and space constraints along with other constraints in order to make the appropriate scheduling decisions. The outcome of this step is a space-constrained resource-constrained schedule for all the typical floors.

Pre-scheduling Step

The *Pre-scheduling Step* consists of several procedures to define and generate activity and project data necessary for the production of the schedule. These procedures are accomplished within a database environment. Data defined in this step is processed in the second step (i.e scheduling step) to generate a construction schedule.

The pre-scheduling step includes a formalized procedure to define work space demand and availability parameters for each activity. The procedure provides a method to estimate work space demand of each activity. The method is based on evaluating work space

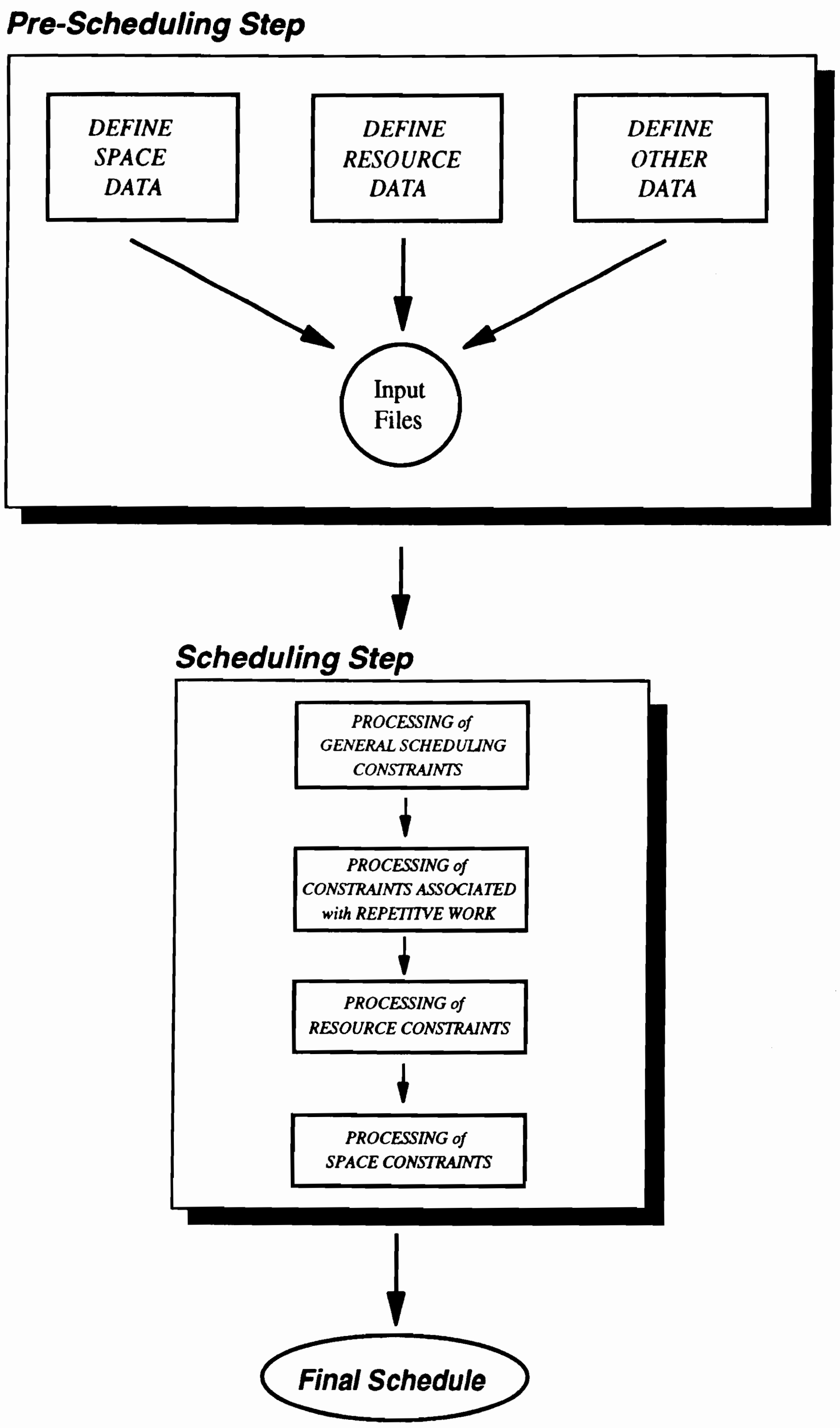

Pre-Scheduling Step
DEFINE SPACE DATA
DEFINE RESOURCE DATA
DEFINE OTHER DATA
Input Files
Scheduling Step
PROCESSING of GENERAL SCHEDULING CONSTRAINTS
PROCESSING of CONSTRAINTS ASSOCIATED with REPETITVE WORK
PROCESSING of RESOURCE CONSTRAINTS
PROCESSING of SPACE CONSTRAINTS
Final Schedule

Figure 1.1 - Scheduling Steps Adopted by the SCaRC System

requirements of all resources allocated to the activity. A global space demand database file is designed to allow for defining space demand values for unit construction resources. These global values can be used to assist the user in estimating space demand of project resources.

The procedure also provides a structured methodology to define and quantify available work space for the activity in the different construction work areas. This methodology is based on dividing the typical floor into a number of work blocks. Each block represents a work area of the floor at a specific location during a specific time period. Available work space in each work block is quantified utilizing geometric data of design elements extracted from a 3D CAD (Computer Aided Design) model of the floor. Geometric information of all design elements in each block is used to calculate the space of that block.

This step also includes other procedures to define other activity and project data. Such data includes: scheduling data extracted from the initial schedule of the typical floor (activity number, duration, preceding activities, succeeding activities, etc), special data associated with repetitive work in multi-story projects (e.g. continuity data), resource demand and availability data, and other project data (e.g. start and finish floor numbers).

Scheduling Step

The *Scheduling Step* is the second step of the developed SCaRC system and comprises of different scheduling procedures responsible for the actual production of the schedule. These procedures integrate work space constraints with resource constraints and other constraints to generate a construction schedule.

The generation of the schedule in this step is accomplished in two stages. In the first stage, limited resource availability along with other scheduling constraints are checked and verified. In the second stage, space constraints are checked to insure that adequate work space is available for each activity.

The scheduling procedures responsible for this second step constitute the knowledge-based component of the SCaRC scheduling system. These procedures are grouped into three main modules. The modules interact with each other to generate the sequence of activities and establish a schedule. The three modules are:

1- An External Data Interface Module.

2- A Controller Module.

3- A Sequence Generation Module.

The ***External Data Interface Module*** is responsible for extracting scheduling data stored in the system's database. The module is also responsible for printing the outcome of the scheduling process (i.e. the generated schedule) to an output file. Data stored in this file is extracted by the user from the database system for review of scheduling results.

The ***Controller Module*** controls the overall scheduling process of the system. The module controls the execution of the external data interface module to start the extraction of data. Once data is extracted, the module starts the scheduling process. The module determines activities that are eligible for scheduling at any time period and continuously updates the scheduling time. The module is also responsible for the heuristic prioritization process of the activities to rank those competing for resources and/or space.

Another function of the controller module is to manipulate or control the sequence of the scheduling process. The module is responsible for sending the appropriate instructions to the sequence generation module (third module) to carry out the appropriate scheduling step. The scheduling process is continuously monitored by the controller module. Once the overall schedule is generated, the controller module transfers control back to the external data interface module to print the generated schedule to the output file.

The ***Sequence Generation Module*** is the center piece of the scheduling system. The module is responsible for the actual sequencing of each activity along the typical floors in order to establish the schedule. The module generates the sequence based on verifying different constraints including limited resource and work space availabilities. These constraints are checked by the module in two consecutive stages. In the first stage the module verifies horizontal and vertical logic constraints among the different activities along with satisfying continuity issues. Activity production rates are established and activity resource demand pools are modified, if necessary, to correspond to the new rate.

The module is also responsible for verifying resource availability to insure that enough resources are available to perform the activity for its entire duration along the typical floors.

In the second scheduling stage, the module verifies the availability of space for each activity and carries out the different scheduling decisions necessary to schedule or delay the activity based on space availability considerations.

Constraints verification in both stages are continuously repeated until each activity satisfies all applicable constraints and its final start and finish dates are determined. The process continues until all activities are scheduled. At this point, the control is shifted by the controller module back to the external data interface module to output the scheduling results to the system's database.

The scheduling system is developed using a microcomputer based system running under MS-Dos system. The ART-IM (Automated Reasoning Tool for Information Management) expert system development environment is the prime software tool for this research. dBASE-IV and Quick C 2.5 are also utilized in developing the scheduling system.

1.4 Research Contribution

The research takes the construction scheduling process a step further by acknowledging the requirements of activities for work area or space necessary for material storage and movement of manpower and equipment. The developed model recognizes work space as a new constraint in the production of the schedule.

The research characterizes work space in terms of two parameters; work space demand and work space availability. Space demand for an activity defines space required to accommodate the activity in a work area. Space availability for the activity defines available space of work area at a particular location during a particular time period.

The developed scheduling model comprises a method to quantify these parameters. Space demand for an activity is quantified based on space required to accommodate each resource allocated to the activity. A mathematical formulation is proposed to estimate required space for any resource. To quantify space availability, a zone/layer work block structure for any work area is proposed. A work block defines a particular location in the work area (zone) during a particular time period of construction (layer). Space availability for any activity is evaluated based on the total available space of work block(s) to which the activity is allocated.

The model also comprises a space-based scheduling procedure to implement these space parameters for sequencing of activities. The proposed procedure compares space demand with availability for any activity and considers several scheduling decisions and actions to account for lack of work space. The space-based scheduling procedure is integrated with other scheduling procedures in order to generate the overall schedule. The result is a more feasible and realistic schedule.

In addition to recognizing and incorporating work space as a constraint in the scheduling process, the developed model allows for the following:

- The scheduling model recognizes the combined effect of travel time and learning curve phenomena associated with repetitive work in multi-story buildings on the overall crew productivity.

- The model provides for a step to modify the initial resource demand pool defined by the user for each activity. During the scheduling process, the model allows to adjust the resource demand pool based on the selected activity duration.

- The model allows for scheduling non-continuous activities in segments with variable number of floors. Based on a maximum number of splits parameter defined by the user, a procedure is adopted to schedule each segment independently.

1.5 Research Limitations

The research provided in this dissertation is a first attempt for quantifying space as a limiting resource in the constraints scheduling process. The SCaRC scheduling system developed under this research constitutes a first implementation prototype to tackle the problem of congestion in scheduling construction projects. The system is not robust and additional work is needed to test and validate the system to actual construction scheduling case scenarios. In addition, further programming is needed to enhance the capabilities of SCaRC as an efficient scheduling tool.

1.6 Dissertation Organization

The dissertation is divided into 10 chapters. This chapter ***"INTRODUCTION"*** provides an introduction to the proposed research. Background of the limited space availability problem in construction scheduling is presented. Scope and objectives of research are discussed. An abstract description of the model and scheduling system is also introduced. Contribution and limitation of research is outlined.

Chapter 2 ***"LITERATURE REVIEW"*** reviews the direction of research in the area of construction planning and scheduling that utilizes state-of-the-art computer technology and Artificial Intelligence (AI) technology exhibited by expert systems (ESs) and knowledge-based systems (KBSs). A brief description of several planning and scheduling systems developed for research purposes is presented.

Chapter 3 ***"KNOWLEDGE-BASED SYSTEMS TECHNOLOGY CONCEPTS and REVIEWS"*** reviews the general concepts and components of knowledge-based systems.

The chapter provides a background of this technology and elaborates on the overall architecture of such systems. The chapter also presents techniques of knowledge representations and search methodologies adopted by KBSs.

Chapter 4 ***"CONSTRUCTION CONSTRAINTS in PROJECT PLANNING and SCHEDULING"*** discusses the different constraints that control the generation of construction plans and schedules. The chapter introduces the types of logical relationships that can exist among the different construction activities and provides a description of the different project-related and schedule-related constraints that define such relationships.

Chapter 5 ***"ISSUES in SCHEDULING MULTI-STORY BUILDINGS"*** introduces several important issues and concepts that should be included in the planning and scheduling of this type of construction projects. The chapter discusses production rates and continuity issues as important scheduling decision factors that control the schedule. The chapter also discusses vertical logic constraints in scheduling repetitive work.

Chapter 6 ***"PROJECT PLANNING with RESOURCE CONSIDERATIONS"*** presents the limited resource-constrained problem in scheduling of construction projects. The chapter describes the approaches and techniques that have been developed to deal with the problem and focuses on both resource leveling and limited resource allocation.

Chapter 7 ***"WORK SPACE SCHEDULING CONCEPTS for MULTI-STORY***

BUILDINGS" introduces the space-based scheduling model proposed by the research for using work space as a new constraint in making scheduling decisions. The chapter first describes a formalized procedure to evaluate work space and availability. The chapter also describes a step-by-step scheduling procedure to implement work space as a constraint in scheduling activities of the repetitive segment of multistory projects. A detailed worked example is presented to elaborate the theory presented and proposed in the chapter.

Chapter 8 ***"The SCaRC SCHEDULING SYSTEM"*** discusses the overall SCaRC scheduling system. The chapter presents a general description of the structure and components of the system and the function of each component. A detailed description of the SCaRC components and its scheduling modules and processors is provided in Appendices A, B and C.

Chapter 9 ***"THE DATABASE SYSTEM"*** describes in detail the database component of the SCaRC system. The chapter presents the different categories of data and information needed by the system that should be defined in the database system. The chapter also describes the file structure of the database system.

Chapters 10 ***"SUMMARY and CONCLUSION"*** provides a closing chapter of this dissertation. A summary on the work presented by the model and the developed SCaRC system is provided. The benefits and contributions of the research along with future recommendations are outlined. A conclusion on the overall dissertation is included.

2.0 LITERATURE REVIEW

The development of KBESs (Knowledge-Based Expert Systems) for planning and scheduling has been initiated by many researchers and practitioners. There are many completed and ongoing research efforts aimed at the utilization of KBESs in construction planning and scheduling (Generation and Analysis). Research work in this area has concentrated on a KBES application that combined one or more of the following applications:

1- KBESs for activity duration estimation;

2- KBESs for logic (or network plan) generation;

3- KBESs for schedule generation;

4- KBESs for project monitoring;

5- KBESs for schedule analysis and evaluation;

6- KBESs for support of construction planning.

The third category 'KBESs for Schedule Generation' is the one that is mostly relevant to this research. This chapter presents a review and description of the different research efforts in the construction field regarding the implementation and development of knowledge-based expert systems for generation and analysis of plans and schedules.

Several KBESs for planning and scheduling that has been developed are reviewed. These systems are: Sipe [Wilkins, 84], Sipec [Kartam and Levitt, 89], Oarplan, [Darwische, 89], Janus [Axworthy, 89], research at the University of Illinois [De La Garza, 88] and [Echeverry et. al., 89], Construction Planex [Hendrickson et. al., 87], Ghost [Navinchandra et. al., 88], Priority Ranking [Chang and Ibbs, 90], and Know-Plan [Morad and Beliveau, 91]. The following sections presents a detailed description of the objectives and approaches of each system.

2.1 SIPE and SIPEC

SIPE (System for Interactive Planning and Execution Monitoring) [Wilkins, 84] is an advanced general purpose (classical) planner. It uses hierarchical, constraint-based, interactive planning to achieve greater domain independence. It is an interactive system where the user is able to watch and guide the planning process. The system has a rich set of formalism for knowledge representation, including both frame-based and logic-based representation.

SIPEC is a construction planning Customized SIPE [Kartam and Levitt 89]. The system implements SIPE to plan a multi-story building with repeated cycles of operation from a description of the components of the facility.

In developing SIPEC, the developers used the frame representation of SIPE to store the information of the different building components that are part of a facility. The installation of the different components is performed by generic operators attached to the frames representing the components. The installation of a component by an operator becomes an activity.

SIPEC produces plans using a least-commitment approach. This approach aims to delay decisions concerning ordering links and variable instantiations until the system has as much useful information as possible for making such decisions.

In generating the activity plan, SIPEC employs two approaches. The system utilizes general principles, that can be extracted from experienced human planners, and physical laws. Modeling such general principles using proper representation techniques allows the generation of specific activities and their sequence. SIPEC also utilizes generic operators that can be applied to several objects to define specific construction activities. These construction operators are generic in that they can be applied to different individual elements. SIPEC relies mainly on these generic operators to generate the construction plan.

For formulating the input for the structural elements, for example, SIPEC utilizes five generic operators: Do-CF, Do-Column, Do-Beam, Do-Deck and Do-EW [1]. All the activities for the construction of the building structure can be described by the application of these five operators to specific elements of the structure. Another fifteen operators are used for the finishing, mechanical and electrical activities.

Information for all the operators of the system comes from three sources: object hierarchy, the world model and constraints specified in the description of the operators. First, the object hierarchy contains information about various members organized in classes, sub-classes, and instances. Second, the world model contains information about the topology of the members. Such project-specific knowledge are currently provided to the system manually by a user. Research efforts are also for looking at ways of generating this knowledge directly from a CAD system. Finally, the specified constraints contain information that should be satisfied for each specific activity. These building constraints allow the planner to determine the construction logical sequence for all the elements of

---

[1]

Do-CF :to build a column and its associated footing at ground floor.
Do-Column : to build a column at first floor and subsequent floors.
Do-Beam : to build a beam.
Do-Deck : to build a deck
Do-EW : to build an exterior wall.

the facility.

SIPEC uses two dependency principles for generating much of the needed logical sequence for the office building: supported-by and enclosed-by. For those activities not covered by one of these two principles, explicit predecessor activities were given, such as would be input to conventional sequencing method based upon CPM or PERT algorithms.

SIPEC does not account for resource requirements and resource limitations in producing the logical sequence plan of the facility.

2.2 OARPLAN

OARPLAN (Object-Action-Resource Planning System) is a prototype knowledge-based system for generating construction plans. The prototype is developed by A. Darwiche, R. Levitt and B. Hayes-Roth [Darwiche, 89]. The OARPLAN takes as its input a description of the facility to be constructed and generates a hierarchical project plan for construction of the facility. The system was implemented using the BB1 blackboard environment running under common LISP on a T1 Explorer.

The OARPLAN project combines the generality of the general-purpose planners, such as STRIPS [Fikes 71], and the high level performance of the domain-specific planning systems such as GHOST [Navinchandra 88] and CONSTRUCTION PLANEX [Hendrickson 87]. This combination generates project plans based on facility description. The core planner in OARPLAN does not have any domain-specific knowledge. It only knows about the general structures of activities and knowledge sources (KS's) and how to apply KS's. To generate its construction plans, the user must supply the OARPLAN system with extensive knowledge for a class of construction along with the hierarchies of its objects, actions, and resources constituents and about spacial, topological, temporal and other relations that may exist between them.

OARPLAN is based upon the notion that activities in a project plan can be viewed as intersections of their constituents: objects, actions, and resources.

In an attempt to elaborate more on OARPLAN, the following sections describe the system's representation of the facility, the plan and the activities in the plan along with the system's reasoning capability (plan generation).

2.2.1 Representation used in OARPLAN

Facility description representation: The prototype OARPLAN system takes as input a description of the facility entered by the user. The user describes the facility in the form of component classes (such as: floors, beams, and walls along with further classification of these components) and the relationship of the components with one another. These inter-component relationships are derived from the geometry and topology of the components in the facility and are used to derive part of the precedence logic ('Hard Logic'). Such relationships include: supported-by, enclosed-by and adjacent-to relationships.

An effort to extract component description directly from CAD models is currently being done at Stanford University. The goal is to create a knowledge-based database interface system between OARPLAN and a CAD system containing a description of the facility for which a plan can be generated. This will allow OARPLAN to format high level queries to the CAD system used, and to operate across multiple CAD systems.

Plan representation: In OARPLAN, the generated plan is represented by a collection of activities related to each other by different types of relations. One type of relation is sequential dependency relation that defines the proceeding and succeeding activities to any one activity. Other types of relation reflect the different levels of abstraction of a project plan. Such relations include: sub-activity and super-activity.

Activity representation: The means of representing an activity in OARPLAN is adopted from the PIPPA system [Marshall 87]. Marshall defined an activity as an action that applies to a product and that needs resources. In OARPLAN, <actions>, <objects> and <resources> are constituents that define any activity. For example, painting a wall can be defined as the action <paint> being applied to the object <wall> using the resources <paint, ladder,painter>.

In the current prototype OARPLAN system, only action and object constituents are represented and used in the reasoning process; resources will be considered in future versions of the system.

Actions are either simple or compound. Simple actions are performed directly without refinement (e.g. Formwork Erection). Compound actions can be elaborated to lower level ones (e.g. Placing concrete can be elaborated to pouring, screeding, finishing and curing.

Similarly, object constituents can be simpler compound. Classifying an object as being simple or compound depends on the level of reasoning required to be performed by the system. In the prototype, a compound object is defined as a collection of other compound or simple objects known as its components. OARPLAN defines two types of compound objects: zone and assembly. If the components are contained within a defined space, then the compound object is a zone, e.g. a building floor. If the components are parts of a physical object, then it is an assembly, e.g. a concrete beam.

2.2.2 Plan generation in OARPLAN

In contrast to some other knowledge-based systems such as CONSTRUCTION PLANEX, OARPLAN embodies a top-down approach for creating activities. The system starts with a high level activity such as <construct><building-1> and work down using different knowledge sources (KS's) until it reaches an activity scale that the user has defined to be appropriate. The different knowledge sources (KS's) contribute to the development of the plan by either elaborating each activity or posting some sort of dependency constraints onto it.

Activity elaboration: OARPLAN has elaboration KS's which reduce the level of abstraction of higher level activities creating multiple levels of a plan. Such elaboration may or may not introduce orderings among the elaboration set.

Elaboration knowledge sources break-down higher level activities into more detailed activities in different ways. KS's can reduce the scale of the object constituent of the activity while the action constituent remains constant. As an example, elaborating activity <construct><building-1> may result in <construct><floor-1>, <construct><floor-2>, ...etc. Such knowledge sources are known as generic KS's and they apply to activities which can vary across projects.

On the other hand, other specific KS's reduce the scale of the action constituent, while the object constituent of the activity remains the same. Elaborating activity <construct><beam> may result in the activities <pour><beam>, <finish><beam>, and <cure><beam>. As apposed to generic KS's, such specific KS's apply to activities that are constant through different projects. Constructing a beam remains constant across projects in terms of the activities that elaborate it.

Activity dependencies: The current OARPLAN system infers dependencies in two ways.

The first is utilizing predefined dependencies that are inherited from activity sub-plans, such as placing concrete. These are of the type that are constant across projects and about which little reasoning is needed. The second is by inferring dependencies through applying KS's which reason about the constituents of activities.

As stated earlier, OARPLAN only utilizes action and object constituents in its activity representation. Activity dependencies between the different project activities are thus created as a result of the interaction among their constituent actions and the presence of relations between their object constituents.

Dependencies that result from relations among action constituents are reflections of methodological or technical dependencies,

> ' Inspecting something has to happen after installing it, curing concrete has to come after finishing it and so on... '

Those that result from object relation reflect the spatial and topological description of the building. Current object relations are: supported-by, adjacent-to, enclosed-by and in-same-floor, among others. The current OARPLAN system utilizes mainly object relations to create most of the logic relationships between the project activities.

Current relations between activities include only : before and after relations; other important relations to be further implemented in the system is: requires, causes, and lags.

The first two relations are similar to 'after' and 'before', except that they force the related activity to be included in the plan, if it is not included. These two relations are of importance as they represent a way to introduce activities into the plan. OARPLAN starts with a plan that has activities at the facility component level and elaborates the plan from these activities. However, supporting activities, such as: scaffolding and clean-up, are not directly related to any of the project's components and can not be elaborated using the 'top-down' approach. Hence, these two relations, requires and causes, can be used to introduce such supporting activities. The third relation requires some lag between the two activities.

2.2.3 Limitations and Future Extensions

OARPLAN generates the needed activities in a project plan using a 'top-down' approach by elaborating a high level activity such as <build><building> into a set of project activities at one or more sub-levels of detail, guided by activity scale reduction and activity sub-plans.

The current version of OARPLAN deduces relations among the generated activities by reasoning about attributes of, and relationships among actions and objects. The system has no knowledge sources to allocate resources to activities, nor to compute their durations. The developers expect future versions to the system will incorporate these capabilities.

Another future extension to the current version would allow the system to retrieve project information directly from any CAD system without any user input.

2.3 JANUS

JANUS is a knowledge-based system for simulating and analyzing time-cost tradeoff issues on facility projects. The system was developed at Stanford University by Alan Axworthy and Raymond Levitt [Axworthy, 89]. JANUS takes as input an initial schedule and produces as output a modified schedule, reflecting the results of the time-cost tradeoff analysis. The input schedule must include activity description, precedence relations and estimates of the man-hours required for completion.

The JANUS system tackles the time-cost tradeoff issue from a motivational as well as a technical point of view. JANUS uses activity sequence, time and cost relationships and a set of generic expediting tactics for its time-cost tradeoff analysis. It also represents project participants' views (i.e. owner's view and contractor's view) by explicitly modelling the relationships among them to derive the actual time-cost tradeoff facing each project participant. Each activity has a responsible participant who determines the value of time and therefore the maximum amount that may be spent expediting the activity.

JANUS will perform the following tasks: 1) calculate traditional critical path method, and 2) determine activity durations, activity crew preference, alternate crews, feasible expediting tactics and activity ownership.

The prototype system is implemented using IntelliCorp's KEE^{TM} knowledge processing environment.

2.3.1 JANUS System Architecture

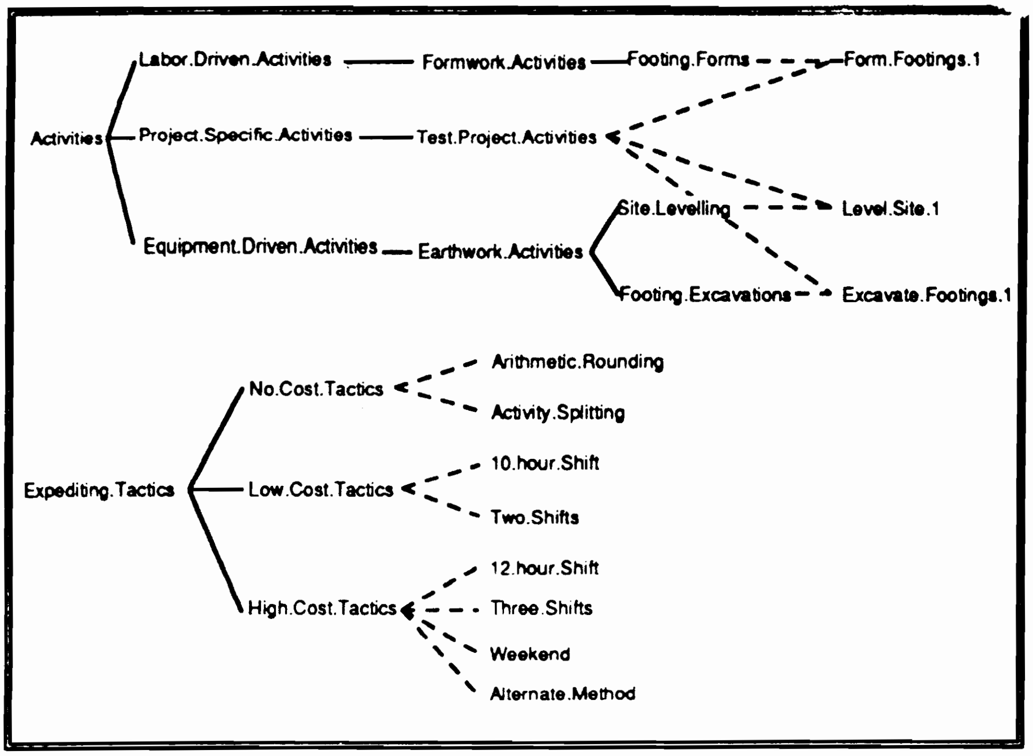
The research focuses on two primary functions: 1) determining the value of time (dollars/day) for the various project participants, and 2) matching expediting tactics with critical activities to shorten the overall project duration.

JANUS uses hierarchical frame description of the participants, their contracts, the project, its activities and available resources along with production rules to control the analysis. Figure 2.1 shows portions of the frame hierarchies for activities and expediting tactics[2]. Tactics are categorized in three groups: 1) no cost tactics, 2) low cost tactics, and 3) high cost tactics.

No cost tactics involve network manipulations, including arithmetic rounding of durations and activity splitting. The application of these tactics is independent from any other considerations and is performed on each critical activity before any other tactics are considered.

Low cost tactics and high cost tactics involve additional cost endured. However, they differ from each other in two ways: basic hourly cost necessary for each tactic, and the

[2]Tactics in the context of JANUS, relate to the specific ways of shortening the activity duration.



Hierarchical frame representations of project activities and expediting tactics are depicted. Subclass relationships are indicated by solid lines, instance relationships by dashed lines. These hierarchies are incomplete but include some useful concepts. Note the multiple inheritance of all instance activities receiving project specific data from Project.Specific.Activities and other data from their generic activity parents.

Figure 2.1 - Examples of Frame Hierarchies

(adopted from [Axworthy, 89])

efficiency of the tactic. High cost tactics involve many hours of high premium work, ie. time-and-a-half or double-time. Low cost tactics typically involve a base rate of straight time, supplemented by some

amount of overtime or shift premium. A cost slope of 50% of an activity's normal cost per day is selected as the dividing line between low cost tactics and high cost tactics. Figure 2.2 shows an example of a frame representing a specific expediting tactic.

2.3.2 Reasoning

The factors affecting a participant's perceived value of time include its daily indirect costs, its capital structure, its contractual relation with other participants and its continuing relations with other participants.

The time cost tradeoff analysis utilizes a 'favorite crew' for each activity. This crew represents the most efficient means of performing that activity. If the combination of activities and selected crews satisfies project goals, then the analysis discontinues. If not, activities are selected for expediting and tactics are applied for calculating new durations and costs. Analysis continues until either the goals are satisfied or no critical activities can be expedited for less cost than their responsible participant puts on the value of time.

Activities are selected for expediting based on their criticality and their location in the network, with priority given to early activities and to those constraining many others.

10.Hour.Shift

Calculate.Coefficient	method	(from Expediting.Tactics)
Calculate.Duration.Reduction	method	(from Expediting.Tactics)
Calculate.Premium	method	(from Expediting.Tactics)
Crew.Hours.per.Day	10	(from 10.Hour.Shift)
Cost.Premium.Percent	0.067	(calculated locally)
Cost.Slope.Coefficient	-0.333	(calculated locally)
Duration.Reduction.Percent	0.25	(calculated locally)
Effective.Hours.per.Day	10.0	(calculated locally)
Efficiency	1.0	(from 10.Hour.Shift)

This figure shows some of the slots in the frame 10.hour.Shift, which is a member of the class Expediting.Tactics. Note the data-holding slots Crew.Hours.per.Day, Cost.Slope.Coefficient, Efficiency, etc. and the method or procedure-holding slots Calculate.Coefficient, etc. Calculate.Coefficient is inherited from the root class, Expediting.Tactics, Cost.Slope.Coefficient is calculated by one of the inherited methods and Efficiency is defined locally.

Figure 2.2 - One of the Expediting Tactics Frames With its Slots.

(adopted from [Axworthy, 89])

Expediting tactics selection is dependent on the state of the analysis and on the amount of schedule reduction still required. In general, 'no cost' tactics are employed first, 'low cost' tactics if 'no cost' tactics provide insufficient schedule reduction and 'high cost' tactics only when all else fails to achieve project goals. An activity is not expedited if the cost of the future reduction exceeds its responsible participant's perceived value of time.

2.3.3 Future Work

The developers anticipate future work to include extending the complexity of the participants' relationship analysis, and extending the range of feasible expediting tactics. Future work may also include the integration of JANUS with other systems that support other aspects of project management, e.g. generating plans, revising activity durations, etc.

2.4 Research at the University of Illinois

Two research efforts that have been completed at the University of Illinois are discussed. The first research effort [De La Garza, 88] involved an analysis and evaluation of construction schedules. The second research effort [Echeverry et. al., 89] focused on the production of construction schedules for mid-rise building construction. Both researches are discussed in more detail in the following sub-sections.

2.4.1 Analysis and Evaluation of Construction Schedules

This research work was done by J.M. De La Garza and W. Ibbs [De Le Garza, 88] at the

University of Illinois to develop a prototype knowledge-based expert system for the analysis and evaluation of construction scheduling networks of mid-rise construction from an owner's perspective.

Two versions of the system were produced. At the early stages of the research, a first implementation was done at USA-CERL (US Army Construction Engineering Research Laboratory) using Personal Consultant Plus (PC Plus) expert system programming environment on a personal computer (PC) and TI Professional . The final improved version of the prototype was implemented using Inference ART (Automated Reasoning Tool) development environment on a TI Explorer LISP machine.

The knowledge base consisted primarily of scheduling decision rules, general construction knowledge, and project-specific knowledge.

> Scheduling decision rules assures that the construction schedule complies with imposed general requirements, logic, time and cost constraints. General requirements rules consist of rules for verifying activity numbers, activity descriptions and codes, government activities, participation of major subcontractors, etc. Time-related rules verify the project completion date, the adequacy of activities' durations and procurement lead time, etc. Logic-related rules check to insure that submittal activities precede related construction activities, that procurement activities proceed material approval activities, and so forth. Cost-related activities insure the adequacy of cost-related issues in the schedule. Knowledge to revise the remaining estimated duration of unfinished activities and rules to identify risk occurrences are also incorporated.
>
> General construction knowledge represent generic expertise that may be applied to a variety of projects. This included knowledge about the effects of weather, crew sizes, and production rates. For example, a winter-sensitive activity should not be scheduled in a period when ambient temperatures are expected to be

below specified minimums.

Project-specific knowledge includes knowledge specific to the project under consideration.

In building the knowledge base, several elicitation techniques were utilized. These fell into three categories:

1) Textbook scheduling knowledge: Available texts and technical manuals were analyzed in order to determine the breadth and depth of the construction schedule analysis domain.

2) Direct interviews with scheduling experts: Several direct expert interviews, using dialogue acquisition techniques, were conducted to elicit and formalize the expert's knowledge.

3) Observing experts' performance while at work on specific problems: A knowledge acquisition experiment was conducted to observe several project managers while they interact given a specific problem.

Four sources of construction knowledge expertise were approached to contribute to the knowledge acquisition stage: 1) contractors (W.E. O'Neil Construction Company), 2) owners (senior representatives from the US Army Construction Engineering Research Laboratory, USA-CERL), 3) consultants (Pinnell Engineering Inc.), and 4) in-house (several faculty members in the Department of Civil Engineering at the University of Illinois).

<u>Prototype KBES environment</u>

1- <u>PC Plus-based KBS</u> Data from a relational database management system, user-supplied project-specific information, and a knowledge base supply the necessary input to this version of the prototype knowledge-based system.

As shown in Figure 2.3, standard user supplied initial schedule or project progress data are first processed by Primavera (a microcomputer based project management system) to generate five customized reports. These reports contain information necessary from which inferences are drawn. The reports are then automatically loaded onto dBASE-III plus (a microcomputer based relational database management system).

The database environment is designed to accommodate the imported Primavera data along with other user-defined databases with non-project dependent information. Such information include hourly rates, productivity rates, cost data, contractor information, etc. A statistical module is also incorporated to contrast project progress data against the original project plan.

Neither Primavera, nor dBASE III Plus, nor PC Plus has access to each others data structures.

Communication among the three programs is done by means of different external files that can be exported and imported by these programs.

A sample example of the steps applied when PC Plus needs to retrieve a value from the schedule database or when it requires performing a calculation using the same external database is shown in Figure 2.4.

Upon completion of developing this prototype version, it was realized that the sequence of steps applied by PC Plus was inefficient. At the time of implementation, the PC Plus versions lacked database-like relational capabilities and a pattern-matching language. Using this configuration for the system, it was clear that the function of the PC Plus rules is mainly to decide which command file to select and execute from the database files and it was evident that the prototype could have been implemented using dBASE III Plus alone without using PC Plus as the anchor.

2- <u>ART-based KBS</u> This improved version of the prototype was implemented using ART on a TI Explorer. Communication between the LISP machine and an IBM-PC/AT, housing Primavera, was established across a serial line and controlled by LISP functions.

Primavera's output schedules are represented in two customized ASCII reports. The first

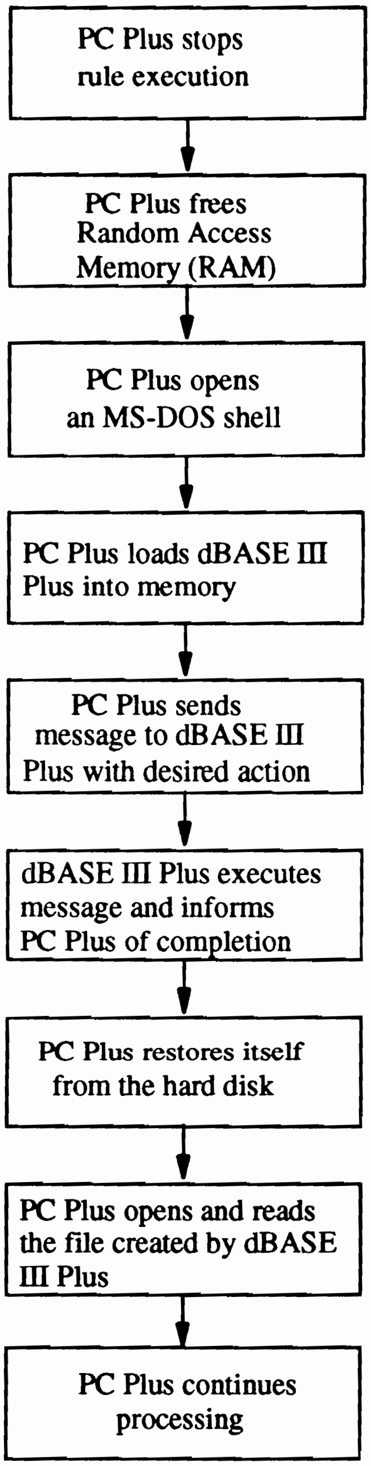

PC Plus stops rule execution
PC Plus frees Random Access Memory (RAM)
PC Plus opens an MS-DOS shell
PC Plus loads dBASE III Plus into memory
PC Plus sends message to dBASE III Plus with desired action
dBASE III Plus executes message and informs PC Plus of completion
PC Plus restores itself from the hard disk
PC Plus opens and reads the file created by dBASE III Plus
PC Plus continues processing

FIGURE 2.4 - Sequence of Steps Applied in PC Plus

report contains all activity data, except logic. The second report mainly consists of the activity's logic relationships. These two output reports contain the minimum project information from which inferences and alternative scenarios are explored by the prototype KBS.

1) Project information is transferred from the customized ASCII reports to ART via the serial link where it is stored in schemata format.

2) Upon completion of data transfer and storage, a set of rules in the knowledge base of the system is triggered to assign each activity to the corresponding frames in the semantic network.

The semantic network is based on a work breakdown structure (WBS) that is subsequently based on project phases, goals, and organization. This network of attributes consists of different levels that are enter-connected in a hierarchical format with well defined inheritance paths/links among its different attributes. Any specific schedule activity may be attached at any level in the hierarchy, and thus inherits general or specific attributes.

For example, when an activity like "cast in situ 2nd floor slab" is found in the schedule, the KBS will be able to automatically deduce a series of attributes about such activity: i) contains all basic schedule parameters, ii) represent a slab in the superstructure, iii) is made of cast in situ concrete, iv) consists of formwork, reinforcing steel, concrete placing curing and stripping, etc.

3) Using other rules of the knowledge base, the inference engine can modify, if necessary, activity attributes of the original input schedule. Hence, generating a new schedule data structures.

4) The inference engine sends these new data structures to a CPM-Kernel for processing (re-scheduling).

CPM-Kernel is a rule-based and frame-based framework that has been written using ART and Common LISP to provide basic features of a PMS within the same KBS computing platform. The purpose of such system is to evaluate alternate schedule scenarios resulting from inconsistencies found in the original imported schedule. Having such module embedded in the same environment of other module would mainly 1) increases efficiency 2) using the same computing language allows the PMS module to share the same object-oriented representation, data-structures, and procedures of the evaluation modules of the KBS, 3) avoid sending modified project data across a communication serial link every time the schedule needs to be re-calculated.

The CPM-Kernel consists of a project scheduling system and a resource conflict resolution system. The project scheduling system apply CPM techniques through various phases to re- evaluate a project schedule and generate a possible alternative solution. The resource conflict resolution system follows to allocate resources to tasks and attempt to solve resource conflicts.

5) New results are returned and inserted in the appropriate schemata, and appropriate processing continues. Hence, original schedule evaluation is not limited by any number of schedule re-calculations by ART that remains maintaining control during the whole process.

2.4.2 Plan Generation of Construction Schedules

This research work involved the implementation of a prototype knowledge-based system for the generation of plans and schedules of mid-rise construction of commercial nature. The system was developed at the University of Illinois by D. Echeverry, C. Ibbs and S. Kim [Echeverry et al, 89] and was implemented using Intellicorp KEE development environment. It utilizes rule-based and frame-based knowledge representation techniques for the generation of the logic among the different activities.

The prototype system models the relevant elements of the building construction process for scheduling purposes. The system performs the following: defining the set of activities to build a particular project, sequencing of the activities, allocation of resources, determination of activity durations and a verification of the satisfaction of constraints (weather, imposed milestones, etc.) and of work continuity for trades operating through a sequence of repetitive tasks.

The approach followed in developing the system encompassed three major phases: 1) knowledge acquisition phase, 2) knowledge formalization phase, and 3) knowledge implementation phase.

The knowledge acquisition phase consisted of interaction with construction schedulers for acquiring mid-rise construction scheduling knowledge. Four construction firms were contacted and involved. A formal program session of knowledge acquisition was performed.

Two main approaches were used to acquire the knowledge from the experts: 1) Direct observation of the experts while in the process of producing the schedule of a 10-story building from a complete set of plans and specifications. This technique was time consuming and was only performed with two of the four selected construction firms. 2) Direct discussion based on the examination and analysis of already available schedules that the experts had generated in the past for real projects. This technique was used with all four firms.

Adopted Approach for Schedule Generation

As shown in Figure 2.5, the schedule generation process is executed in two stages: a

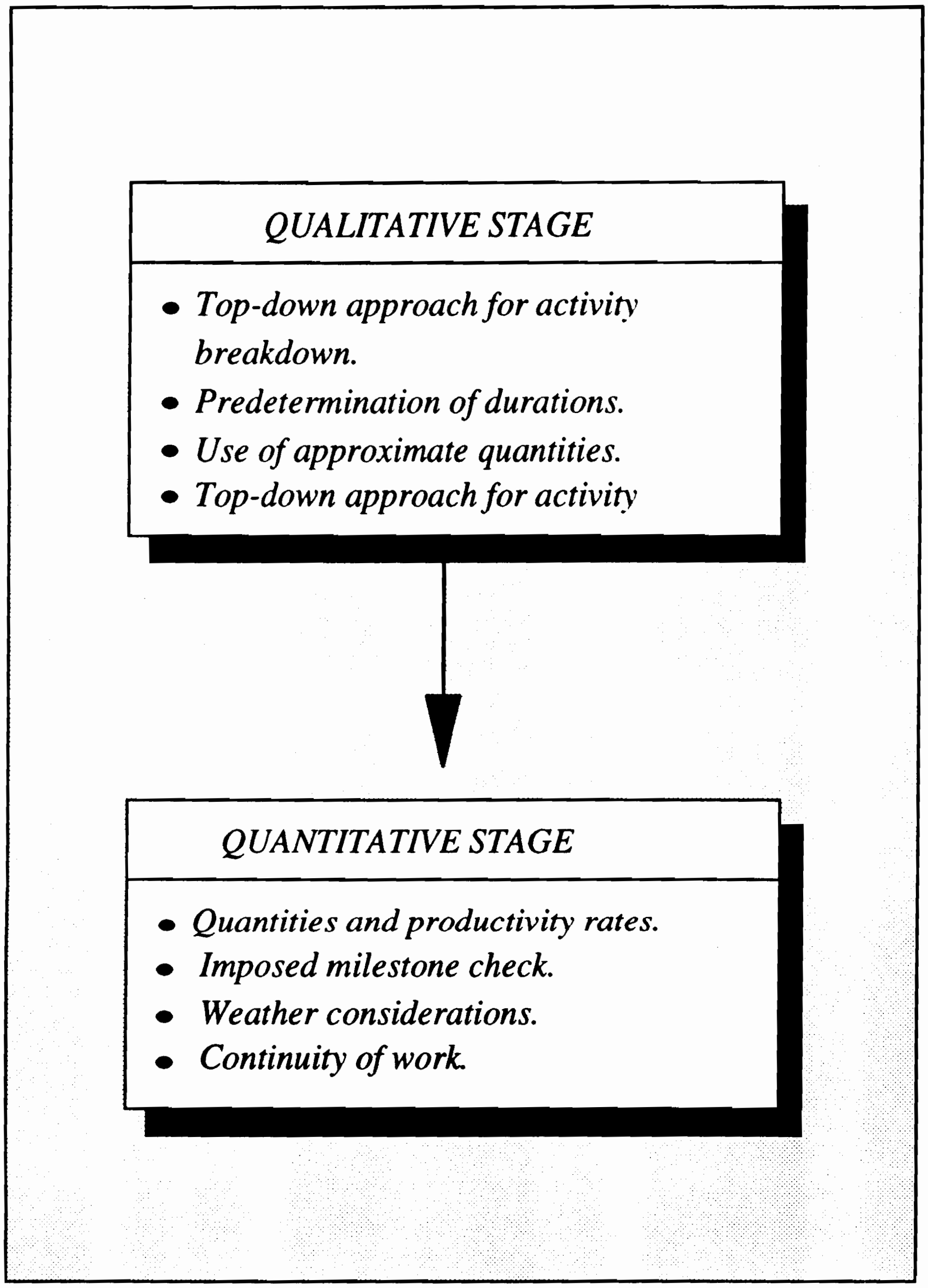

QUALITATIVE STAGE
• Top-down approach for activity breakdown.
• Predetermination of durations.
• Use of approximate quantities.
• Top-down approach for activity
QUANTITATIVE STAGE
• Quantities and productivity rates.
• Imposed milestone check.
• Weather considerations.
• Continuity of work.

Figure 2.5 - Stages for Schedule Production

Qualitative Stage and a *Quantitative Stage*.

In the qualitative stage, the system follows a top-down approach to breakdown the construction of the project into activities and sub-activities. The concept of work area is employed here for performing the breakdown[3]. A preliminary sequence (logic) of the activities along with their preliminary durations are also determined in this stage. The system determines preliminary activity durations utilizing approximate quantities with the objective of producing an adequate pace of construction for the overall construction process.

In the quantitative stage, quantities of work are associated with crew productivity rates to verify the preliminary durations predetermined in the qualitative stage. The schedule is then

checked against imposed milestones and weather constraints. The main goal in this stage is to maintain the flow of the major trades and to avoid interruptions in the work of crews performing repetitive activities.

Logic Generation

In generating the logic[4], the system uses four major factor/issues that determine

[3] Typically, a work area is associated with a story (e.g. floor) and a system (e.g. concrete pouring 2nd floor). Under certain circumstances, the prototype system carries out a break down of the work area into smaller work areas.

[4] The system generates preliminary logic in the qualitative stage, then generate the final logic in the quantitative stage.

precedence relationships among activities:

1- Functional (physical) relationships among building components - Four main physical relationships are considered in the prototype: i)supported-by, ii) covered-by, iii) weather-protected-by, and iv) embedded-in. This type contributes to most of the sequencing rules as it governs the sequencing of activities describing the installation of building components (e.g. columns 1st floor, paint 5th floor, etc.).

2- Trade interaction - Interaction of trades (crew and equipment) affects the scheduling logic as the performance of one trade affects that of another. Four cases of trade interaction are considered in the reasoning process of the prototype: i) occupies-same -space-as, ii) provides-service-to, iii) may-damage, and iv) affects-environment-of.

If a crew is competing for space with another crew or a piece of equipment, a sequence has to be established. Similarly, if a crew is providing a service to another crew, the latter has to follow the one providing the service (e.g. electrical crew providing power for elevator testing crew). In the case in which the installation of a building component damages another component, a precedence relationship has to exist (e.g. wall painting may damage the carpet). A clear sequence has to be established as well if the installation of a component affects negatively the environment (e.g. spraying of fire proofing material).

3- Resource limitations - Competition for resources necessarily forces non-parallelism in the execution of the competing tasks. The prototype considers the effects of limited resources, tasks competing for he same resource are not scheduled in parallel.

4- Code regulation - These are regulations imposed on the construction process dictated mainly by safety standards.

Knowledge base organization:

The knowledge base of the system comprises several modules that interact following a blackboard architecture. One module consists of a hierarchical breakdown of the different building systems into building components to provide project information. Another module supports a hierarchical breakdown of the construction of the building into activities. A third module contains the knowledge to determine activity logic. A fourth module determines preliminary activity durations. The knowledge to perform allocation of crews and that checks the preliminary schedule against different criteria is contained in another module. A procedural module, responsible to perform CPM calculations, is also written in KEE following an object-oriented approach.

Future Extensions

This work was focussed to the generation of plans and schedules of mid rise construction of commercial nature. Future expansions are expected to include:

1) Support of the schedule control process;

2) Coverage of other types of construction;

3) Retrieval of project information (geometric data) directly from CAD models of the facility under consideration.

2.5 CONSTRUCTION PLANEX

CONSTRUCTION PLANEX is a knowledged-based expert system developed at Carnegie Mellon University by C. Hendrickson, C. Zozaya-Gorostiza, and others [Hendrickson et al., 87] and [Zozayz et al., 89]. The main objective of this effort were to formalize the knowledge utilized for scheduling, and to embed it into an automated assistant to generate more detailed and accurate networks. Focus was on the excavation and the structural erection of concrete or steel-frame buildings.

CONSTRUCTION PLANEX integrates different elements of the construction planning process into a unified modelling and planning system. To generate a process plan, the system formulate activity networks, selects technologies and construction methods (select crew type and determine number of crews to be used) to perform construction activities, estimates activity duration and costs based on the selected technologies and construction methods, and prepares project schedules using a CPM algorithm. The system features a user interactive GANTT schedular that allows the user to calculate, display and modify scheduling information.

The system was implemented using KNOWLEDGE-CRAFT from FrameKit and Common Lisp on three different machines: a Texas Instrument Explorer, a SUN 3/60 and an IBM PC/RT computer.

2.5.1 Generation of Activity Network

In generating a project activity network, CONSTRUCTION PLANEX uses the "bottom-up" activity formulation approach. As opposed to OARPLAN, CONSTRUCTION PLANEX starts with activities at the component level, and aggregates upward. Figure 2.6 (adopted from [HENDRICKSON et al., 89]) depicts the formulation model adopted by the system. The activity network is built in four steps:

<u>Step 1</u>: Model Building - The building is described in terms of unitary components called 'design elements'. Example components are beams, columns, footings, etc.

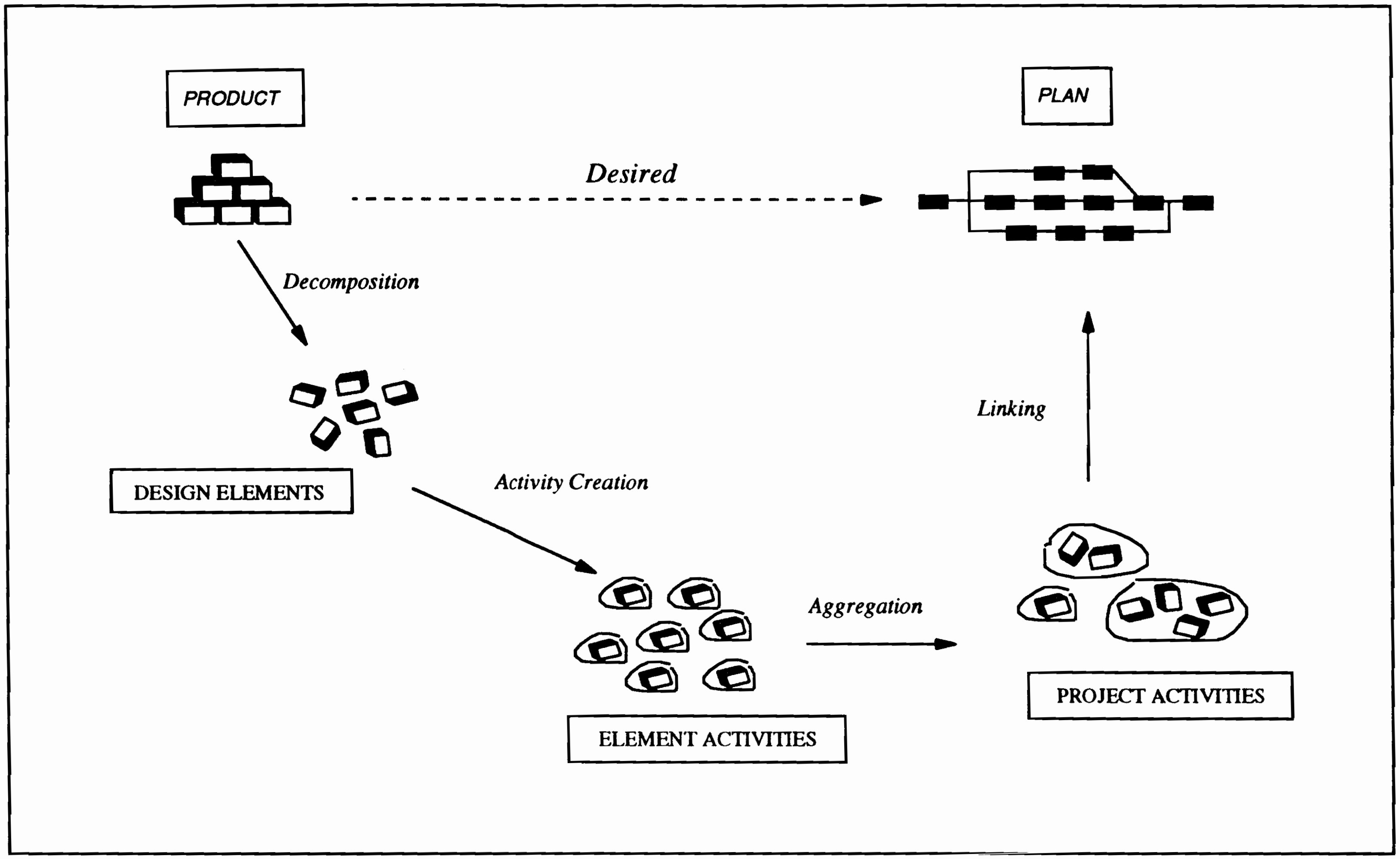


Figure 2.6 - Bottom-Up Activity Formulation Model Used by CONSTRUCTION PLANEX.

(adopted from [ZOZAYA, 1989])

Step 2: Identify Activities - The system determines the element activities required to construct each design element. Example element activity is form placement for individual concrete column.

Step 3: Aggregate Activities - The system aggregate element activities into project activities. Project activities represents group of element activities on a particular floor of the building. Example project activity is form concrete placement on a particular zone (floor or section).

Step 4: Link Activities - The system creates a project network by establishing precedence links among project activities.

CONSTRUCTION PLANEX generates activities associated with the construction of the design elements. To generate other activities, artificial design elements must be created (e.g. an object describing the geometric characteristics of the site is needed to generate a site-clearing activity). Also, the system identifies the activities and establish precedent relationships separately. Alternative knowledge sources could perform these tasks simultaneously.

2.5.2 System Architecture

CONSTRUCTION PLANEX has three essential parts: a context, an operator module, and a knowledge base.

1- The Context contains information on the particular project under consideration, including the design, site characteristics, the activities involved in the project, and the resources available to perform these activities.

Information is stored in a series of hierarchically organized frames forming a representational structure. Each frame is linked to parent or children frames from which information can be inherited. Frames are named and contain various slots to record information. CONSTRUCTION PLANEX uses three basic representational structures shown in Figure 2.7:

- Tree of Design Elements;
- Tree of Element Activities; and
- Tree of Project Activities.

The facility is described in terms of individual design element schemas (frames) that represent building components such as beams or columns. These schemas are aggregated by material (e.g. concrete or steel), element type (e.g. columns), and location (e.g. first floor). Activities used to construct each of the design elements are aggregated, using a special coding system, into a tree of element activities represented by element activity schemas. Construction project activities are aggregated by the type of activity represented (e.g. formwork) and the type of design element associated with the activity (e.g. formwork for columns versus formwork for beams).

Frames of design elements are created from basic input to the system. Example 'concrete footing' design element frame is shown in Figure 2.8(a). Design element frames contain four type of slots:

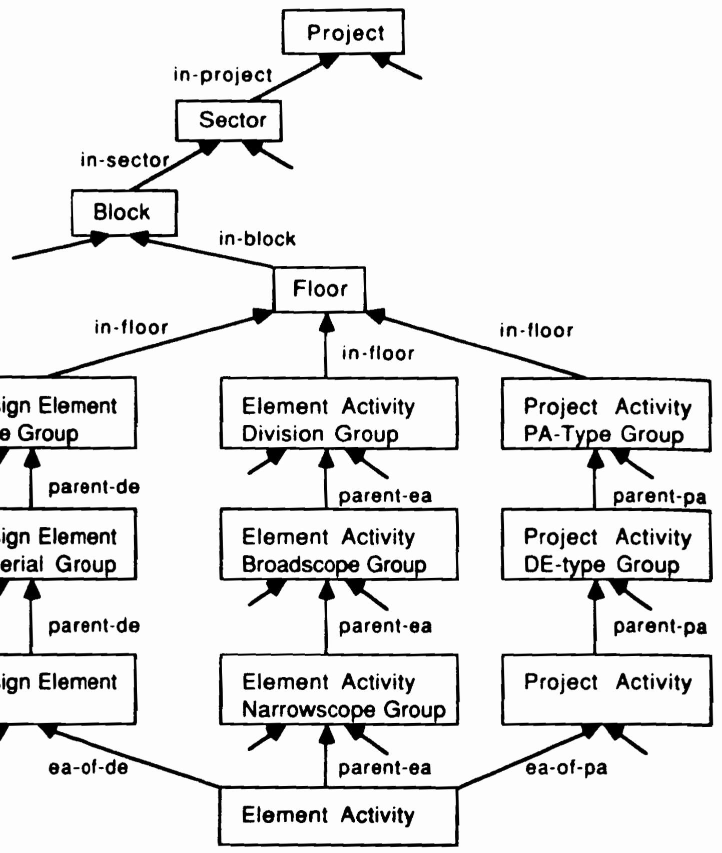

Project
in-project
Sector
in-sector
Block
in-block
Floor
in-floor
in-floor
in-floor
Design Element Type Group
Element Activity Division Group
Project Activity PA-Type Group
parent-de
parent-ea
parent-pa
Design Element Material Group
Element Activity Broadscope Group
Project Activity DE-type Group
parent-de
parent-ea
parent-pa
Design Element
Element Activity Narrowscope Group
Project Activity
ea-of-de
parent-ea
ea-of-pa
Element Activity

Figure 2.7 - Representation Structure of Construction PLANEX

Adopted from [ZOZAYA, 89].

DESIGN-ELEMENT p01-s00-b00-f00-de-60-01-01	
SLOT	VALUE
is-a	de
name	column-footing
name-code	60
type-element	01
number-element	01
project	p01
sector	s00
block	b00
floor	f00
construction-type	cast-in-place
concrete-type	normalweight-3000
re-steel-density	rsd-1
xg-coordinate	10
yg-coordinate	10
zg-coordinate	-4
xl-coordinate	10
yl-coordinate	8
zl-coordinate	-1.50
x-angle	0
y-angle	0
z-angle	0

a) Design Element

ELEMENT-ACTIVITY p01-s00-b00-f00-ea-02-220-10-01	
SLOT	VALUE
is-a	ea
ea-name	excavation-column-footing-01
ea-code	02-220-10-01
ea-of-DE	p01-s00-b00-f00-de-60-01-01
parent-EA	p01-s00-b00-f00-ea-02-220-10
ea-of-PA	p01-s00-b00-f00-pa-10-60
amount-of-work	24.0
unit-of-measure	cu-yd
crew	excavation-foundation-05
material-package	none
duration	16 hours
successors	p01-s00-b00-f00-ea-02-220-10-02

b) Element Activity

PROJECT-ACTIVITY p01-s00-b00-f00-pa-10-60	
SLOT	VALUE
is-a	pa
pa-name	Excavation-Foundation-p01-s00-b00-f00
pa-code	10-60
parent-PA	p01-s00-b00-f00-pa-10
pa-has-eas	p01-s00-b00-f00-ea-02-220-10
amount-of-work	720.0
unit-of-measure	cu-yd
crew	Excavation-Foundation-05
number-crews	1
material-list	none
duration	480.0
successors	p01-s00-b00-f00-pa-20-60
succ-lags	16 hours
EST	day 15
LST	day 15
EFT	day 75
LFT	day 75

c) Project Activity

Figure 2.8 - Sample Frames in CONSTRUCTION PLANEX

Adopted from [ZOZAYA, 89]

(1) Classification slots to identify the type of design element. Different design codes are used to identify the different characteristics of the design elements.

(2) Location slots to specify in what project, sector, block and floor the design element is located.

(3) Geometry slots that describe the geometric characteristics of the design elements.

(4) Specifications slots that contain descriptive information about the element that is relevant to the planning process. (e.g. Type of concrete is a useful specification for determining the appropriate construction technologies).

During operation of the system, and in the process of generating the network, element activity frames and project activity frames are created to represent activities and decisions.

For example, for the 'column footing' frame in Figure 2.8(a), the system will automatically create a frame to describe the excavation activity required for the column footing, Figure 2.8(b). Attributes describing the element activity frame include:

(1) Classification slots to identify the type of element activity.

(2) Hierarchy frame that specify the relationship between the element activity frame and other frames in the context.

(3) Quantity take-off slots that contain information about the amount of work and the unit measure of the element activity.

(4) Specification slots that describe other information relevant to the activity such as the type of material package that has to be used.

(5) Technology-decision slots containing information about technology choices such as the type of crew to be used. Technology information is inherited from other frames

in the context, but the user can override the value by specifying the technology choice at the element activity level.

(6) Technology-consequence slots that describe information dependent upon the technology choice affecting the element activity (e.g. durations, successors, cost, etc.).

Project activity schemas represent aggregations of element activities. CONSTRUCTION PLANEX creates project activity schemas to obtain a reasonable level of granularity in the activity network that is used as the basis for technology choices and for scheduling purposes. A sample project activity frame is shown if figure 2.8(c). Slots in this frame include:

(1) Classification slots that identify the type of project activity.

(2) Hierarchy slots that specify the relationships of the project activity frame in the context.

(3) Quantity take-off slots that contain information about the amount of work and the unit of measure of the project activity.

(4) Specification slots that describe other aggregated information such as the quantity of materials required to perform the activity.

(5) Technology-decision slots containing information about the type of crews allocated to this project activity.

(6) Technology-consequence slots that describe information such as duration, cost, successors and successor lags.

(7) Scheduling slots that describe the scheduling information such as earliest and latest start times of the activity and milestones imposed on the activity.

<u>2- The Operator Module</u>: CONSTRUCTION PLANEX has a set of problem-solving operators. Each operator is a procedural function that modifies the context by creating

and/or deleting frames, or modifying attributes. There are two types of operators: domain and control operators. Domain operators perform specific planning tasks such as technology choice, activity duration estimation, or scheduling. These operators perform planning tasks by operating on a specific type of context object. Control operators decide the sequence in which domain operators are executed. Control operators are used to implement different control strategies for solving the problem.

CONSTRUCTION PLANEX includes three basic types of problem-solving operators: (1) design element operators, (2) element activity operators, and (3) project activity operators. This classification is based on the type of context objects that the operators take as arguments.

Design element operators: CONSTRUCTION PLANEX has two design element operators that are applied to design elements. One operator aggregates design elements in a bottom-up approach and produces a tree of design element schemas. The second operator creates element activity schemas describing the tasks used to construct a design element.

Element activity operators: CONSTRUCTION PLANEX includes six operators applied to element activity schemas. One operator creates the tree of element activities. Another operator computes the quantity take-off for an element activity. A third operator identifies the unit of measure for the quantity of work associated with an element activity. Selecting the material needed to perform an element activity is accomplished by another operator. Another operator is responsible for aggregating element activities into project activities. A sixth operator estimates the duration of element activities using inherited information from the project activity schema to which the element activity is linked.

Project Activity Operators: There are eight operators that manipulate project activity schemas in CONSTRUCTION PLANEX. These operators perform the following tasks on project activities:

(1) Build the tree of project activities using the bottom-up approach;
(2) Choose crew types;
(3) Compute the quantity take-offs;
(4) Recommend an appropriate duration;
(5) Determine how many crews to allocate to the activities and computes the normal and overtime hours of activities;
(6) Establish precedent relationships;
(7) Computes the leads and lags between consecutive activities;
(8) Estimate activity cost.

In addition, the system includes two other operators that are applied to the set of project activities as a whole. One operator (Floyd-Warshall algorithm) computes the project schedule. It calculates the earliest and latest start and finish times for the project activities. These results are stored in the scheduling slots of the project activities. Another operator determines the net present value of project using scheduling and financial information.

3- The Knowledge Base: The knowledge base of CONSTRUCTION PLANEX is organized in a set of knowledge sources (KSs) that store the knowledge required by the operators. Each KS provides information for particular operations such as activity formulation, duration estimation or precedence determination. As noted previously, the knowledge in these knowledge sources is limited to that for the planning of the excavation and erection of concrete or steel buildings. The following types of knowledge sources are present in CONSTRUCTION PLANEX:

(1) Element Activity KSs describe the set of activities required to construct a design element;

(2) Amount KSs specify formulas used when computing the quantity of work for each element activity;

(3) Unit KSs indicate the default unit of measure of the work quantities for each type of element activities;

(4) Material KSs specify the set of materials used in performing an element activity;

(5) Project Activity KSs specify the name of the project activity associated with a particular element activity;

(6) Technology KSs recommend appropriate crews for constructing project activities;

(7) Duration KSs compute desirable durations for project activities;

(8) Successor KSs specify the names of the successors of particular project activities;

(9) Lag KSs are used to determine the types and values of leads and lags between two consecutive project activities.

An example of an element activity KS is shown in Figure 2.9 as a decision table used in the original acquisition or representation phase.

Name: KS-Example		Type: first				
Object	Slot	Op	Value	Rules		
current-object	type-element	is	cast-in-place concrete column-footing	T	T	F
soil-characteristics	possible-use	is	backfill	T	F	I
excavate-column-footing				X	X	I
dispose-of-excavation-column-footing				I	X	I
pile-up-excavation-column-footing				X	I	I
borrow-material-column-footing				I	X	I
place-forms-column-footing				X	X	I
reinforce-column-footing				X	X	I
pour-concrete-column-footing				X	X	I
remove-forms-column-footing				X	X	I
KS-other-elements				I	I	X

Figure 2.9 - Example of a KS for Element Activity Creation

Adopted from [ZOZAYA, 89]

In addition to these three components, CONSTRUCTION PLANEX contains a menu driven interface used to control the execution of the operators and a Knowledge Source Acquisition Module used to modify the contents of the Knowledge Base.

2.5.3 Output Graphics

CONSTRUCTION PLANEX features a user interactive GANTT Scheduler to provide the user with a useful mechanism for modifying the information stored in the domain objects. GANTT allows the user to calculate, display and modify scheduling information for a particular set of activities by providing the user with means to invoke the execution of domain and control operators directly from the screen.

In addition to the interactive graphics of GANTT, CONSTRUCTION PLANEX includes the following passive output displays:

1) an Activity-on-Node diagram displays the project activity network with some scheduling information;

2) a Cost Curve displays an X-Y plot of total activity cash flow versus time;

3) a Cumulative Cost Curve displays a cumulative cash flows of project activities versus time; and

4) an animation program, called ANIMATOR, provides a graphical simulation of the construction process.

2.5.4 Conclusions

CONSTRUCTION PLANEX demonstrates the applicability of knowledge-based expert systems techniques to the overall problem of construction project planning. It provides a framework which incorporates both the time value of money and value engineering in planning. The system constitutes an operational prototype that could be extended to a full system.

Although CONSTRUCTION PLANEX is the first system that integrates all the elements of the construction planning process into a unified modeling and planning system, and contributes very relevantly to the goal of automating the scheduling generation process, it has several limitations.

a) Presently, the knowledge in the system is limited to excavation and structural erection. Most of the complications of scheduling occur in planning the work of trades (e.g. Finishes and Services). Expansions of the knowledge base to include other aspects of building construction requires that additional design elements and activities be defined.

b) CONSTRUCTION PLANEX considers the type and location of activities during planning but does not reason about the geometry of the element.

c) There is no considerations for the interaction of trades. Also, resource leveling and resource allocation are done manually by hand.

d) The system does not use a formal database management system (DBMS) to store planning data such as unit costs. A DBMS could be used to store and retrieve objects such as knowledge sources.

2.6 GHOST

GHOST (Generation of Hierarchical networks for cOnSTruction) is a prototype knowledge based system for generating construction networks [Navinchandra et al., 88]. GHOST takes as input a set of activities and produces as output a schedule by setting up precedents among the activities. The knowledge base is made up of several knowledge sources known as critics (CKSs). GHOST has the following CKSs: (1) critics that know about physics, (2) critics that know about construction, (3) refinement critics, and (4) critics that check for redundancy. GHOST uses a blackboard architecture that facilitates the integration and interaction of these critic knowledge sources.

In generating a possible plan, GHOST uses its critics to criticize the network rather than building it. Ghost starts with an optimistic schedule network with all the activities in parallel, as shown in Figure 2.10. It then uses its critics to modify the network, by introducing linearizations wherever activities can not be done in parallel. This results in gradual reduction of the parallelism of the network of activities, until all the constraints imposed by the Critics are satisfied.

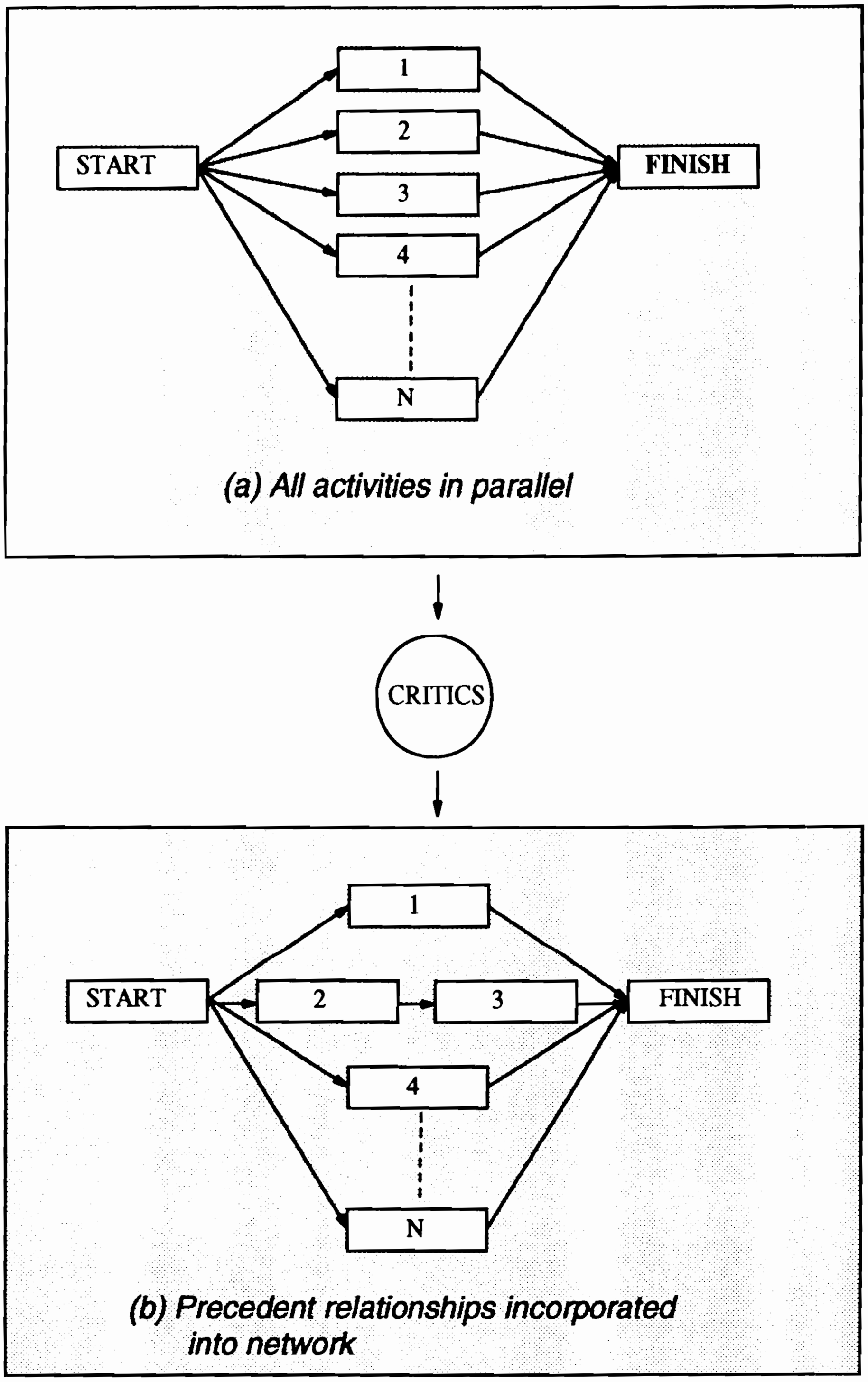

START
1
2
3
4
N
FINISH
(a) All activities in parallel
CRITICS
START
1
2
3
4
N
FINISH
(b) Precedent relationships incorporated into network

Figure 2.10 - Precedence Relationships Introduced by CKSs.

GHOST was implemented using IMST expert system development environment developed at MIT. The prototype is a part of a larger integrated knowledge-based system for construction planning, called CONPLAN, that is currently being developed at MIT.

2.6.1 System Structure

Figure 2.11 (adopted from [Navinchandra et al., 88]) depicts the basic structure of the prototype system. The basic framework of GHOST consists of:

(1) Knowledge base of critic knowledge sources (CKSs).

(2) A Blackboard visible to all the CKSs. The backboard contains a project description and the schedule under develcpment.

(3) The system control mechanism, 'inference engine' that determines the overall order of the problem-solving strategy. The inference engine is also developed in IMST development environment at MIT.

2.6.2 The Knowledge Base

The knowledge base of the prototype system consists of four critics knowledge sources (CKSs):

(1) Critics that know about physics. These critics are represented in an 'IF... THEN' rule format and generate the logical relationships between the different activities based on the physical relationships between their associated objects. For example, if object O2, associated with activity A2, is supported-by object O1, associated with activity A1, then a logical link is created between the preceding activity A1 and succeeding activity A2.

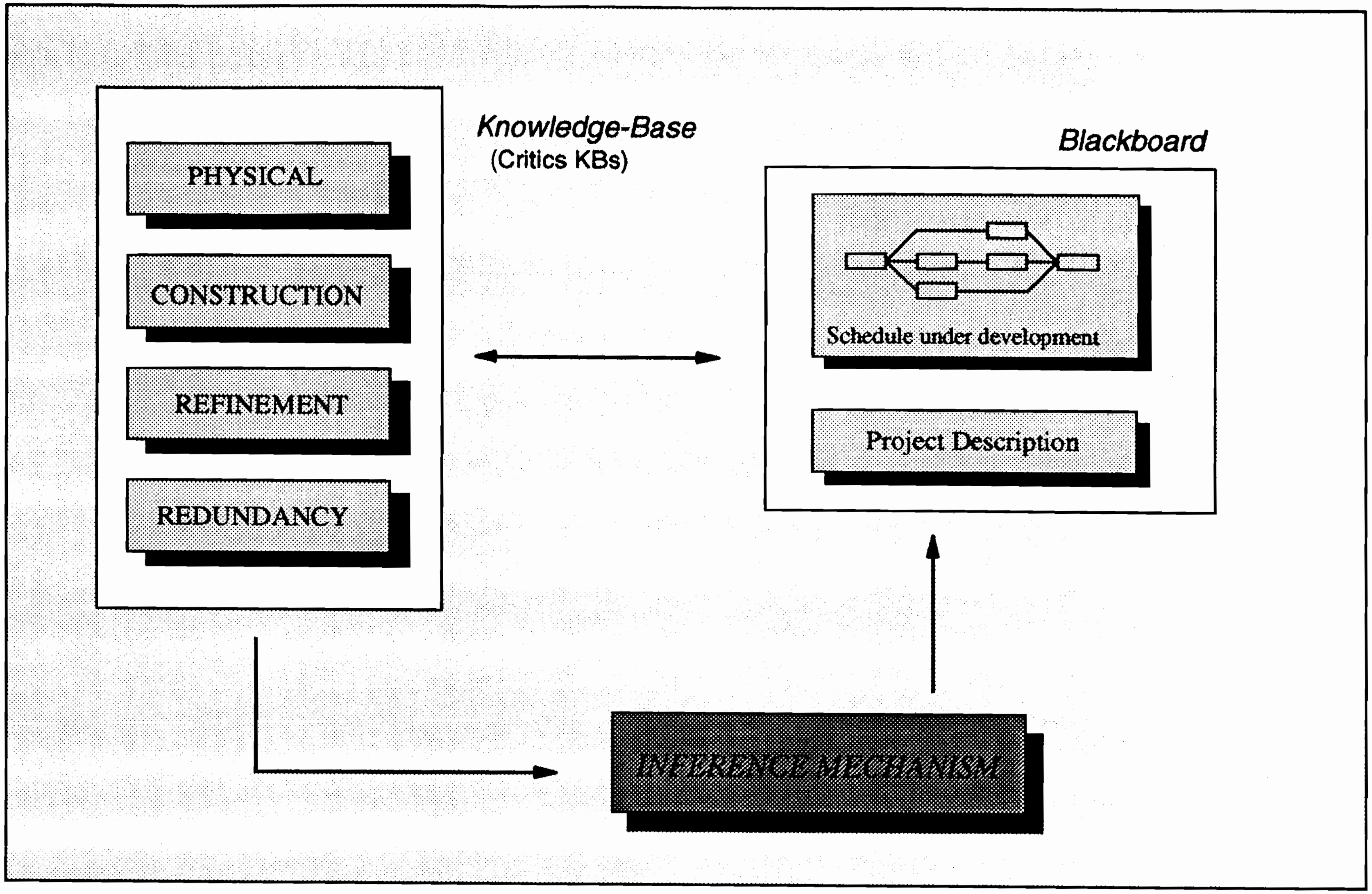


Figure 2.11 - System Structure of GHOST

(2) Critics that know about construction. These critics are

also expressed as rules and incorporate construction knowledge, such as: curing time of concrete, placing of reinforcing bars before concrete pouring, etc.

(3) Inheritance and Refinement critics. These critics perform hierarchical refinement of the general activity network using inheritance characteristics among classes of activities.

GHOST starts with a general network of general gross activities, each associated with a design object (entity) of the facility. The system proceeds to refine the general network by splicing in networks of sub-activities. Sub-networks can be viewed as knowledge associated with more general activities and may be stored with or without precedent relationships. This knowledge is stored in a semantic network structure.

Once all required substitutions are completed, the refinement CKSs remove the general precedence relationships among the affected general activities and setup precedents among the sub-activities. This process is repeated until an appropriate level of detail is obtained.

(4) Critics that check for redundancy. Once all the possible precedence relationships are established between the different activities, these critics 'clean-up' the redundant links and the final project plan is generated.

2.6.3 Logic Generation Approach

GHOST does not extract activities/objects lists from construction drawings, rather it takes as input a list of all these objects and accordingly produces a schedule.

As opposed to other planning systems, were schedules are generated in a step-by-step fashion (e.g. Construction Planex, OARPLAN), GHOST starts with an optimistic schedule with all the activities in parallel. GHOST then apply critic knowledge sources (CKSs) to introduce required precedent relationships between any two activities. A precedent is introduced only when a problem or conflict is found. Other planning systems, on the other hand, use knowledge to look for problems and to decide when actions can or can not be done in parallel.

CKSs can also introduce new activities into the network.This is done by first placing the new activity in parallel with all other activities. CKSs will then introduce appropriate precedent dependencies and, in effect, will place the new activity in the appropriate position in the network.

2.6.4 Limitations and Future Considerations

Currently GHOST does not deal with activity definition or initial activity logic determination. GHOST does not consider 'Soft Logic' in generating the schedule. Trade interaction and limited resources are considered for future research, by the developers, to improve and extend the scheduling capabilities of the system.

Other future expansions to the system include: (1) knowledge for estimating activity durations; and (2) a richer refinement structure.

2.7 PRIORITY RANKING

PRIORITY RANKING is a knowledge-based scheduling system, with fuzzy logic reasoning, for priority decision making in network resource allocation. The prototype system was developed by T. C. Chang and C. W. Ibbs [Chang and Ibbs, 90] at the University of California, at Berkeley. The system was implemented in PROLOG. Some computation functions were written in C and assembly language to accelerate the run-time speed. These functions were linked to PROLOG to ease the development of PRIORITY RANKING.

PRIORITY RANKING uses possibility theory and the generalized modus ponens logic inference rule to measure the impact of external forces (i.e. project-related factors such as weather conditions, change orders, resource delays, resource availability, etc.) and determines a priority ranking for all activities that are candidates for resource assignment at any one point in time. Output results from PRIORITY RANKING, combined with a set of internal factors (i.e. network-related factors such as total slack) are considered as input for a resource allocation algorithm entitled Resource Allocated and Levelled Schedule (RALS), also developed by Chang and Ibbs [Chang et al., 90]. Thus both internal and external factors are taken into account in the resource allocation issue.

Resource Allocation - Preview

A commonly discussed version of the resource-constrained project scheduling problem is that of minimizing project durations under fixed resource constraints. This problem arises when resources required for performance of project activities are available in fixed limited amounts such that the demands of concurrent activities can not be satisfied. Under these conditions, activity sequencing decisions are required, often with a resultant increase in project duration beyond the resource non-constrained duration determined from a preliminary "Critical Path" analysis.

Approaches used to solve this problem may be classified into two categories:

1) Optimization methods that search for the best schedule, and

2) heuristic methods that search for a good schedule.

Optimization methods have been applied and have been successful on at most only smaller sized projects because of the combinatorial nature of the problem. Heuristic methods, which produce 'good' feasible solution, have been successfully applied to real problems. The heuristic or rules used in obtaining such solutions are schemes for assigning activity priorities in making the activity sequencing decisions required for resolution of resource conflict.

[Davis and Patterson, 75] have studied eight different heuristic rules (minimum activity slack, minimum late finish time, resource scheduling method, greatest resource demand, greatest resource utilization, shortest imminent operations, most jobs possible, and random activity relation) for 83 multi-resource constrained projects. Results proved that heuristic methods are suffice to provide reasonable solution to the 'too-expensive' problem.

To date, however, non of these algorithms (optimization procedures or heuristic rules) has the capability to resolve the impacts of the external forces [Chang et al., 90]:

> "... for a practical algorithm, neither the effect of internal factors (i.e. network-related factors) nor the impact of external forces should be ignored..."

2.7.1 Basic concept of resource allocation

In the process of resource allocation, one important issue is determining a priority ranking for all activities that are candidates for resource assignment.

Given a list of criteria that has been established for allocating limited resources in a specified time frame, each candidate activity is checked against the list of criteria to see if the criteria are met. If the criteria are met, the activity is scheduled according to its network constraints. If not, a check is made for the next candidate. In cases where two or more candidates meet the criteria and they require the same resources that are insufficient to start all of them at the same time, priorities must be set before any one of them can be started. Then each candidate will be eventually scheduled according to the priorities.

The criteria can be categorized into two groups: (1) external criteria (project related), and (2)internal criteria (schedule related).

The priority rank P_j for any activity (j) in the candidate set of activities can be defined as follows:

$$P_j = (1 - s_{ij}\, w_i /n)$$

Given a set of criterion C_i, w_i denotes the measure of the weight for importance of the criterion C_i, and s_{ij} denotes the susceptibility of activity (j) to criterion C_i.

The importance does not simply mean that C_i is either important or not important, but rather, as the degree of importance or the weight of importance (fuzzy-set concept). For example, if there exists three criteria C_1, C_2, and C_3, they may be described with criterion C_1 as very important (w_1 = very high), C_2 as more or less important (w_2 = more or less high), and C_3 as less important (w_3 = low).

Similarly, letting the susceptibility s_{ij} of an activity (j) may range between zero, then, when $s_{ij} = 0$, activity (j) is not susceptible to C_i, and when $s_{ij} = 1$, activity (j) is absolutely or totally susceptible to C_i. When s_{ij} = any other intermediate value, activity (j) has some degree of susceptibility to C_i.

The term $(1 - s_{ij}\, w_i)$ can be interpreted as a measure of activity (j)'s priority against criterion C_i. For example, assume that a criterion C_i, severity of rain has been established, and an activity (j), placing concrete, will be done in an open area with no protection from rain. Also, assume that there is a period during which it rains a lot in the area. therefore, the activity is highly susceptible to rain (s_{ij} is large or close to one) because it is exposed to rain. During the rainy period, the weight of the criterion severity of rain is high (w_i is high or close to one). Thus, the priority to schedule the activity under the probability of impact by rain should be low. On the other hand, the weight of the criterion severity of rain is low (w_i is close to zero) during a sunny period. Thus, during the sunny period, the priority to schedule the activity should be high. the term ($1 - s_{ij}\ w_i$) reflects these situations and is defined as a relative value to measure the priority against each criterion.

Using possibility theory, which is an extension of fuzzy set theory, and the approximate reasoning concept, the weight of importance w_i and the susceptibility s_{ij} can be determined. Details of developing these methods can be found in [CHANG, 87].

2.7.2 Knowledge Base

The sample knowledge base of priority ranking contains two major types of rules. The first type is for measuring the weights of criteria. It is a collection of rules like those shown in Table 1(a).

To determine a weight for a criterion, users have to provide linguistic probabilities like those listed in Table 1(b). Seven external criteria (rain, temperature, cash shortage, change order, changed

TABLE 1. If Rain Is F_i then It Is G_i Recommended to Consider Rain as Criterion

Rule (1)	F_i (2)	G_i (3)
1	Heavy	Very strongly
2	Moderate	Strongly
3	Small	Moderately
4	None	Not

(a)

TABLE 2. Probability of Rain in Specific Period

Severity of rain (1)	Probability (2)
Rain is heavy	Not likely
Rain is moderate	Likely
Rain is small	Very likely
Rain is none	Unlikely

(b)

TABLE 3. If Activity j Tends to be A Affected by Rain then Susceptibility (s_{ij}) of j under Criterion of Severity of Rain Is B

Rule (1)	A (2)	B (3)
1	Greatly	High
2	Moderately	Moderate
3	Slightly	Low

(c)

TABLE 4. If Activity j Is A_k Exposed to Rain then Degree of Impact due to Rain Is B_k

Rule (1)	A_k (2)	B_k (3)
1	Almost entirely	Very high
2	Partially	High
3	Slightly	Moderate

(d)

Table 1: PRIORITY RANKING Knowledge-Base Rules

Adopted from [Chang and Ibbs, 90]

conditions, morale, and resource availability) were considered in the sample knowledge base.

The second type of rules is for evaluating the susceptibility of activities to the criteria. It is a collection of rules like those shown in Table 1 (c) and (d) respectively. Users can input information regarding individual activities (for example, pouring concrete is entirely exposed to rain) or relay on default information in the knowledge base.

The inference engine in PRIORITY RANKING includes an algorithm to compute the weight w_i and a fuzzy logic system to find the susceptibility s_{ij}. In addition, the inherent backward-chaining strategy of PROLOG is used to control the fuzzy information stored in the sample knowledge base.

Knowing the weight w_i and the susceptibility s_{ij}, the priority rank P_i for an activity (j) can be computed according to Eq. 1 This P_i can be input in the RALS [Chang et al., 90] algorithm to account for the external forces impact in allocating resources.

2.8 KNOW-PLAN

Know-Plan is a prototype knowledge-based system developed by A. Morad and Y. Beliveau [Morad Beliveau, 91] for generating construction plans. The system combines CAD and 3D computer modelling technology with AI technology to generate and visually simulate construction plans. Know-Plan is implemented using Inference ART-IM (Automated Reasoning Tool for Information Management) on an IBM PS/2.

The system generates a possible plan by reasoning mainly about the geometric data of the different project components that is extracted from the CAD model of the facility, hence, minimizing the user time for data input that would be required, otherwise. The generated plan can be modified by the user by allowing him/her to override any system created relationship (logical link) between any two activities. Hence, the user has the upper hand in deciding on the final generated network.

The prototype also incorporate an object-oriented CPM module that provides basic features of a PMS (Project Management System). Given a possible plan generated by the system, this module calculates start and finish activity dates and generates a schedule report based on user input durations. The module co-exist within the same KBS computing platform.

The model utilizes an advanced visual simulation system (WALKTHRU from Bectel Inc.) to visually animate the construction process. Using the generated plan and the 3D computer model of the project this module provides a graphical display of the construction of the facility.

2.8.1 Know-Plan Computing Environment

The prototype system is comprised mainly of four computing modules:

1) Dynamic sequencer module;

2) Interactive sequence modification module;

3) Scheduling module;

4) Visual simulation module.

These modules interact with a central database management system that contains the necessary database files (DBFs). The central database is designed to provide the interface between the 3D model and the KBS's modules. In addition, the central database provides the user with the tools to manipulate the stored data before the interface process is performed. Data is manually input by the user and consists of geometric data files ,extracted from the CAD model of the facility, in addition to other data files necessary for the reasoning process (e.g. object classes data, zoning classes data, object/class connection type data, etc.).

Object's Geometric Data: The outcome of a design phase, which uses a CAD system, is a 3D computer model of the designed facility. The 3D is then transferred to a format readable by WALKTHRU^{TM}. The geometric data required by Know-Plan is then extracted from WALKTHRU from two source files, a "Volume Definition" file and a "Initial Record" file. The volume definition file provides maximum and minimum values of object boundary coordinates in the x,y,z directions. These data are essential for the geometric reasoning process by Know-Plan for plan generation. The initial record file provides coordinates of the center of the object and the rotation of the object in the x,y,z directions. These data are used by the visual simulation module for the visual simulation.

Each component object in the 3D model is associated with one activity in the project plan generated by the system (e.g. object 'Column B1' is creates one activity say 'Act#21') . Know-Plan does not define different levels of abstractions. The lack of a defined hierarchical structure of the construction activities does not allow Know-Plan to associate one object with many activities or many objects with one activity.

For example, if such hierarchy is present, object 'Column B1' would be associated with activities 'Formwork column B1', 'Concreting column B1', 'Curing column B1', and 'Curing column B1'. Given all the objects' list available from the extracted data files, the system would be able to automatically generate construction activities associated with each object. This hierarchical structure is proposed by the developer as future work.

Central Database: The central database consists of seven database files that provide the system with the necessary data for plan generation and visual simulation. These files are: 1) activity file, 2) class file, 3) zoning file, 4) activity connection file, 5) activity class relationships file, dominating class file, and special class relationship file.

Data is manually input by the user into the central database files and then exported into

ASCII format to be read by the different modules of the Know-Plan system.

2.8.2 The dynamic sequencer module

The objective of the dynamic sequencer is to generate a possible project plan of the facility by reasoning about its geometric data available in the central database. Information about activity classes, activity connection types, zoning data and other data available in the central database is also utilized by the module to generate a possible plan.

From the design elements records of the 3D model, available in the central database, the module creates associated construction activities in the knowledge base. To generate a plan, the system creates precedence relationships (logical links) between any two activities. Links between activities are generated in the form of execution facts which has the following pattern:

(<u>Execute</u> <u>Act#(i)</u> <u>Act#(j)</u> <u>KS</u> <u>Priority-level</u>)

The relationships are viewed as different networks depending on the knowledge source that asserts such relationships. The Know-Plan system is designed to accommodate several knowledge source. Each knowledge source will contribute to the plan generation by a creating logical links between some or all the activities resulting in a single network with different links of different priorities.

In the current version, the system only incorporates geometric knowledge source to create logical links between the different activities. These links are associated with a priority value of (50). Other created links from other future knowledge sources, such as resource constraints, weather constraints, constructability constraints, etc., will be assigned higher priority values to override these generated links. User-created logical links, that is manually input to the system in the format of an execution fact as above, will be assigned the highest priority value to override any other generated link between the same two activities.

Once all the different links are created between the activities, the module proceeds to combine all the generated links into a final network which represents a possible construction plan.

In asserting links between activities, Know-Plan utilizes three geometric-based reasoning rules:

1) Rule#1 to assert logic between activities of the same class in the same zone;

2) Rule#2 to assert logic between activities of the same class in different zones; and

3) Rule#3 to assert logic between activities of different classes and located either in the same zone or different zones.

In the case of objects that are of the same class and located in the same zone, the rule starts by checking if the two objects are overlapping or sharing a common surface, the rule finds which object is higher in each direction of the three coordinate axes: X, Y, and Z. Using the knowledge stored about the direction of installation and the priority of installation of the objects' classes, the rule asserts an execution fact that define a possible logic between the two objects based on their spacial relationship.

In the case of two objects of the same class and located in different zones, the same procedure used in the first rule is used. However, the rule checks only if the two objects share a common surface. If it finds a common surface, the rule proceeds with remaining actions.

The third rule asserts logic between activities of different classes. The rule uses the type of connection between the two objects' classes to define the logic between the two associated activities. According to this approach, the rule first checks which object is higher in each direction: X, Y, Z. Then it checks if their is a special class relation defined in the knowledge-base that match the geometric relationship between the two objects. If a special class relation exists, then the direction of installation between the two classes, to which the two objects are related, is used to assert the logic. If a special class relation does not exist, then the dominating class relation is used to assert the logic.

The geometric reasoning process asserts execution facts of the form

(Execute ACT-1 ACT-2 GEOM 50)

These facts results in the generation of a geometry logic network.

All facts generated from the geometry knowledge source are assigned a minimum priority value of 50. Other future knowledge sources will generate a set of other logic networks and will be assigned higher priority values to override any facts generated from geometric reasoning in the network refinement process. In the network refinement process, all execution facts of all the different generated networks will be combined, to generate a final network.

<u>Network Refinement Process</u>: Once all the execution facts from the different knowledge sources are generated, the system applies a refinement algorithm to generate a final network. This process is executed in two steps:

1) Step 1: Preliminary logic refinement, and
2) Step 2: Final logic refinement.

In the first step, execution facts of lower priority are deleted. Given the two sets of facts below,

```
fact#1  (Execute  ACT-1  ACT-2  KS#1   50)
fact#2  (Execute  ACT-1  ACT-2  KS#2   70)

fact#3  (Execute  ACT-1  ACT-2  KS#1   50)
fact#4  (Execute  ACT-2  ACT-1  KS#1  100)
```

the module will delete facts 1 and 3.

The purpose of the final logic refinement process aims at deleting all links that are already implied by other links (i.e. redundancy links) or detect any looping in the network. In this step, the remaining execution facts are asserted in a schemata hierarchical structure and the inheritance characteristics of ART-IM is utilized to detect any redundancy or looping in the logic.

The outcome of the refinement process is a final possible network plan for the project.

2.8.3 Interactive sequence modification module

The execution of this module is optional where the user is allowed to modify the generated sequence. User defined links (in the execution fact format) is input manually to the system. The module combines these user defined links with the already generated links to come up with a modified plan.

2.8.4 Scheduling Module

The scheduling module computes the early and late start and finish dates for the different activities of the final network. This results in a preliminary scheduling for the project. A final scheduling should take into consideration other issues of project planning, such as: resource allocation and resource constraints, weather constraints, legal constraints, etc. This is not included in the scope of this version of KNOW-PLAN.

The resulting dates from the preliminary scheduling are transferred to the central data base to generate a 'Record file' to be used by WALKTHRU to perform the visual simulation of the construction process.

2.8.5 Limitations and Future Consideration

The research contributes to the body of knowledge by formulating a planning model which uses the geometric data of the different project components as the bases of generating construction plans using a knowledge-based system.

The current version of KNOW-PLAN generates activities that are only associated with the design elements of the project components. The knowledge-base of the system only contains knowledge that reasons about the geometrical data of these design objects or components.

To generate a feasible plan and schedule additional construction knowledge should be added to the knowledge-base in a hierarchical structure format.

2.9 Previous Work versus Current Research

Most of the previous research efforts exhibited by the development of different prototype knowledge-based systems aimed at the generation of the plan and schedule utilizing mainly knowledge about technological relationships among the activities. Several research efforts considered limited resource availability for the plan and schedule generation. None of these research works have investigated or utilized work space as a decision factor in the generation of the schedule.

The current research work investigates the impact of limited work space availability as a new constraint in the schedule generation process. The research integrates work space constraints with resource constraints and other constraints to develop a schedule. A knowledge-based scheduling system is developed for schedule generation.

The research focuses its application on scheduling construction work with repetitive nature. The research domain focuses on multi-story building scheduling and the developed knowledge-based system is applied to the repetitive floors.

Other issues associated with this type of construction is also investigated and considered in the schedule generation process. In addition to limited work space availability and resource constraints, the generation of the schedule takes into consideration continuity issues and productivity rates as important factors that influence the scheduling of work with repetitive nature.

2.10 Chapter Summary

This chapter has presented a general overview of the current research efforts that have been completed in the field of construction planning and scheduling utilizing knowledge-based systems' technology. Several prototype knowledge-based systems that has been developed were reviewed. Description of each system's structure and components were presented. The methodology adopted by each system for plan and schedule generation was also described.

The chapter also presented a brief description of how the work presented by the current research differs from the previous research efforts.

3.0 KNOWLEDGE-BASED SYSTEMS TECHNOLOGY CONCEPTS AND REVIEWS

This chapter presents a general overview of Knowledge-Based Systems (KBSs) technology. This will be an attempt to introduce the fundamental concepts of KBSs and illustrate their role and contribution to the implementation of the proposed model. A brief discussion on general KBSs' concepts is first presented. A description of the general architecture of these systems along with the different methods used for knowledge representation and control strategies employed by such methodologies is then reviewed.

The aim of writing this chapter is to show my understanding and familiarity of the concepts and terminology of this new technology. If the reader is already familiar with such concepts, he or she may wish to skip this chapter without losing any continuity.

3.1 Artificial Intelligence and Knowledge-Based Systems

Knowledge-based Systems (KBSs) have created as much excitement in the computer users community as the emergence of Fortran in the 50's, problem-oriented languages in the 60's and CAD in the 70's.

KBSs' technology originated from a branch of computer science that is referred to as Artificial Intelligence (AI). AI is concerned with a broad range of topics that focuses on the development of computer systems to simulate the processes of problem solving and duplicate the functions of the human brain.

AI can be viewed as computer-based solutions of complex problems through the application of processes that are analogous to the human reasoning process. AI comprises hardware and software systems and techniques that attempt to emulate human mental and physical processes. The mental processes emulated include thinking, reasoning, decision making, data storage and retrieval, problem solving, and learning. The physical processes include human senses and motor skills. AI is also referred to as machine intelligence.

The main objective of AI applications is to enable computers to process information, gain knowledge, and understand their environment. At present, conventional computers process data and produce information. These conventional systems can transform, amplify, modify, and distribute this information. With the addition of AI, the essence of the computer changes to the processing of information and the production of knowledge in the form of recommendations. The task of the AI pioneer is to redefine the computer from a data processor into a knowledge processor. Therefore, an intelligent computer will be one that can collect, assemble, choose, understand, perceive, and know.

Artificial Intelligence can be applied to a wide range of problems. Any problem that does not lend itself to an algorithmic solution is a candidate for AI research. Currently, several areas of application have received attention and varying levels of success in AI research. These areas include: Natural Language Processing (NLP) which involves research into communicating with computers using human communication languages such as English; Knowledge-Based Systems; Speech Recognition; Computer Vision; Robotics research; etc. Figure 3.1 provide an AI tree with its many branches.

As illustrated in Figure 3.1, Knowledge-Based Systems are one example of the AI technologies. KBSs use logical relationships to embody knowledge about a specific domain and perform specialized tasks that typically require human judgement and expertise. They are well recognized for their potential power to replicate human knowledge. A standard definition of KBSs is given by [Gashnig,81]:

> "Knowledge-based expert systems are interactive computer programs incorporating judgement, experience, rules of thumb, intuition and other expertise to provide knowledgeable advice about a variety of tasks."

KBSs contain the facts and procedures representing the rule of thumb (heuristic) decision

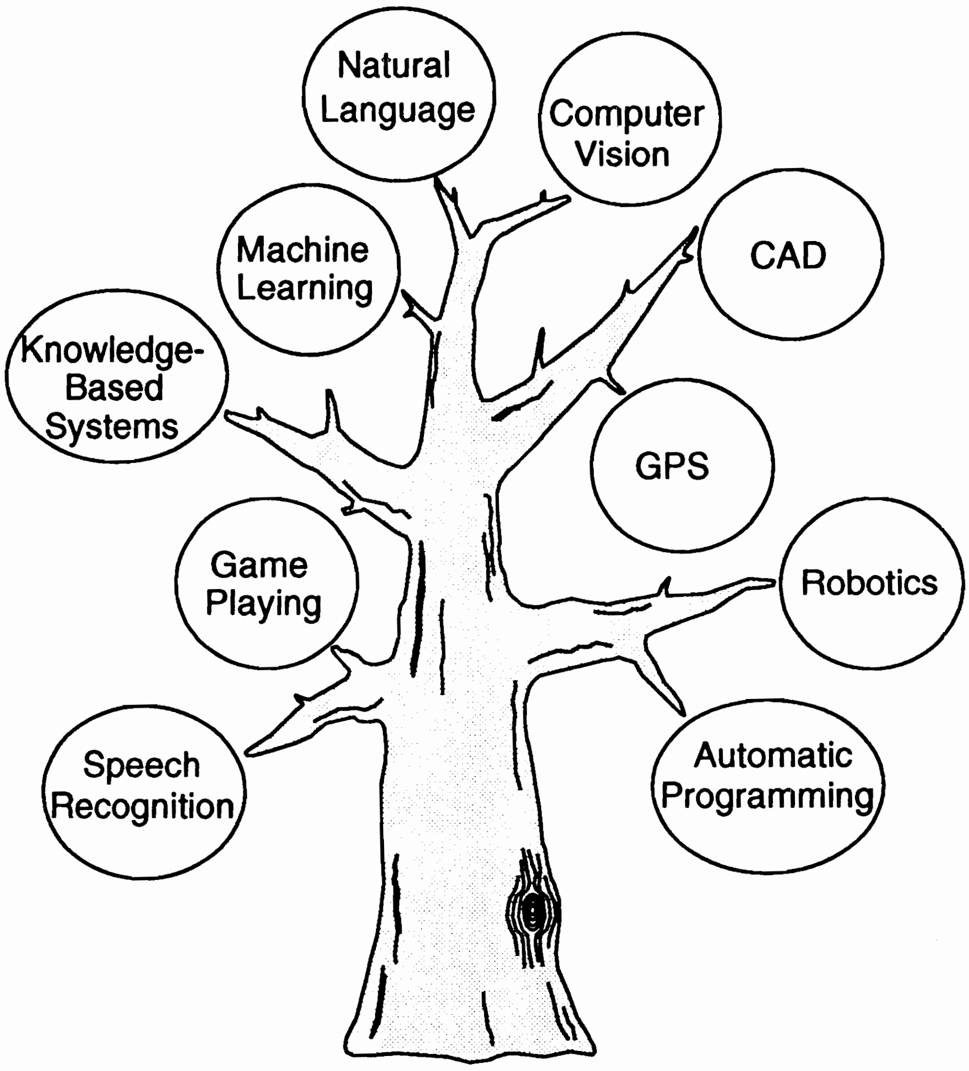


Figure 3.1 - The AI Tree

making processes of an expert. The collection is kept in a knowledge base that is separate from a control program. The architecture of these systems is discussed in more detail in the following section.

Knowledge-based systems are used to improve productivity, preserve knowledge, cultivate understanding and human performance, and help evolve information into knowledge. Most KBSs in current use were developed as an in-house or custom systems. There are very few generic applications available. However, the near future may offer a variety of off-the -shelf systems for both business and home use.

It should be emphasized that KBSs applications are made to supplement the supply of experts, not replace them. Because the knowledge bases of the current group of systems can not automatically update themselves, the knowledge they contain is static. If conditions change or new information is required, the knowledge bases are manually revised. Some of the reasons for KBSs are:

- There exist a decided lack of experts.
- Procedures are becoming complex and burdensome.
- There is little time to groom new experts.
- There is a growing flood of information.
- The competitive edge is difficult to maintain.
- Rapidly made judgements are becoming necessary.

3.2 Architecture of Knowledge-Based Systems

Knowledge-based systems use a wide variety of specific system architectures, primarily because one architecture will be more applicable than another for a given application. As a result, there is no system standard that governs the components, developer screens, or other developmental or user concerns. In spite of the significant differences, most of the architectures have general components in common. These components are: a knowledge base, an inference engine, and a conjecture component. Figure 3.2 shows a block diagram representing the general architecture of a knowledge-based system with typical components.

The Knowledge Base

The *knowledge base* is the key support element to the KBS. The worth and effectiveness of the system are largely determined by its knowledge base. This is because the knowledge base is the component of the knowledge-based system that contains the facts and the heuristic associated with the domain in which the system is applied. Such knowledge permits the program to behave as a specialized intelligent problem solver. The worth of a knowledge base, like the worth of a book, lies in the quality of the contents, not the quantity.

The knowledge embedded in the knowledge base represents the problem-solving identification and solution strategies of those experts who are recognized as superior in their field.

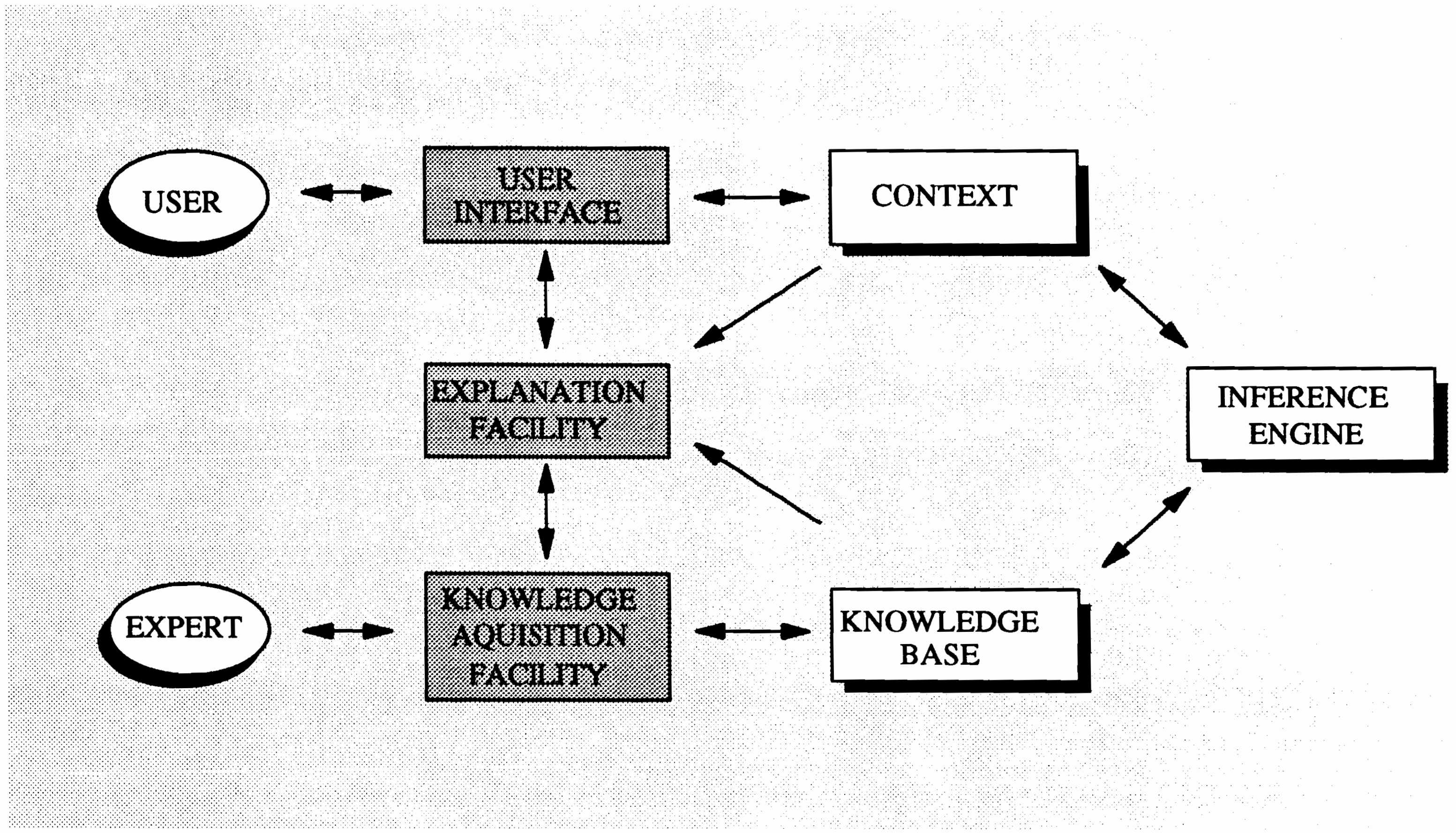


Figure 3.2 - Architecture of a Knowledge-Based System

Adopted from [Maher, 87]

The facts possessed by an expert are collectively called declarative knowledge. The actions constitute the procedural knowledge. Together, the declarative and procedural statements are the codified representation of the expert's skills, experience, education, and training.

Representation of the domain knowledge in the knowledge base can be done using many knowledge representation alternatives. These alternatives include production rules, frames, and other forms of knowledge representation. For example, the combination of the declarative and procedural knowledge forms rules. A complete knowledge base may contain from a few to several thousand rules. Knowledge representation is discussed in section 3.3 below.

A key feature of knowledge-based systems is that the knowledge base is independent of the inference engine, or control program, that accesses it to solve problems. Should new information become available, or knowledge stored become absolute, it is relatively easy to make necessary changes.

The Inference Mechanism

The *inference mechanism,* also called the control program or rule interpreter, is the part of the system that contains the control information. Once the knowledge base is in place, a rich resource exists. However, the resource is of little value without a program to access the knowledge. Once the knowledge is accessed, it is channeled to problem solving. This inference mechanism controls the reasoning process and uses the knowledge base to modify and expand the context. It directs the system in its use of the knowledge base and implements the most appropriate strategies or reasoning processes for the problem at hand.

The inference engine runs the whole show. Its two basic functions are *inference* and *control*. Inference refers to examining the rules and performing the pattern-matching, while control refers to the search techniques and sequence in which rules are examined. The search techniques might eliminate alternatives or search for a match. The inference engine can also interact with the user at the beginning of a session, or when information is needed and is not found in the knowledge base. The entire process is conducted by the system in a manner that is transparent to the user.

Conjecture Component

The third of the conceptual components of a knowledge-based system is the conjecture component. The conjecture component in many systems has four parts. The user interface, a context, an explanation facility, and a knowledge acquisition facility.

The ***user interface*** facility is responsible for interactions with the user. It allows the system to accept information from the user and translate it into a form acceptable to the remainder of the system. It also accepts information from the system and convert it to a form that can be understood by the user.

Interactions with the user are accomplished through a variety of input devices such as keyboards, mice, and touch screens. The input devices activate selections in displays of graphic icons, windows, and forms. Output to the user include explanations, display of the rules used in the reasoning process, on-line helps, graphic representation of critical components, etc. Through the user interface, the user must be able to communicate the problem, provide required and requested information, and interpret the system's advice.

The ***context***, sometimes called the working memory or global database, is the portion of the computer's memory set aside for keeping track of inputs, intermediate conclusions, and outputs. The inference engine uses the context as a scratch pad to track what's going on in the system.

Initial inputs are stored in the context. As the inference engine sequences through the rules, the conclusion drawn from each of those rules are stored in the context. The inference engine uses these intermediate conclusions as new inputs to search for new matches. At the end of a run, the context contains the entire chain of facts that include not only those entered initially but also those that were concluded along the way to the final reasoning decisions.

The ***explanation facility*** component in a knowledge-based system varies from a trace of execution to the ability to respond to questions about the reasoning process used to develop a solution. By seeing the logical reasoning process, users can better accept the outcome.

Users can also use the explanation facility for debugging reasons. During development, it can serve as a way to get feedback on rule construction and sequence, enabling users to readily test the system on practical problems. The explanation subsystem is also an excellent feature for instructional purposes. Through solving a number of problems and at the same time asking for an explanation on each, users may begin to understand the reasoning process.

The ***knowledge acquisition*** facility is the component that facilitates entering knowledge into the knowledge base of the expert system. In the simplest case, this facility acts as an editor and knowledge is entered in a form acceptable by the software that is used to implement the application system. On a more sophisticated level, the knowledge acquisition facility understands the inference mechanism used and can actively aid the expert in defining the knowledge base.

3.3 Knowledge Representation in KBSs

Two basic kinds of knowledge can be put into a knowledge base: declarative knowledge, and procedural knowledge. Most knowledge-based systems will contain both.

Declarative knowledge, also called "descriptive" knowledge, is primarily a statement of fact about people, places, or things. Declarative knowledge permits the programmer to state information, deduce relationships, and classify objects. Using declarative knowledge, the programmer can not explain anything, but can present truths and their association with each other. In knowledge-based systems, declarative knowledge representation schemes include semantic networks, frames, and production rules.

Procedural knowledge or ***prescriptive*** knowledge, is explanatory; it provides a way of applying the declarative knowledge. With this kind of knowledge, the programmer can show procedures for performing a course of actions. Procedural knowledge recommends what to do and how. Procedural knowledge is represented in knowledge-based systems using production rules and object-oriented programming.

The following subsections will describe some of the different knowledge representation techniques.

3.3.1 Semantic Networks

One of the oldest knowledge representation schemes in AI research is the *semantic network* or *semantic net.* A semantic net is a graphical representation of knowledge that shows the relationship between the objects. Semantic networks are excellent for representing declarative knowledge, particularly which has a hierarchical structure. When the knowledge can be classified or categorized, it is a good candidate for a semantic net.

An example of semantic network is shown in Figure 3.3. The circles are called "nodes" and are used to represent people, places, things, or ideas. The nodes are connected to one another to show relationships. These links between nodes are called "arcs". On each arc is a label that states the relationship between the nodes that it connects. While semantic networks are an excellent visual tool, they can also be programmed into a computer to form a complete knowledge base. By referring to the semantic network of Figure 3.3, most nodes represent an object, but other nodes represent attributes of the related object, such as size, color, or specification.

An important characteristic of semantic network is that some nodes may inherit properties or characteristics from other nodes. Since semantic nets are used to represent hierarchical information, some nodes will be higher in the hierarchy than others. Nodes that are lower in the hierarchy can inherit properties from the nodes higher in the network. This characteristic of a semantic network eliminates the need to repeat information at each node.

3.3.2 Frames and Object-Oriented Programming

A *frame*, like a semantic network, is a structure for representing chunks of declarative knowledge. Frames differ from semantic nets in that the values are grouped together in to a single unit called frame. Thus, a frame houses the composite knowledge as a single set attribute-value pairs, whereas semantic nets require several object-attribute-value triplets to represent the same knowledge.

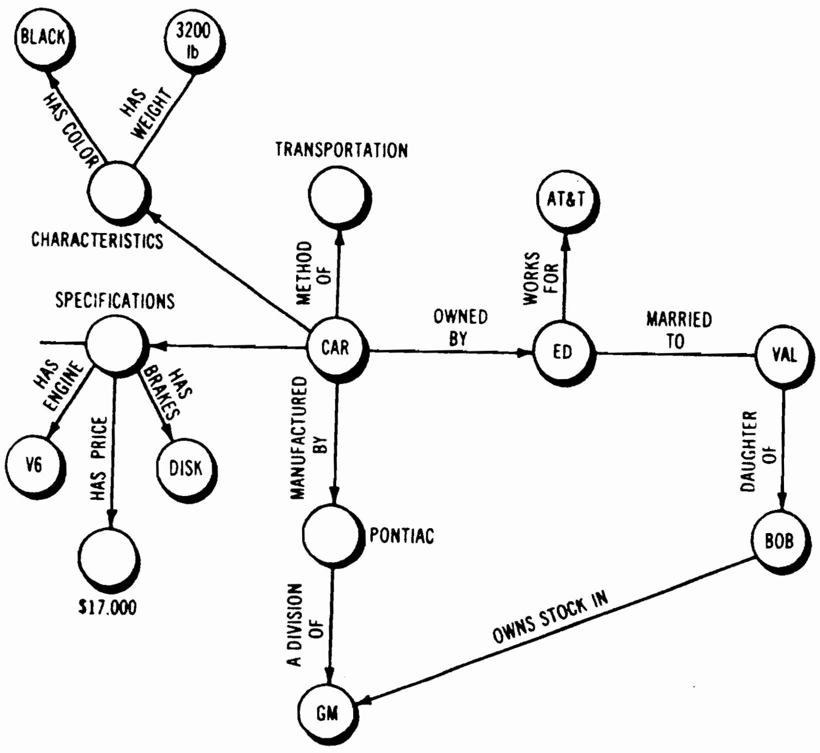


Figure 3.3 - A Semantic Network

Adopted from [Frenzel, 87]

Figure 3.4 shows the concept of a frame. A frame is divided into components called slots and facets. The slot contains an object attribute. These attributes represent a particular characteristic, specification, or other datum used to define an object or situation. Each slot contains one or more facets. One facet may be the value of the attribute. Another, may be a default value that can be used if the slot is empty.

Similar to semantic networks, frames can be used to represent hierarchical knowledge. This is possible by linking multiple frames. As shown in Figure 3.5, a particular slot in a frame may reference another frame that contains detailed information about that particular attribute. A slot in that second frame then could reference another, and so on.

When frames are linked together in a hierarchy, as in Figure 3.5, one frame may inherit the properties of a higher-level frame. As with semantic networks, this ability of frames to inherit properties makes knowledge storage more compact and permits in depth reasoning. As with other forms of knowledge representation, inferencing is done by detailed search of the slots and frames.

Object-Oriented Programming

Frames can also provide a structure for organizing procedural knowledge into the facets. A procedure or method designated in a facet can carry out a certain function. This procedure can be activated by, for example, sending a message to the frame and the appropriate slot that contains the method. If the slot does not have a method for the

PC/2 MODEL 50	
SLOTS	FACETS
Type	Personal Computer
Manufacturer	IBM
Model	PC/2 Model 50
RAM	1 MB
Floppy Disk Size	3.5"
Hard Disk Size	20 MB
Monitor	14" Color RGB
Operating System	PC DOS 3.3

Figure 3.4 - A Frame Describing a Personal Computer

Adopted from [Frenzel, 87]

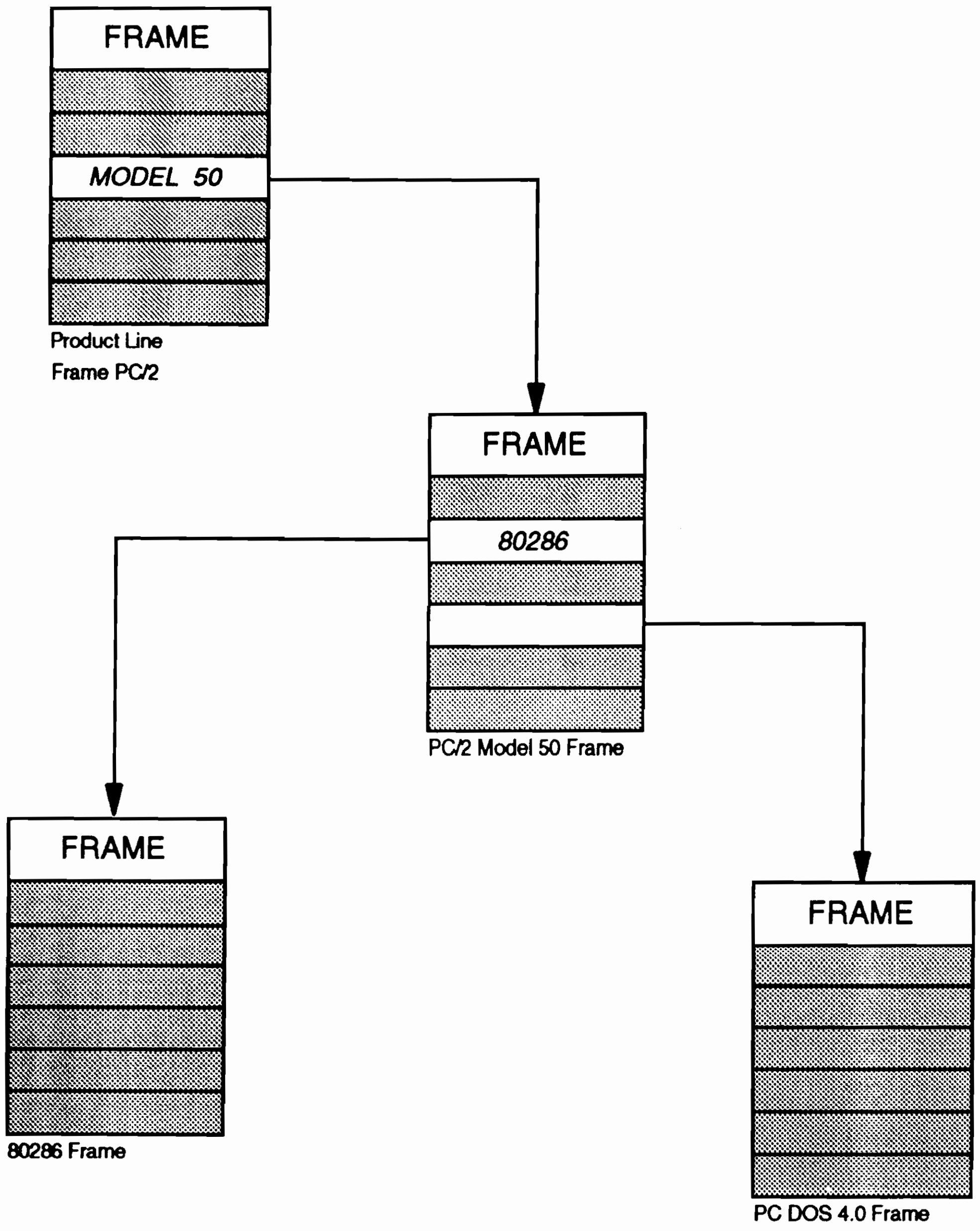


Figure 3.5 - Linking Frames to Represent Hierarchical Knowledge

Adopted from [Frenzel, 87]

received message, it searches to see if it has inherited any appropriate methods from other frames in the hierarchy. Once a method has been found, the frame carries out the actions associated with the method defined in the knowledge base. These actions may result in the firing of rules and/or possibly sending messages to other frames.

This process of defining methods and sending messages to frames to activate such methods is termed *object-oriented programming.*

3.3.3 Production Rules

Semantic nets and frames have their strengths as declarative knowledge representation techniques. Cleverly manipulated, they can also address some procedurally based knowledge as well. However, when the need is to recommend a course of action based on observable events or situations, it may be more effective to use a procedural system for the knowledge representation. This procedural system contains both the situational facts and the appropriate application of those facts to effect an action. *Production rules* are the most common knowledge representation used to date.

Rule Format

The two parts of a rule is a premise and a conclusion, a situation and an action, or an antecedent and a consequent. These statements are written in an *IF-THEN* format. The first part of the rule, generally called the "left-hand" part, is prefaced by the word *IF*, to state a situation or premise. The second or "right-hand" part of the rule is prefaced with

THEN to state an action or a conclusion. Production rules are simple to understand and use and are ideally suited to a wide range of heuristic knowledge. Most knowledge domain are easily represented in this format.

An example of a rule is shown

> **IF** the water level exceeds 5 feet
> **THEN** start the pump

Some types of knowledge require a more complex rule, such as this:

> **IF** the water level exceeds 5 feet
> **AND** it is a holiday
> **THEN** start the pump

Another way to make knowledge statement is to use **OR** statements in the premise.

> **IF** the water level exceeds 5 feet
> **OR** it is a holiday
> **THEN** start the pump

The main benefit of rules is that they facilitate creation, modification, and maintenance of a knowledge base because the knowledge is modularized. Much of the domain knowledge changes over time, new rules must be added and old rules removed or modified to keep the knowledge base current. With rules, these changes can be made quickly and easily.

3.4 Problem Solving Strategies for Knowledge-Based Systems

Given that humans solve problems using various strategies, it is easy to compare and contrast problem-solving techniques for machines. Computers, like humans, require strategies to solve problems. The objective is to find an expedient path through the knowledge base to reach a solution.

The basic problem solving method used in AI is search. Search, as its name implies, is the process of examining a large set of possible solutions to a problem in an attempt to find the best solution. Conceptually, machines deal with three basic elements in searching their knowledge bases in order to reach solutions. These elements are *problem states*, *goal states*, and *operators*. As illustrated in Figure 3.6, the initial state of the problem is supplied by the user through the user interface. The initial goals are predetermined and stored in the knowledge base as results. Operators are procedures used by a system to move from one state to another within the search space. The knowledge base, referred to as the "search space," comprises all final solution to the problem. In a knowledge-based system, the search space is usually a set of IF-THEN rules, and it might also be the nodes and arcs of a semantic network or a collection of frames.

The operators move the problem from one state to another state which (one hopes) more closely matches the goal state. The operators follow the master control strategy set by the system's inference engine. An operator can be as simple as an algorithm, or as complex as a sophisticated search technique.

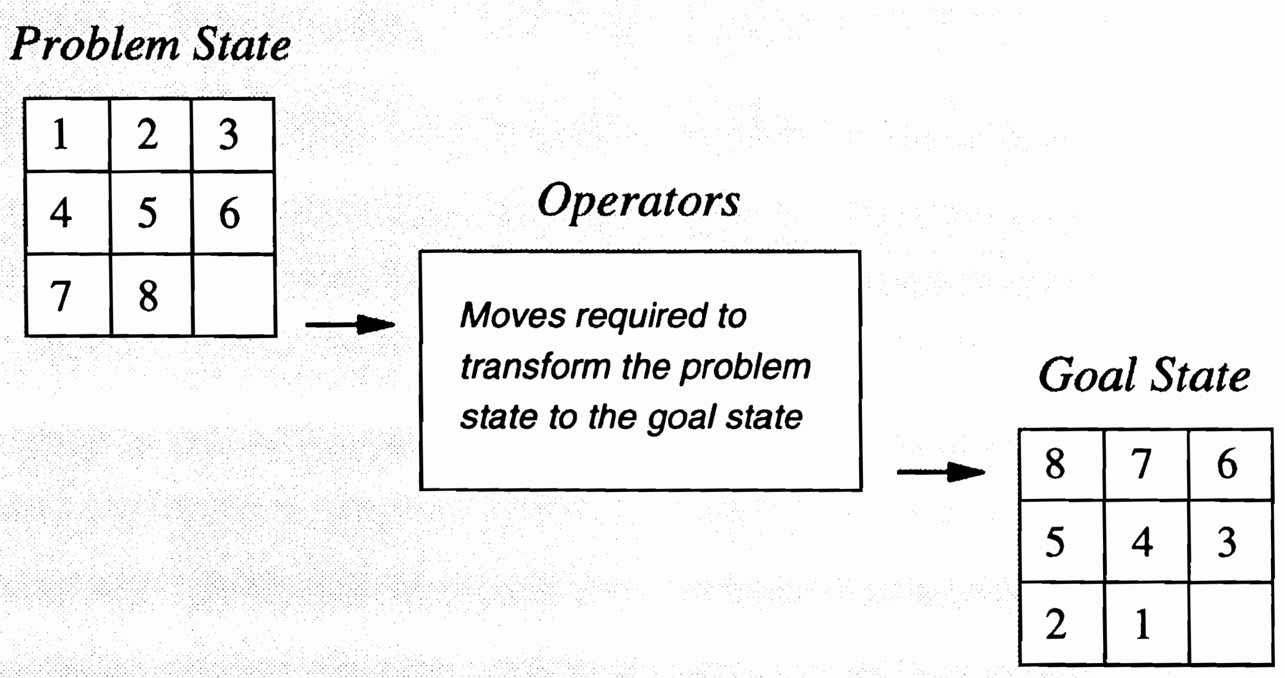

Problem State
1 2 3
4 5 6
7 8
Operators
Moves required to transform the problem state to the goal state
Goal State
8 7 6
5 4 3
2 1

Figure 3.6 - State Space Representation

Search Tree

A *search tree* is a graphical method of representing the search space using nodes and arcs. Each node is a fact, rule, or another knowledge element. The nodes are interconnected with arcs showing the relationships. The resulting diagram often looks like an inverted tree, with roots and trunks at the top and the branches and leaves spread out towards the bottom. A typical search tree is shown in Figure 3.7.

Nodes represent possible states for the system, and whose arcs correspond to operators that change the system from one state to another. The system's initial state is the root of the tree, and the leaves represent final states for the system. The system's control program, or inference engine, starts at the root and tries to find a path through the tree (i.e. a sequence of states) that will reach a leaf corresponding to a (or the) desired state or goal. Often it attempts to find the path that is optimal by some criterion like length or tree search time.

In many cases, the search tree is much too large to exist in its totality anywhere in the computer at any one time. When this is true, the system must generate whatever part of the search tree it needs for its immediate concern. In the case of chess solving, for example, after the first pair of moves are made, the system need worry about only the possible successor states to the opponent's move. It does not have to concern itself with all the positions that could have resulted if the first moves had been different.

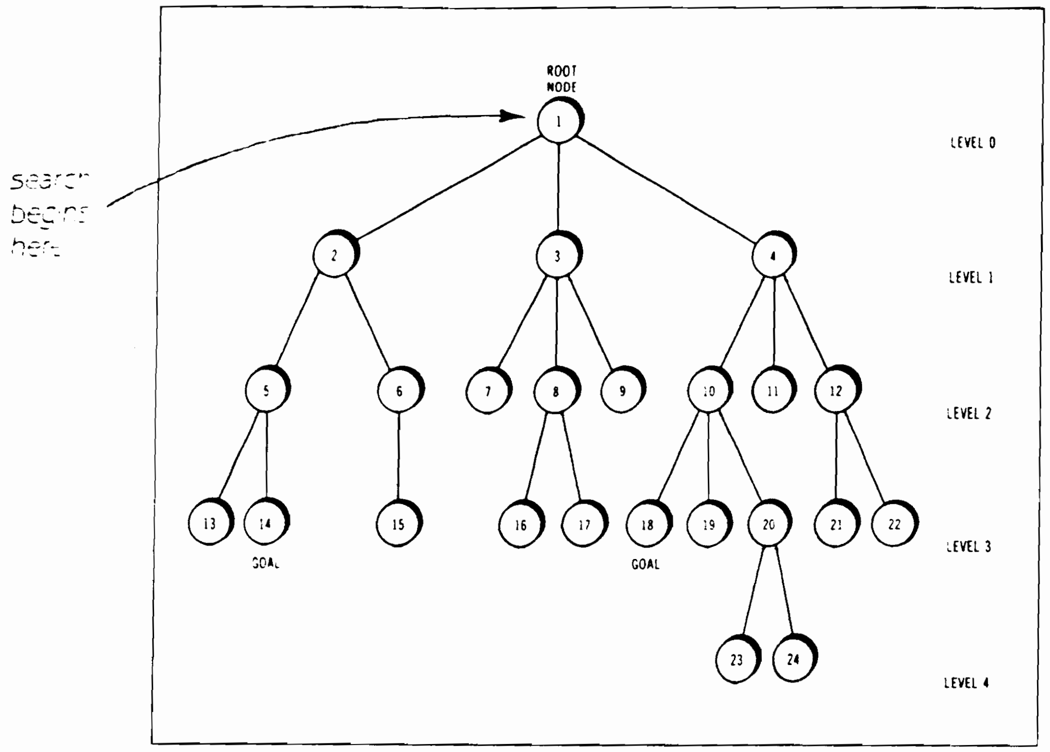


Figure 3.7 - A Search Tree

Adopted from [Frenzel, 87]

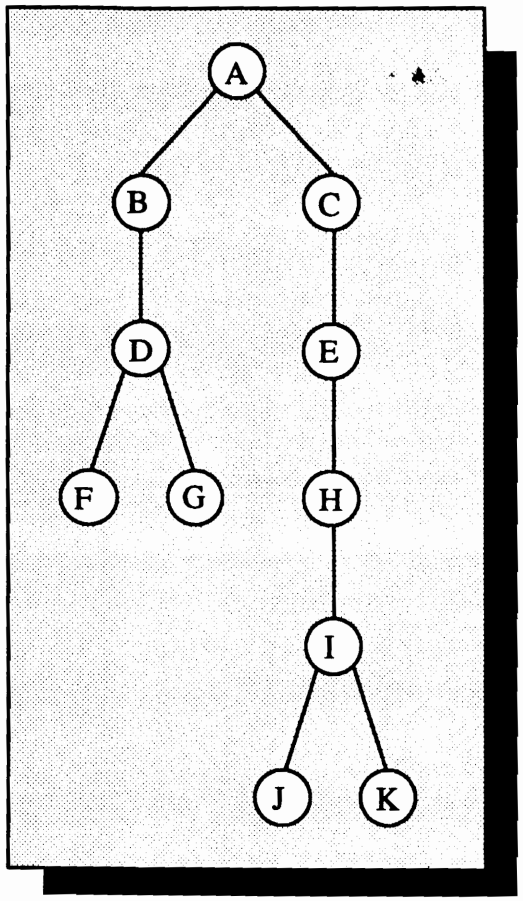
Search Methods

Knowledge-based system use the inference engine to decide how and in what sequence the knowledge base is searched. The inference engine controls and executes the reasoning strategies used by the system. The search techniques are contained within the inference engine of the selected language or tool.

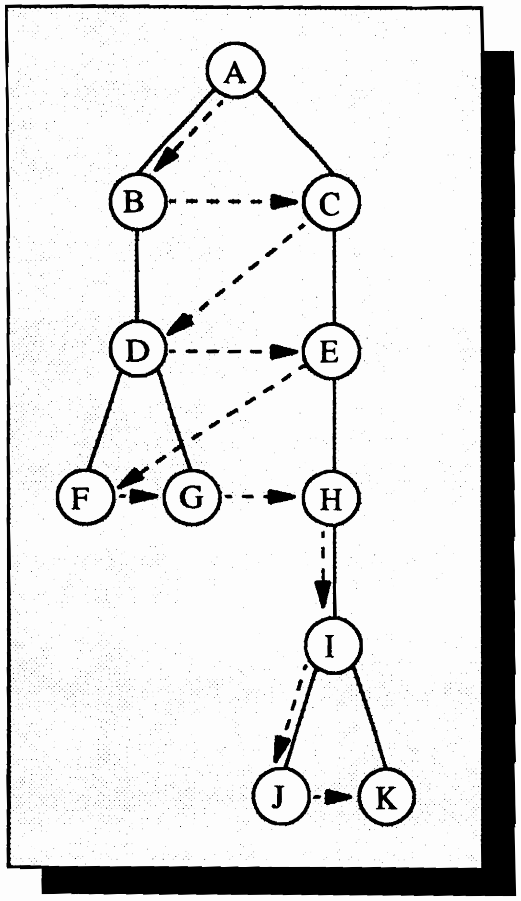
There are two basic methods for searching state space representation of the knowledge base. Theses methods are *blind search* and *heuristic search.* In either case, the system's objective is to find a solution, or at least a node that seems closer to a solution. Blind search can be used in those cases in which the system is unable to determine how close it is to a solution, and therefore must search the state space according to some pattern that potentially passes to through every node, and therefore will certainly bring it to a solution eventually. If there is a way of evaluating the "quality" of the various exits from node, heuristic methods can be used.

Two examples of blind search are breadth-first and depth-first. Figure 3.8 illustrates these two techniques on the search tree.

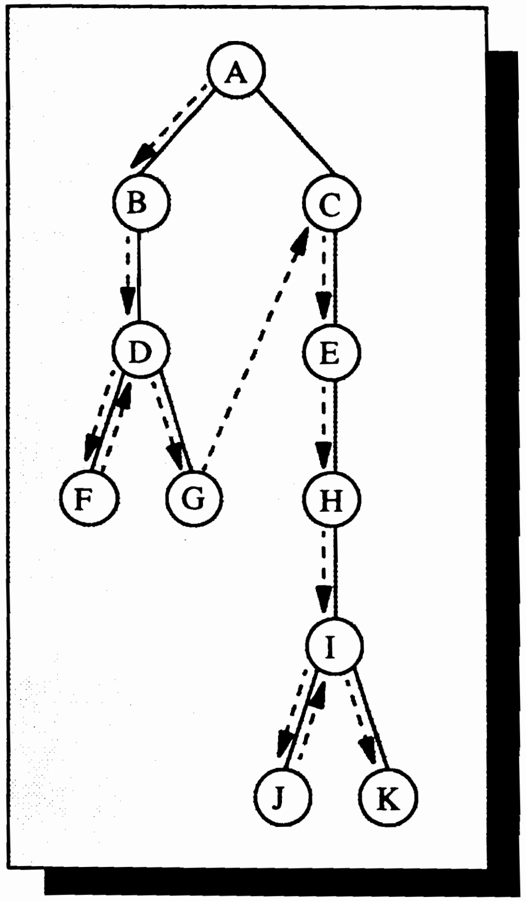
Breadth-First. A breadth-first search begins at the root node and continues across the nodes at each level before moving to the next lower level. It stops when it finds a solution or gives up. A breadth-first search will find the shortest path between the given initial state and the goal state.



a) Search Tree.



b) Breadth-First Search Strategy.



c) Depth-First Search Strategy.

Figure 3.8 - Search Strategies: Breadth-First and Depth-First Search.

Figure 3.8(b) illustrates a breadth-first search. The breadth-first search explores all of the steps at a given level before moving to the subsequent level. A breadth-first search explores a problem in a general manner.

Depth-First. A depth-first search begins at the root node and continues down through successive levels of the left most branch. When it gets to a leaf, it backs up to the last previous node where there is an unexplored path, and takes it. This search process continues until a solution is reached or the search reaches a dead end and is forced to backtrack.A depth-first search is illustrated in Figure 3.8(c).

Heuristic Search

In this type of search, heuristics are used to focus the search on those portions of the search tree most likely to yield a solution. Heuristic techniques can eliminate large portions of the search tree, thus greatly accelerating problem solution.

There are general-purpose heuristics and domain-specific heuristics. An example of a general-purpose heuristic is ***depth-bound*** search. This technique is used in depth-first searches to help eliminate the possibility that the search will go off into some deep network of branches where there may be no possibility for solution. A depth-bound search arbitrarily limits the depth of the search to some maximum level.

Domain-specific heuristics are applicable only to certain types of problems. One type of domain-specific heuristic is the use of special rules, called "meta rules." that state ways that the knowledge rules can be used. If a meta rule is true given certain input facts, it will limit the search to a subset of the knowledge base containing rules that will most likely lead to the solution.

Search Strategies for Control

In a rule-based system, another kind of search must be carried out to choose or control the next rule to be fired. Different types of control strategies are available. These include: forward chaining and backward chaining. Forward chaining is effective for planning and design-based applications. Backward chaining is effective for diagnostic and control-based applications.

Forward Chaining

Forward chaining is also called forward reasoning. Forward chaining is a way to emulate human deductive or data-driven reasoning.

Using a forward chaining control sequence, the inference engine will start with any available facts or frames in the knowledge base and search for those facts and frames in the IF portions of the rules. If the IF portion of the rule matches, the rule is fired. The THEN portion of the rule is said to be true and new information is inferred and stored in the context. With this new information, the inference engine moves forward to find a match for a new rule. This process continues until no further conclusions can be reached.

Rules that may be available to the system, but do not apply, are eliminated by the system. This method of reasoning is known as fact-driven or data-directed reasoning.

Forward chaining is usually associated with knowledge bases that have large numbers of possible solutions. Further, these systems are frequently used when data is the starting point for solving a problem. Examples of applications that are associated with forward chaining are planning, designing, and forecasting.

Backward Chaining.

Backward search or backward chaining is also called backward reasoning. The backward chaining is a way to emulate human inductive reasoning or goal-directed reasoning. Backward chaining starts with the goal node and works backward towards initial states. The strategy for backward search is for the user to assume a particular event outcome and search for evidence that supports the assumption.

In a backward chaining procedure, the inference engine looks at the THEN part of the rule first and then attempts to prove the IF portion. It looks in its knowledge base for rules that conclude that portion of the IF statement.

Backward chaining will provide the fastest search if there are more possible outcomes than initial states. Examples of applications that are associated with backward chaining are diagnostic problems.

3.5 Chapter Summary

This chapter has presented an overview of the general concepts of knowledge-based systems (KBSs).

The general components of KBSs architecture were discussed. Most KBSs comprise the following key parts: knowledge base, inference engine, context, explanation subsystem, and knowledge input subsystem.

Knowledge represented in the knowledge base consists of declarative and procedural knowledge. Knowledge representation techniques include: semantic nets, frames, object-oriented programming and production rules.

The chapter also discussed some of the search methods and techniques for search the state space of the knowledge base. Blind and heuristic search were presented. Examples of blind search include breadth-first and depth-fist. Control strategies implemented by the inference engine include forward- and backward-chaining.

4.0 CONSTRUCTION CONSTRAINTS in PROJECT PLANNING and SCHEDULING

A construction project is a collection of individual operations or activities. Establishing the logical relationships among these operations and determining their start and finish dates constitutes the construction plan and schedule. Network diagrams seek to represent the logical relationships between these various operations that are required to be performed. Such relationships are defined by different constraints that control the sequence among these operations and, hence, govern the start and finish dates of the activity schedule.

This chapter presents an overview of the different constraints that define the logical sequence among the different construction activities. The chapter first discusses basic concepts of hard and soft logic in activity scheduling. The chapter also examines the different constraints that control the relationships among the different activities and define their logical sequence. If the reader is familiar with the concepts and issues associated with these different constraints, he or she may wish to skip this chapter without loosing any continuity.

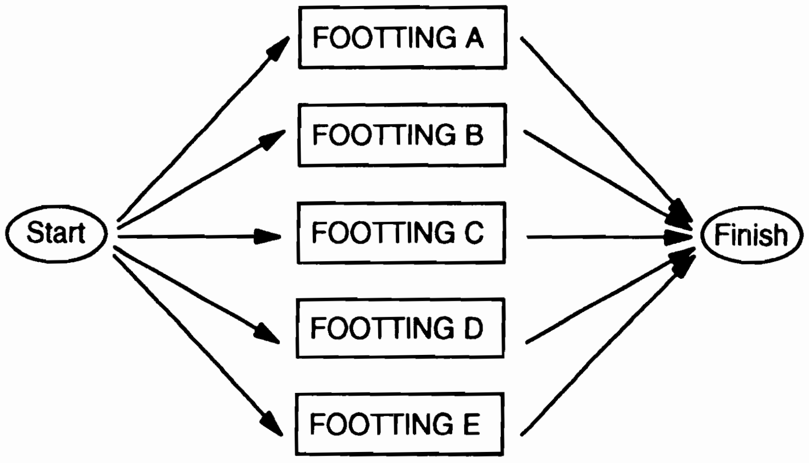
4.1 Hard Logic and Soft Logic in Activity Relationships

Logical relationships amongst activities in a network may be categorized into *highly dependent* or *largely independent* [Chang et al., 89]. The performances of certain construction activities are highly dependent due to their physical and technological characteristics. Therefore, their logical relationship in a network is considered fixed or "hard". This is because these activities can only be arranged in a single logical sequence determined by their technological dependency. That the construction of first floor columns can only proceed after the completion of the ground footings is a typical example of this type of activities.

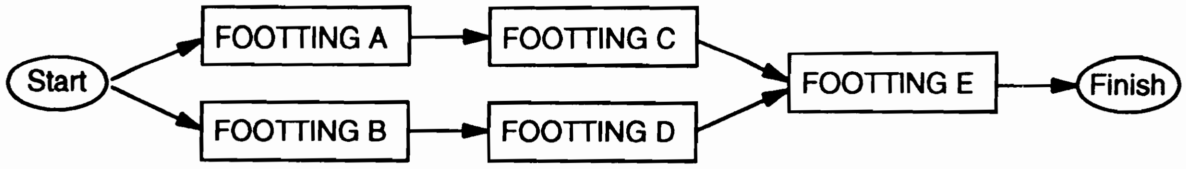
Some activities, however, are largely independent; they are much more parallel or interchangeable in nature. As opposed to highly dependent activities, activities belonging to this category can be arranged in a variety of logical sequences in the network. For example, the erection sequence for the ground footings is largely independent. They can be poured in parallel or sequentially. In case they are sequentially installed, the order probably will be interchangeable. In theory, activities of this kind have no technological constraints and can be simultaneously performed. In reality, they may be subject to some degree of external constraints such as insufficient resources, cost considerations, etc. When such conditions are changed, the constraints are changed and thus the relationships are altered. Therefore, this type of relationships amongst activities in a network is considered as variable or "soft".

It should be noted, however, that precedence relationships among independent activities based on external constraints should not be introduced in the initial activity network. Rather, independent activities should be initially treated as possibly being parallel. This is because treating such constraints as fixed logic relationships deprive independent activities of their parallel or interchangeable characteristics. When these imposed constraints change or disappear, the fixed logic relations can not automatically reflect the new situation.

This is illustrated in the example of Figure 4.1. In Figure 4.1(a), the construction of the ground footings is independent from one another and are initially assumed to be performed simultaneously. That is, no logical dependency is assigned among the individual footings. On the other hand, Figure 4.1(b) has introduced hard logic dependencies among the highly independent footing activities as a result of limited resource considerations. If such an external condition is to disappear or change once construction starts, it will be impossible for the scheduling algorithm to reflect such changes on the schedule unless such logic is rearranged in the network.



a) Footing activities scheduled in parallel



b) Footing activities scheduled in series (Hard Logic introduced)

Figure 4.1 - Hard and Soft Logic in Activity Networks

4.2 Construction Constraints

Many constraints determine the hard and soft logical relationships among the different operations or activities. Some operations get constrained either because of limited availability of resources or work space; others are constrained due to weather and climatic conditions. The relationship among some operations is defined by technological constraints; others may be dependent on receipt of some materials having a certain delivery date. Unless these constraints are suitably provided for in the network plan (hard logic) or considered by the scheduling algorithm during the scheduling procedure (soft logic), the generated schedule will not be realistic and achievable.

As illustrated in Figure 4.2, construction constraints can be classified into two main categories or groups:

1) project-related constraints; and

2) schedule-related constraints.

Project-related constraints are external factors directly related to the project or facility being built. Such constraints are unique to the project and the geographical location to where the project is constructed. Examples include: *imposed dates, specific technological requirements, safety and code requirements, resource availability constraints, work space availability constraints, weather conditions, damage constraints, etc.*

On the other hand, schedule-related constraints are not unique to any one project but

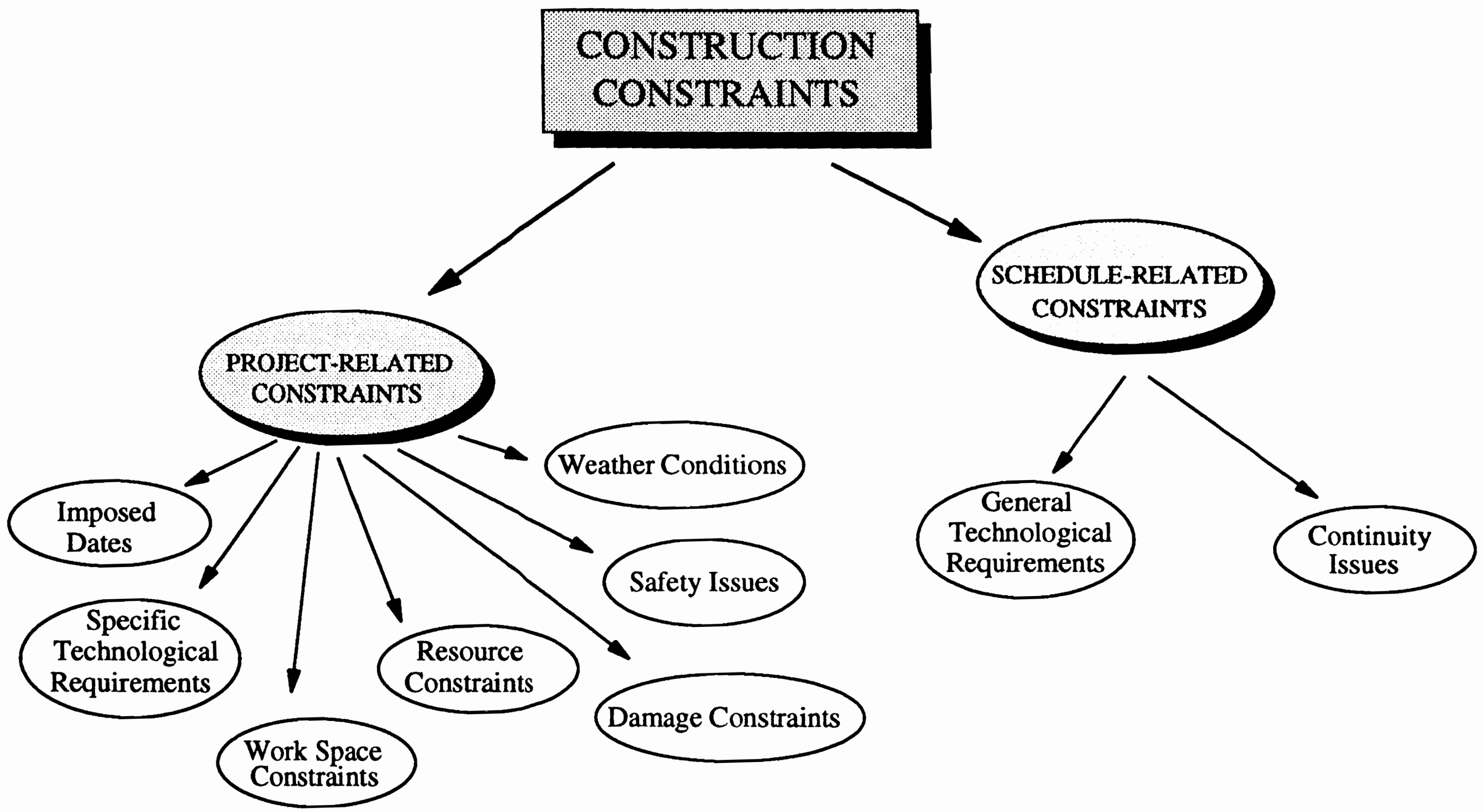


Figure 4.2 - Categories of Construction Constraints

rather general and applies to different projects. Examples include: ***general technological requirements, work continuity issues related to scheduling tall buildings.*** The following subsections elaborate in more detail on these constraints.

4.2.1 Project-related Constraints

Project-related constraints are external criteria that are dependent on the nature and scope of the project under consideration. These criteria include:

1- Imposed dates:

These are start/finish milestone dates that must be met by the corresponding scheduled dates of certain activities during the generation of the schedule. For example, an imposed date to finish a section of a highway or a building should be reflected in the schedule by scheduling the late finish date of the last activity in that section before or at that imposed date.

Imposed dates may be due to many reasons, such as: contractual obligation dates (start date, completion date, partial completion date, etc.), delivery date of material or equipment, shopdrawings dates, material samples dates, etc.

Contractual obligation dates. These are dates usually set in the contract and accepted and approved by all parties. Provisions should be provided in the schedule to respect such dates to satisfy the conditions of the contract and avoid any penalties or liquidated damages.

Delivery dates. Most of the operations in a project require materials for construction, many need special types of equipment. These may have to be ordered or procured from different sources. The activities which depend on the receipt of these materials/equipments can start only upon delivery to site. So, it is necessary to provide in the schedule constraints methods to control the start of these affected operations.

Shopdrawings and material samples milestone dates may be imposed on the schedule to reflect submittal and approval dates of these items.

2- Specific technological constraints:

These are requirements imposed on activity relationships to satisfy physical rules and limitations. As opposed to general technological constraints (discussed in section 4.2.1), these constraints are specific to the scope of the project.

An example of such constraints is the construction of a set of beams before others as a result of a specific technological requirement related to the project under consideration.

In the construction of the skeleton, the construction of any floor can not start before the completion of the floor below is considered a general technological requirement. This constraint associated with the method of construction is general in nature and is applicable to any project that involves construction of floors.

3-. Safety and code requirements

Safety issues associated with construction work are mainly due to hazardous or unsafe physical operations or as a result of environmental effects resulting from the execution of the activity.

Physical issues are usually a result of the nature of the work being executed and are associated directly with the construction operation itself.

Environmental issues, on the other hand, are those effects that accompany the execution of the activity. Almost any construction activity has environmental effects as a by-product of its progress. If these environmental effects are such that the work area is unsafe, the development of the effect-causing activity precludes the concurrent progress of any other activity within the affected area.

Safety and related code requirements constraints are imposed on the construction schedule to insure that the facility is constructed with no or minimum accidents or losses. Examples of safety-based activity relationships include:

1- In construction of steel buildings, some safety code regulations require a horizontal diaphragm following not more than two stories behind the steel erection crew in steel framed structures.

2- Permanent (or temporary) railings activities in balconies and stairways should be completed before other activities can start in or around such areas.

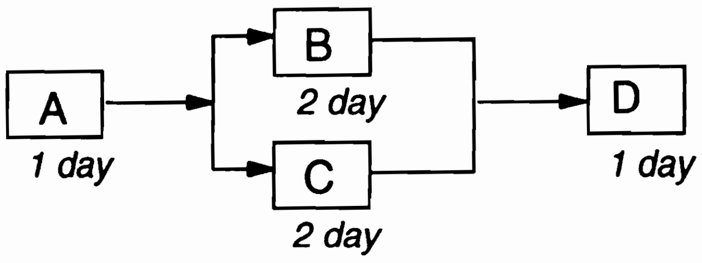
3- During the fireproofing of steel frames with insulating materials, no other activity can be performed concurrently within the affected area.

4- Resource constraints

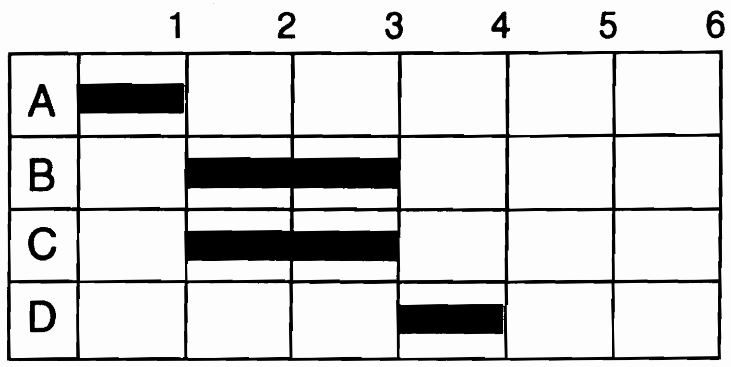
The execution of a project depends amongst other things, on the availability of resources required for each activity. Construction resources may be classified under manpower, equipment and material.

In order to consider resources as a constraint in project scheduling, the conflict between the resources that can be made available to the project and those apparently needed as determined from the network model should be recognized.

The limited availability of resources to execute a project is always a significant factor in determining the sequence of activities in the final schedule. This is illustrated in Figure 4.3. In Figure 4.3(a), activities (B) and (C) are technically independent and can be performed concurrently as shown in Figure 4.3(b). This is done under the assumption that the resources required to perform activities (B) and (C) are unlimited for the duration of the activities. If both activities require a resource *crane* for there execution, and only one crane is available, then activities (B) and (C) must be re-



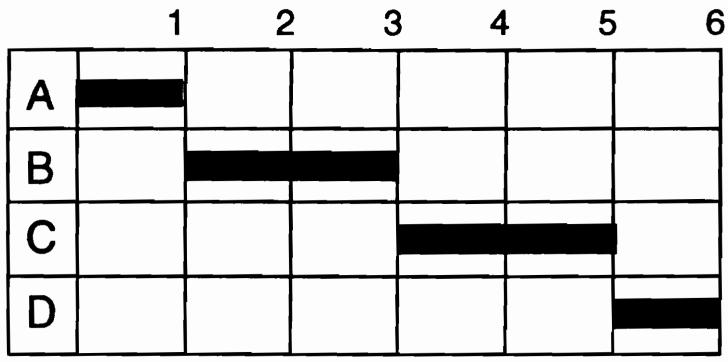
a) Logical network



Activities (B) and (C) are logically independent.

Both can be scheduled concurrently. Unlimited resource availability is assumed.

b) Schedule based on logical requirements only



Resource constraint condition: Activity (B) and (C) require the use of a crane. Only one crane is avialable.

Activities (B) and (C) must be rescheduled sequentially with priority given to one over the other. In this example, priority is given to (B) over (C).

c) Schedule based on logical requirements and resource constraints

Figure 4.3 - Effects of Resource Constraints on Network Initial Schedule.

scheduled sequentially as shown in Figure 4.3(c).

It should be noted that it is appropriate not to take resource constraints into account at the initial stage when the first network is being prepared as was shown in Figure 4.3(a). At this stage, the network should be developed based mostly on the logical sequencing of operations without considering resource constraints. Once an initial schedule is prepared, it then can be tested in relation to the availability of resources and readjusted as considered necessary as was done in Figure 4.3(c).

Scheduling under resource considerations will be dealt with in more detail in chapter 6 of this document.

5- <u>Work Space availability constraints</u>

Any construction activity requires a specific work space, termed *work space demand*, for the execution of the activity. This demand is based on the requirements of the activity's allocated resources for work area or space. When such required demand becomes unavailable, the activity can not be executed or, in some cases, performed with a lower productivity rate and/or more hazardous conditions.

Work space can be viewed as a special type of resource in the sense that the availability of the space is a necessary requirement to perform the work. When two or more activities are concurrently scheduled in the same work zone, the use of limited work space by one activity (with a higher priority), means that another activity requiring it becomes affected. One way to solve this problem is to schedule the activities sequentially rather than concurrently. This scheduling adjustment of the sequence as a result of its limited availability gives work space characteristics of a scarce resource.

In construction, limited work space or congestion is a result of many factors that are mainly due to crew interference and overcrowding and construction material use and storage. Crew interference is mainly confined to multiple craft interferences as a result of poor scheduling of the different trades. This is exemplified in delays that occur as one trade crew waits while another, with a higher priority, occupies a needed work space. Overcrowding, on the other hand, is a result of the physical features of the construction site and/or the high densities of workers scheduled to work in the same confined area.

Use and storage of construction materials in work areas with limited space availability may also results in work space congestion. In high rise construction, for example, storage areas around the site are limited because of the location of such type of construction in crowded downtown locations. As a result, construction materials are usually stored in the floors resulting in limited work space availability. Also, some activities such as HVAC duct installation requires large work space areas to initially install the material before putting it in place. This may deprive other concurrent activities from the required space to execute the work.

6- Weather conditions

It may not be possible to undertake certain types of jobs in the field, during certain periods, due to severe climatic conditions. This is because some construction components are of such nature that they require a weather protected environment for their installation. This implies that the protected component has to be installed only after the protecting component is in place. In addition, construction crews can not operate under certain weather conditions (e.g. very high winds, extreme temperatures, heavy rain).

For example, it is difficult to carry out excavation works during monsoons in tropical countries. Likewise, in very cold countries it becomes difficult to carry out certain types of jobs in winters due to heavy snow, cold, or strong winds.

If the project schedule is to be realistic, the delays due to these known weather or climatic conditions must be provided for by introducing constraints to the execution of the relevant activities which have to wait until after the weather conditions improve.

Examples of weather-based activity relationships include:

1- Waterproofing of certain number of upper floors should proceed the start of finishing works at a lower floor.

2- At any given floor, window installations should proceed the start of finishing activities that are weather sensitive (e.g. dray wall, false ceiling, door installation, etc.).

7- <u>Damage constraints</u>

Some construction activities in progress may cause damage to other activities that are in the same work area or zone. This damage may be a result of physical impact, debris, dust, scratches, etc. In addition, movement of workers and materials may damage other work that is in the movement path of such resources.

If a certain activity damages finished work of another activity, then the damageable work should be performed afterwards. Examples of damage-based activity relationships include:

1- Flooring finishing works should succeed ceiling and walling works.

2- Carpeting should not start before painting works in the area is completed.

3- Cladding finishing works should proceed in a downward direction to avoid damage of installed material from falling debris.

It should be noted, however, that such constraints could in some cases be disregarded if proper precautions are provided to protect completed work from damages (e.g. covering of finished installations with plastic sheets).

4.2.2 Schedule-related constraints

Schedule-related constraints are internal criteria that are independent of the project and are characteristics directly associated with the scheduling procedure. These criteria include:

1- <u>General technological constraints</u>

These are requirements imposed on activity relationships to satisfy physical rules that are general in nature and are associated with the method of construction in general.

Examples for general technology-based activity relationships include:

i) Excavation works proceed foundation and infrastructure concrete and civil works.

ii) In construction of dry wall, taping and painting activities are preceded by framing and mounting of boards.

iii) In electrical works, wiring succeeds first fix electrical conduits mounting.

2- <u>Continuity issues</u>

An important factor in scheduling is the classification of activities into two groups, continuous or intermittent. When a *continuous activity* is started, it must be worked without interruption until finished; an *intermittent activity* can proceed piecemeal in isolated sections at irregular periods of time. Many activities can be divided without harmful effects (e.g. cleaning). In other cases it would be positively detrimental if the activity were interrupted (e.g. pouring concrete).

4.3 Chapter Summary

Logical relationships amongst activities may be classified as "hard" or "soft". Activities of the hard logic category are highly independent because of their physical and technological dependency characteristics. Such activities are introduced in the network with fixed logical relationships. On the other hand, activities belonging to the soft logic category are largely independent. Such activities can be executed interchangeably in nature. As a result, they can be presented in the network in a variety of logical sequences.

Different construction constraints determines the logical relationships among the different activities. Such constraints can be classified into two main categories or groups: 1) project-related constraints; and 2) schedule-related constraints. Project-related constraints are directly related to the project or facility being built. Such constraints are unique to the project and the geographical location to where the project is constructed. On the other hand, schedule-related constraints are not unique to any one project but rather general and applies to different projects.

5.0 ISSUES in SCHEDULING MULTI-STORY BUILDINGS

A certain category of construction projects, termed *linear or repetitive projects*, involves considerable repetitive units performed consecutively at all stages of the project by the same crew. The construction of these units may be analogous to the continuous manufacture of many identical units, each of which requires considerable time for completion.

In such type of linear projects, the sequence of activities is not discrete. Rather, the activities progress continuously in sequence along the length of the project. Each of these activities can be scheduled to commence in sequence at one end of the project and progress towards the other end.

Multi-story buildings construction falls under this category of linear projects. As shown in Figure 5.1, a typical multi-story structure can be separated into two distinct modes of construction [O'Brien et al., 75]. Each mode or segment has its specific characteristics. These characteristics influence the planning and scheduling of the construction activities associated with these modes.

The **first mode** constitutes a small part of the structure and is characterized by a wide variety of activities that represent a typical construction project with a well defined start and completion. These activities are unique and do not inhibit any repetitive nature. Activities in this mode include:

- Basic preparation works (e.g. site planning and layout, erection of temporary facilities, etc.),
- substructure (e.g foundations, underground services, etc.),
- superstructure to the first typical floor, and
- roof works. This include all civil, finishing services works, and elevator machine rooms in the roof area. Depending on the number of floors, a multi-story building may have more than one elevator machine room on different floors.

Scheduling methods and strategies for this type of construction are very basic. Current traditional networking techniques are used effectively to schedule this work.

The **second mode** of construction is the linear or repetitive segment of the structure and constitutes the majority of the construction work. This mode of construction starts when

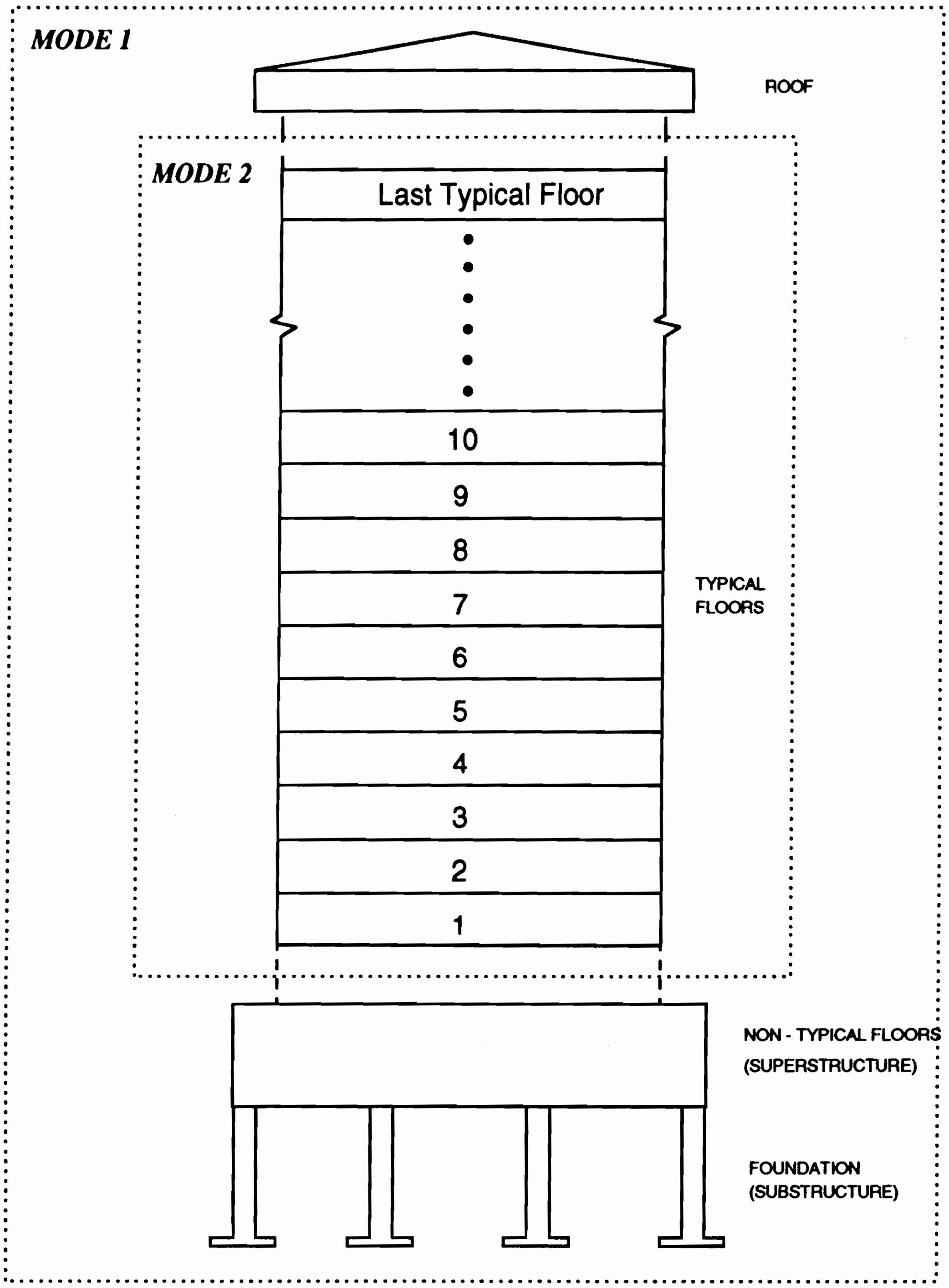

MODE 1
ROOF
MODE 2
Last Typical Floor
10
9
8
7
6
5
4
3
2
1
TYPICAL FLOORS
NON - TYPICAL FLOORS (SUPERSTRUCTURE)
FOUNDATION (SUBSTRUCTURE)

Figure 5.1 - Construction Modes in Multi-Story Buildings

the point of progress reaches the first typical floor. Each floor now becomes a repetitive construction unit with a well-defined start and completion. Similarly, each unit comprises various construction activities that are almost identically repeated as the construction advances from one floor to the next. Because of the repetitive characteristics of this mode, the planning and scheduling procedure should consider certain issucs in order to produce a feasible and realistic schedule.

•

The construction work in this mode involves considerable repetitive activities performed consecutively by the same crew. As a result, it is advantages to insure the continuity of flow of these crews while performing the work along all the typical floors. Maintaining continuity of flow by minimizing interruptions of work will decrease idle waiting intervals of manpower and equipment, thereby saving time and reducing costs considerably. Hence, the planning and scheduling procedure should try to maintain the continuity of the work especially of the major trades, during the scheduling of the different activities along all the typical floors.

•

Another important issue that should be considered while scheduling this repetitive segment of the high rise facility is the impact of decisions made to select the appropriate production rate which will actually control the schedule. As each construction crew moves from one floor to the next, the execution time becomes controlled by the time required by each activity to work upward or downward through the typical floors of the building. This is in turn affected by the production rate established for each activity in relation to its preceding activity. Slow activities with low production rate will control the pace of the work over fast activities with higher production rates. Making decisions to select the appropriate rate will affect the generated schedule. As a result, the scheduling procedure should consider the determination of each activity's production rate as a main decision factor in generating the schedule.

A third issue related to the repetitive nature of the typical floors is that it provides a great potential for performance improvement as a result of the learning curve effect. As each activity is continuously repeated from one typical floor to another, productivity of the crews is gradually improved. This is because crews of each trade becomes more familiar with the work as construction progresses from one floor to the next. This improvement due to repetition decreases construction time and reduces costs. Learning by repetition is maximized by insuring continuity in the schedule.

The following sections discuss in more detail these main issues and characteristics associated with planning and scheduling multi-story buildings.If the reader is familiar with these issues, he or she may wish to skip this chapter without loosing any continuity.

5.1 Modelling Repetitive Work

There has been considerable research in the field of planning and scheduling construction projects with repetitive nature. This section will explain in general the current methods developed for planning and scheduling this type of work. The section will also represent a limited review of the literature available on these methods.

<u>**Traditional Methods**</u> Traditional scheduling methods can be classified into two major categories: bar charts and network based techniques.

The barchart is one of the simplest scheduling methods and is an excellent representation of ow activities are spread over time. However, a major disadvantage of a barchart is that it can not clearly show complex inter-relationships between the different activities. The barchart, however, have proved to be an excellent representation tool for the results of the schedule generation process because of its simplicity and clarity.

Network based techniques, such as **PERT** and **CPM**, have overcome that disadvantage in the barchart by showing logical relationships among the different construction activities. These techniques are based on defining project activities and linking them in a network representing the logical construction sequence. Each activity is then assigned a duration or a set of durations and the network calculation is conducted in order to establish activity start and finish dates.

Earlier attempts used these different traditional scheduling techniques to model construction work with repetitive nature. [O'Brien et al., 85] reported that in the early 1960s' the CPM network technique was used in planning large repetitive projects such as the Philadelphia School District project to build 200 schools. In 1981, CPM was also used in planning the King Khaled Military City, Saudi Arabia. However, such techniques somehow have fallen short towards adequate planning and scheduling linear projects composed of activities with repetitive nature.

Network techniques do not guarantee maintaining the continuity of the work. A crew with a fast production rate might be idle while waiting for preceding crews with slower production rates to finish their work. This is because such techniques schedule the start of each activity as soon as all its preceding activities are finished.

Another major disadvantage of network techniques to model repetitive construction is the difficulty of production balancing to minimize project duration and costs. An activity production rate is dependent on the number of crews assigned to the activity. Because these techniques use activity duration as basic input data, they necessitate the determination of the quantities of resources in advance, with no regard of their effect on the preceding activities. This tend to mask the realities of production decisions which will actually control the schedule. To alter any production rates of groups of repetitive activities, the scheduler must modify the duration of each individual affected activity at all floor to reflect the new duration and working hours, together with any change in resource requirements. This leads to a succession of endless time consuming and repetitive calculation.

as the Philadelphia School District project to build 200 schools. In 1981, CPM was also used in planning the King Khaled Military City, Saudi Arabia. However, such techniques somehow have fallen short towards adequate planning and scheduling linear projects composed of activities with repetitive nature.

Network techniques do not guarantee maintaining the continuity of the work. A crew with a fast production rate might be idle while waiting for preceding crews with slower production rates to finish their work. This is because such techniques schedule the start of each activity as soon as all its preceding activities are finished.

Another major disadvantage of network techniques to model repetitive construction is the difficulty of production balancing to minimize project duration and costs. An activity production rate is dependent on the number of crews assigned to the activity. Because these techniques use activity duration as basic input data, they necessitate the determination of the quantities of resources in advance, with no regard of their effect on the preceding activities. This tend to mask the realities of production decisions which will actually control the schedule. To alter any production rates of groups of repetitive activities, the scheduler must modify the duration of each individual affected activity at all floor to reflect the new duration and working hours, together with any change in resource requirements. This leads to a succession of endless time consuming and repetitive calculation.

A third disadvantage of network techniques for modelling linear projects result from the large number of repeated identical activities required to represent the project. Using these techniques, each activity within a project needs to be defined and linked to others, individually, so that the order of execution may be derived. This large number of activities forces the schedule to be cluttered with repetition of information which makes the project extremely difficult to visualize and makes the resulting schedule cumbersome and confusing.

Combined Approaches Recognition of these disadvantages of traditional network techniques in scheduling repetitive projects led researchers to the use of several other different techniques. The techniques that were developed are generally referred to as '*Linear Scheduling Methods*'. Their origin is not clear; there may actually have been multiple origins. These techniques have been originally devised to solve industrial production problems and their use in the construction industry is rather a recent event. A detailed survey of the available literature on the various techniques was presented by [Johnston, 81].

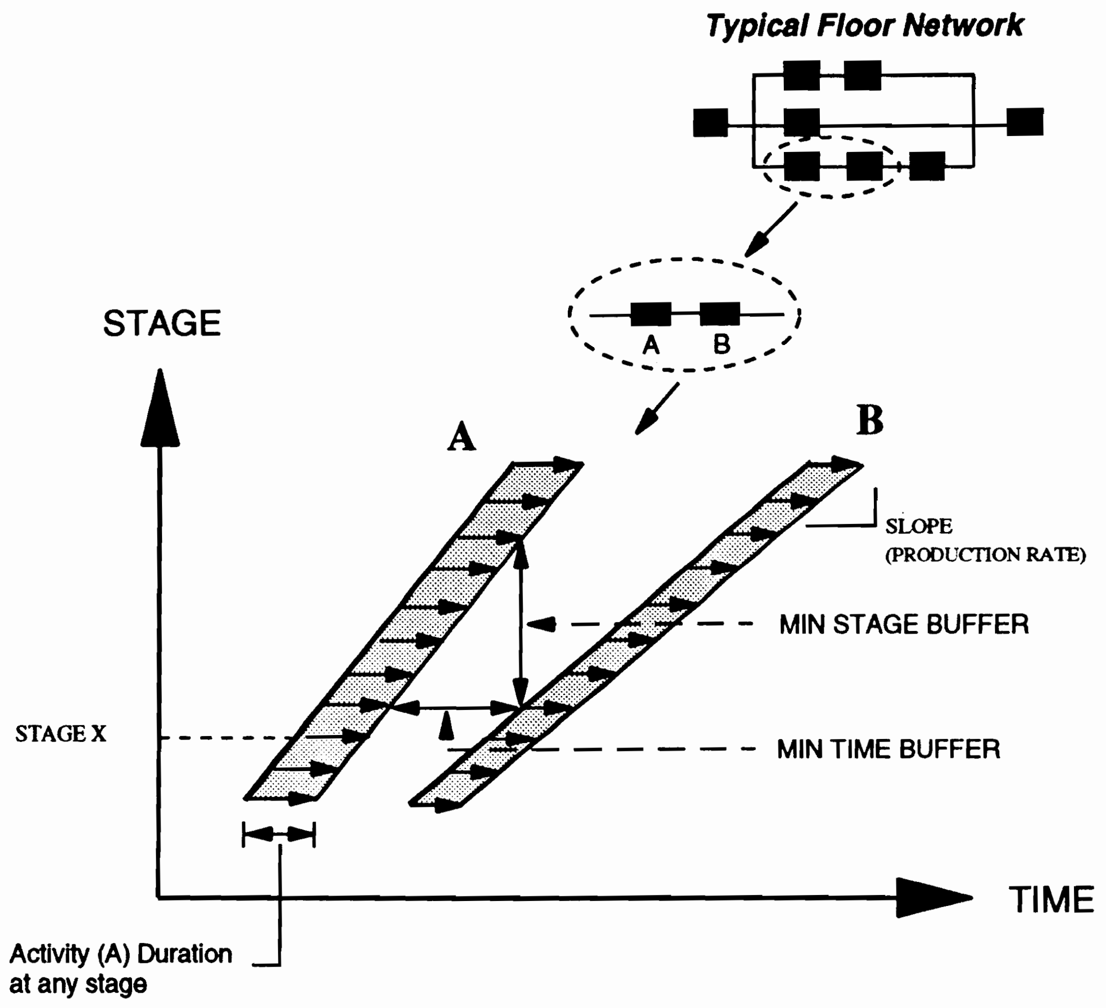
The techniques included a multitude of variations and incorporated combinations of a network technique, a graphical technique and an analytical technique. These combined approaches were named differently. Examples included: the **Line of Balance** [Arditi 1986, Johnston 1981, Carr and Meyer 1974, Khisty 1970]; the **Vertical Production Method (VPM)** [O'Brien, 1975]; the **Time Space Scheduling** [Stradal and Cacha 1982]; **Velocity**

Diagrams [Roech 1972]; the **Linear Scheduling Method (LSM)** [Chrazanwski 1986]; the **Dynamic Programming Approach** [Russell and Caselton 1988]; and the **Repetitive Project Model (RPM)** [Reda, 1990]; **Time Chainage Charts** [Mawdesley et al., 1990].

In these combined approaches, the network techniques are utilized to represent the activities and their dependencies needed to complete one typical stage or unit (e.g. typical floor) of a repetitive project. The analytical techniques define the mathematical formulations that set the relation and linkage among the activities along the repetitive stages. These relations and linkages define the schedule that is presented through the graphical techniques.

The graphical techniques are used to represent the repetitive activities along all the typical stages or units (e.g. typical floors). These techniques use a two axis diagram; time versus location, where location is a measure of progress. As shown in Figure 5.2, each activity is represented by one flow line curve that is projected on the diagram at the activity's rate of progress (i.e production rate) and represents the movement of the activity crew from the first repetitive stage to the last.

Figure 5.2 illustrates flow line curves for two activities **A** and **B**. The slope of each line represents the production rate for each activity. The intersections of a horizontal line drawn at a particular stage (stage x) with the flow lines give the starting times of activities at that stage. The intersection of a vertical line drawn at any particular time with



TIME BUFFER

By definition, a time buffer is an allowance included between activities on the same stage to cater for random differences in productivity, for recieving and dispatching components, moving of crews, recreational breaks, etc.

STAGE BUFFER

This is an allowance introduced between construction activities on different stages. For example, blockwork should be two floors lower than the floor being poured for safety reasons and code regulations.

Figure 5.2 - Flow Line Curves for Repetitive Activities

the flow lines give the activities concurrently working at the different stages. Using a flow line for each activity of the typical floor network, a project flow line schedule can be developed showing flow lines for all the activities at all the stages, as shown in Figure 5.3 [O'Brien et al., 85].

BUFFERS

When construction activities progress continuously in a chain, some spacing between activities is required. This spacing serves as a buffer and may be a required distance or time interval between activities. In Figure 5.2, the horizontal distance between the two lines at any stage represents the time buffer between the two activities at that stage, while the vertical distance between the two lines represents the stage (or distance) buffer between the two activities at a particular time.

Time Buffers During construction, the flow of work is subject to many interferences such as: random differences in productivity, moving of crews, receiving and dispatching of material and equipment, recreational breaks, bad weather conditions, etc. A ***time buffer*** is an allowance included between the activities on the same stage to cater for such interference.

Stage Buffers is a time restriction included between concurrent activities on different stages to allow for certain conditions during construction. Stage or vertical buffers could be viewed as a vertical logical constraint among activities scheduled at

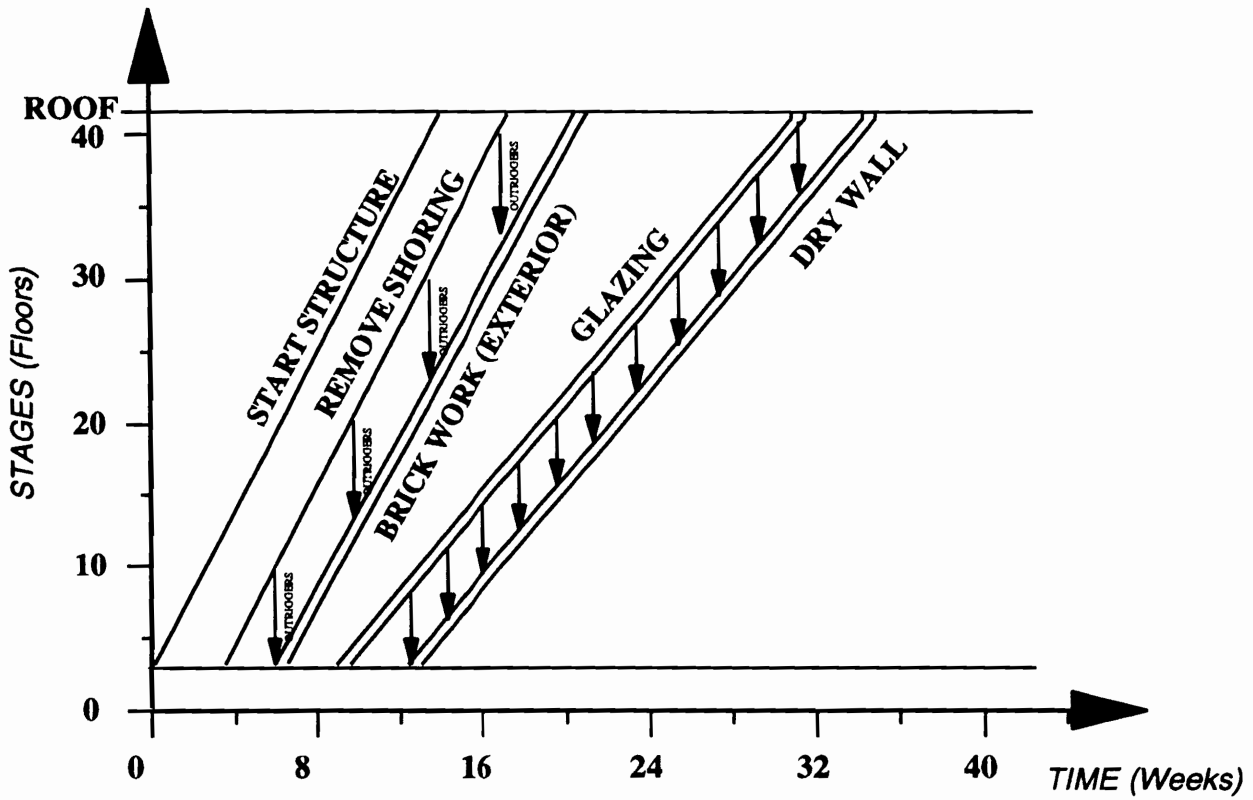


Figure 5.3 - Flow Line Curves for a High-Rise Project

[O'Brien et al., 1985]

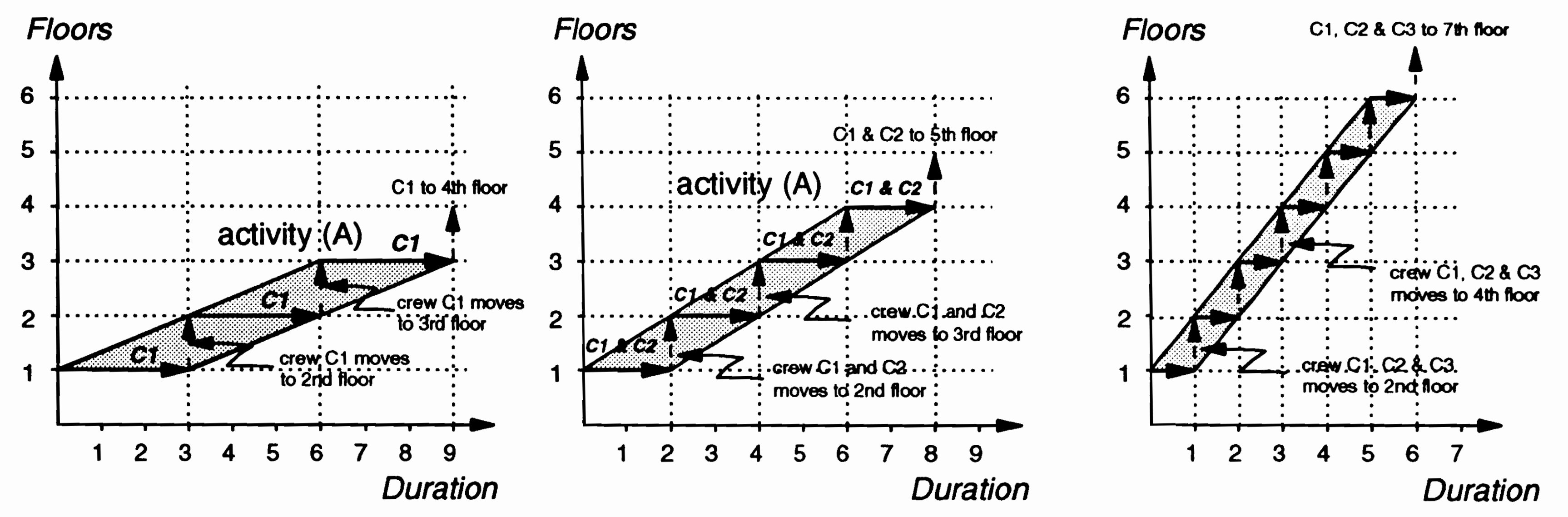
different stages or floors. For example, a stage buffer of a maximum of two floors is usually required between deck installation and the erection of the steel frame for reasons of safety and code regulations. A stage or vertical buffer is discussed further in section 5.4.

The advantages of using the graphical approach in these combined techniques are simplicity and the ease to visualize the whole project. Using such combined techniques with a mathematical formulation model (analytical techniques) as in the Repetitive Project Model (RPM) [Reda, 90], crew production rates could be used as decision variables and are adjusted to minimize project time and costs.

5.2 Production Rates

An activity production rate is dependent on the number of crews or resources assigned to the activity. Almost any activity can be performed with a range of quantities of resources. These quantities determine the rate of production and thus the duration of the activity which may affect that of the project.

Figure 5.4 graphically illustrates the effect of increasing the number of crews working in the floor on the production rate of the activity. Using only one crew, activity (A) requires 3 days for each floor and is said to have a production rate of 0.34 floor/day as illustrated by the flow line in Figure 5.4(a). Increasing the number of crews to 2 and 3 increases the



a) One crew production rate

b) Two crews production rate

c) Three crews production rate

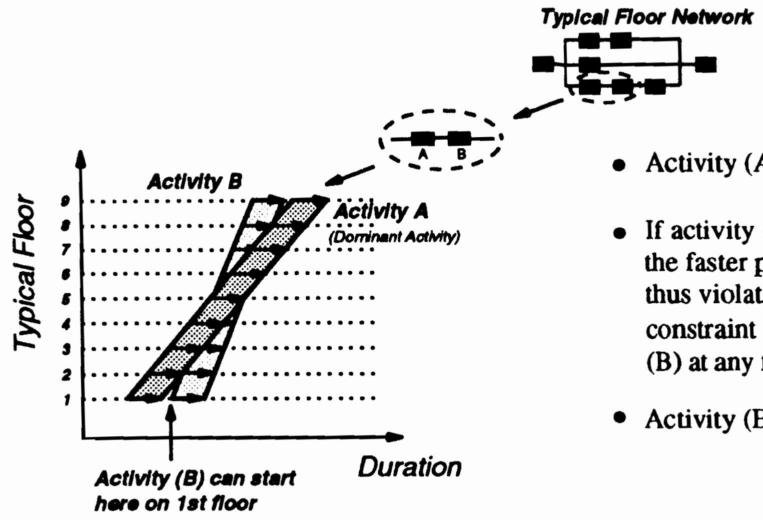
Figure 5.4 - Effect of Number of Crews on Productivity Rates

production rate of activity (A) to 0.5 floor/day and 1 floor/day, respectively.

It should be noted that Figure 5.4 also demonstrates one of the advantages of using graphical techniques in scheduling repetitive work. The flow line graph reveals the relationship between activity duration, number of crews assigned to the activity, and the production rate. Given the quantity of work to be performed, the planner can determine the suitable configuration of crew number and composition to be designated to each activity in order to achieve desired production rates.

In establishing the production rate for each activity, the scheduling procedure should insure that the rate conforms with the pace of the *dominant activity*. Also termed the controlling trade, a dominant activity is the activity that requires the longest duration for its execution. It dominates the production rates of all succeeding activities until the occurrence of an activity with a longer duration. In multi-story construction, the first dominant activity is the construction of the skeleton. As a result, it is advantageous to attempt to balance production rates of succeeding activities to that of a dominant activity in order to achieve an optimum schedule. As illustrated in Figure 5.5, making decisions to schedule a fast activity B with a high production rate succeeding a slower activity with a lower production rate will affect different parameters of the activity. Such parameters include the starting time of B and its continuity.

As shown in Figure 5.5(a), activity B can not be scheduled to start at time t_1 immediately

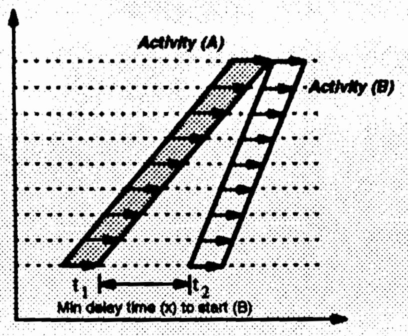


- Activity (A) is the dominant activity with the slower production rate.
- If activity (B) is started on floor 1 exactly after activity (A) is completed, the faster pace of (B) will result in making (B) ahead of (A) in higher floors, thus violating the technological constraints defined by the floor network. This constraint defined by the network implies that (A) should be completed to start (B) at any floor.
- Activity (B) should be rescheduled with the two alternatives shown below.

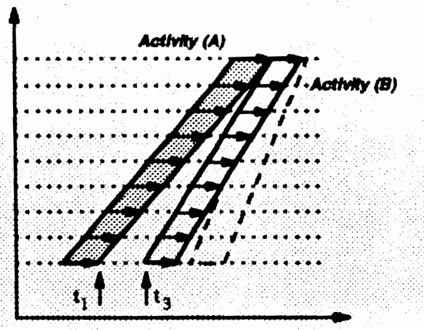
a) Initial state.

Option 1

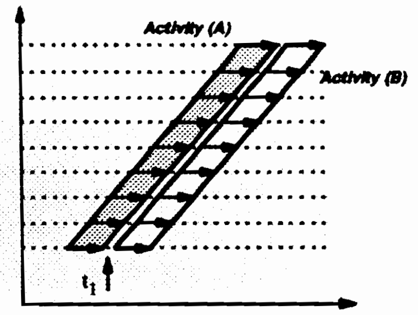
Option 2



b) Delay start of activity (B) by a minimum of x days to allow (B) at last floor to start after completion of (A) while maintaining the normal rate of production.



c) Decrease pace of activity (B) to start activity at an earlier time period (t3) to conform partially with the pace of (A).



d) Reduce pace of activity (B) to conform completely with that of activity (A) and start at the earliest possible time (t1).

Figure 5.5 - Scheduling Alternatives to Overcome the Dominant Activity Pace

after A is completed on the first floor. Because B has a faster rate of progress, this will result in interruptions at subsequent higher floors. To schedule B continuously with its normal production rate, one alternative is to shift the start date of the activity from time t_1 to time t_2. This will insure that start of activity B at any other floor remains succeeding to activity A. This alternative is established by projecting the production progress curve of (B) backwards from the end of the dominant activity A at the last floor. This is illustrated in Figure 5.5(b). To start activity B earlier, the scheduling procedure should make decisions to modify or balance the rate of B partially or completely with that of A. This may be achieved by decreasing, if possible, the production rate of B. Reducing the production rate of B will allow it to start at an earlier date t_3 as illustrated in Figure 5.5(c). If activity C can be scheduled with a production rate equal to that of A, activity B can then start immediately as A is completed on the first floor as shown in Figure 5.5(d).

A third alternative to reduce the starting time of B while maintaining its normal rate of production is to interrupt its continuity. This interruption is achieved by splitting the activity one or more times along the typical floors. Activity splitting is discussed further in section 5.3 below.

Variable Crew Production Rates

All the previous scheduling approaches discussed in section 5.1 above for modeling repetitive work have assumed that the production rates of the crews remain constant along the typical floors. Once a rate is determined for an activity to start at the first floor, these approaches assume that the activity and crew proceed along all the floors with the same rate. In a research work on linear construction projects, [Higazi, 89] showed that given a fixed number of crews allocated to an activity, the productivity rate of the activity will vary as the work progresses upwards and does not remain constant. This was found to be a result of many factors, but mainly due to the combined effect of learning curve phenomena and increased travel time associated with this type of construction.

Learning curve: In very general terms, the more times an operation is performed, the shorter will be the time to perform it. For example, the tenth of ten identical concrete slab pours should take less time and be done more skillfully than the first. This is because skill and productivity in performing tasks improve with experience and practice.

The basic concept of the learning curve phenomena indicates that the time required to execute a repeated task will be reduced resulting in a time improvement. This trend of time reduction (or improvement) due to learning development will eventually approach a minimum value as the task is repeated for a number of iterations.

Although different models have been developed to illustrate this phenomena, there is no precise or definite model that reflects the exact behavior of such reduction. Figure 5.6(a) demonstrates an example of a hypothetical exponential model for representing the reduction in time for a repetitive task as a consequence of learning impact.

The repetitive nature of multi-story building construction enhances the opportunities for performance improvement through learning. The same activities or processes representing the construction of a typical floor are repeated for each floor. Similarly, the re-iteration of each individual process involves several repetitions of various involved tasks.

[Higazi, 89] showed that learning due to repetition of work tends to improve performance during the first few typical floors. Then the improvement fades to a halt during most of the remaining floors. Then a prominent decline in productivity occurs near the end of the work. This decline results mainly because of the labor force become pre-occupied with relocating to another project.

Increasing travel time: Multi-story construction depends entirely on lifting equipment to deliver workers, equipment, and materials to work zones. The higher the floor from the ground level, the longer is the duration for lifting equipment to deliver workers and materials. The relationship between travel time and the number of floors could be represented by an exponential function. [Higazi,89] showed that increasing travel time reduces productivity of construction crews as work proceeds to upper floors. This impact offsets the early gain by learning and results in an overall decline of productivity.

Due to the effect of learning curve phenomena and increasing travel time, the study conducted by [Higazi, 90] concluded that production rates are not constant along the typical floor. Rather, they vary as the work proceeds upwards with a decrease in value as crews move from a lower floor to the next.

The current research will consider the combined effect of the learning curve and increasing travel time. Varying production rates for each activity will be considered in the scheduling calculations. Functions defining the varying flow line curves will be hypothetical and approximate figures will be assumed. The exact manner of how the production rates varies for the different trades will be outside the scope of the this research.

5.3 Work Continuity Issues

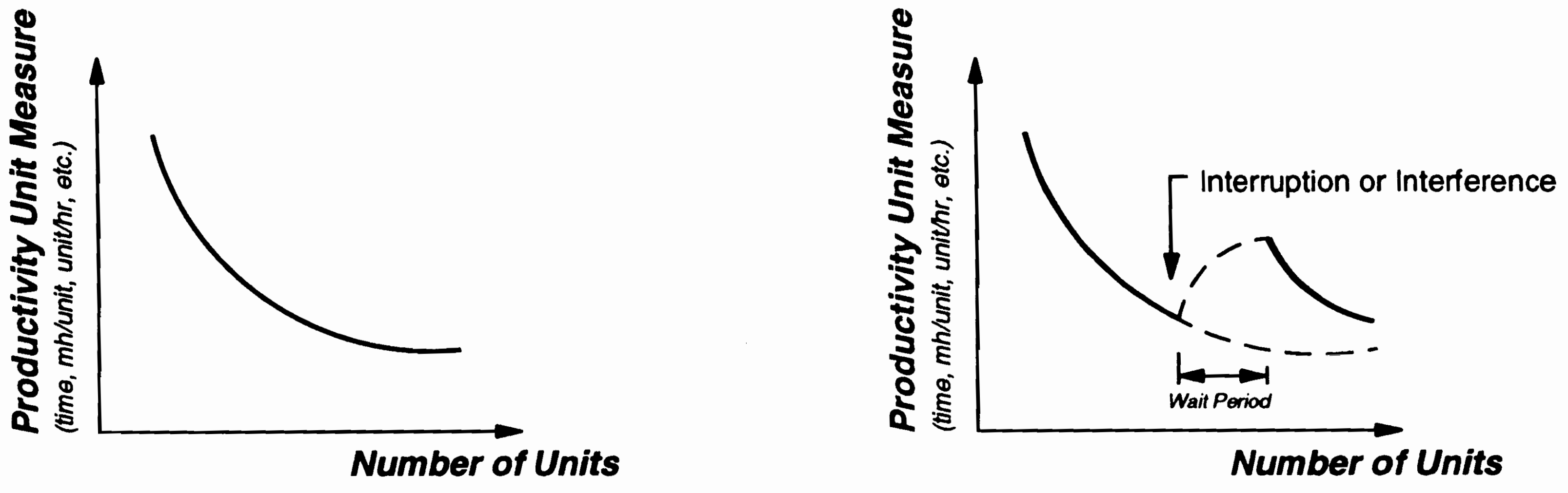
As indicated earlier, multi-story buildings are characterized by *repetitive working* in which a large segment of the facility (typical floors) is composed of a large number of activities or operations that are carried out repeatedly by the same crew or resource. If the planning and scheduling procedures allow for such crews to work continuously without stoppage or interruption from one typical floor to another, idle waiting intervals of equipment and manpower will be minimized resulting in considerable time and cost savings.

Improving continuity of the work will also maximize the learning curve effect and minimize delays and extra effort (e.g. setup time, temporary storage of tools/materials, etc.) associated with work interruptions. In addition, improving continuity reduces the impact of "come-back" delays. [Ashely, 80] pointed out that "come-back" delays occur when a crew (possibly a subcontractor) is not utilized continuously and elects to work at another site during slack periods. Return of the crew to the original site often involves delays of random duration. [Barrie and Paulson, 84] indicated that if interruption of or interference with repetitive operations occurs, an "unlearning curve" effect takes place which may affect project costs. This is illustrated in Figure 5.6. To control such costs (especially labor costs), it is generally good practice to maintain continuity on repetitive tasks.

To achieve continuity of flow for an activity, splitting of the activity should not be allowed. Once the activity is scheduled to start at the first typical floor it should continue with no interruptions to the last floor. In high rise construction, continuity of crews is essential in major trades such as structural framing, blockwork, external cladding, HVAC ductworks, etc.

Where continuity is not a significant issue, splitting of the activity should be permitted to achieve shorter project durations. Splitting is essential to minimize delays in activity start dates when balancing production rates with that of the dominant activity is not possible.

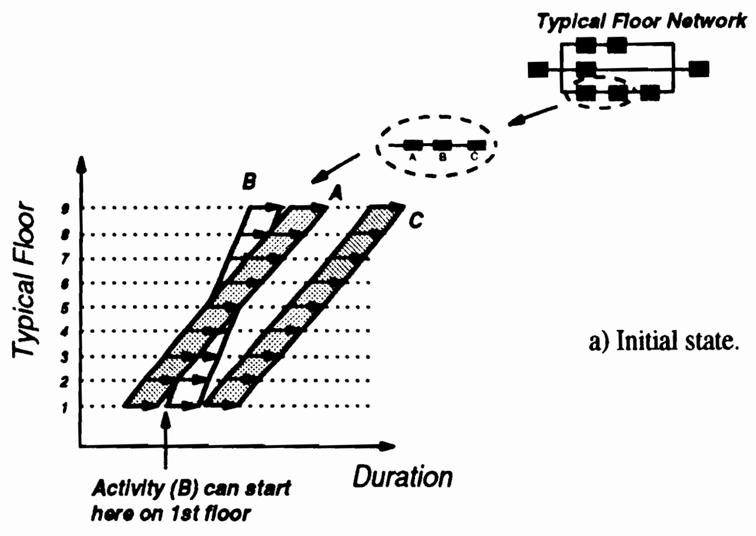
As shown in Figure 5.7(a), the start of activity C is dependent on the completion of B at the first floor. Activity B can not be scheduled to start immediately at the first floor once A is completed. Section 5.2 above presented two alternatives to overcome this problem.


Productivity Unit Measure
(time, mh/unit, unit/hr, etc.)
Number of Units
Productivity Unit Measure
(time, mh/unit, unit/hr, etc.)
Interruption or Interference
Wait Period
Number of Units

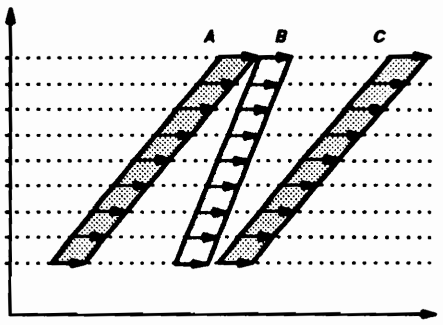
a) Learning curve effect

b) Unlearning curve effect

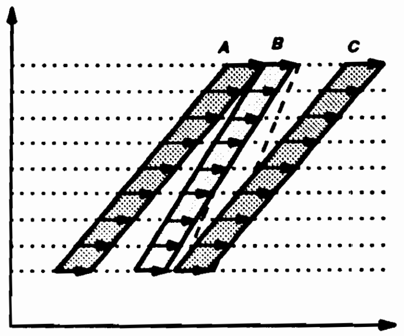
Figure 5.6 - Unlearning Curve Effect in Construction of Repetitive Operations



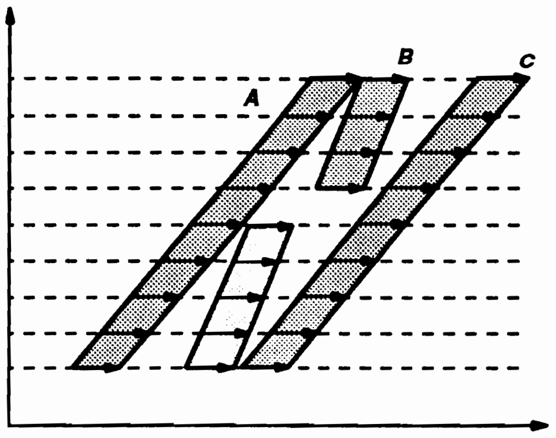
a) Initial state.



b) OPTION 1: Delay start of B maintining normal rate.



c) OPTION 2: Adjust production rate of B to minimize start delays.



d) OPTION 3: Split activity to minimize start delays while maintaining original production rate.

Figure 5.7 - Activity Splitting to Overcome Dominant Activity Pace

One alternative was to delay the start of B to a date that is appropriate to perform the activity continuously with its original production rate. The second alternative was to decrease the production rate of B, if possible, to a value up to that of the of the dominant activity A. These alternatives are again summarized in Figure 5.7(b) and (c).

Because the first two alternatives do not consider the splitting of the activity along the floors, they do have the advantage of maintaining continuity of flow. Most of the major trades activities favor these two alternatives as continuity of flow has an impact on reducing the costs of executing the activity.

The use of any of these two alternatives may not be suitable to all construction activities succeeding a dominant activity. The first alternative delays the activity start considerably. This might be favored if the later start were to move the activity into a more favorable time period. However, this alternative definitely delays the start of all the succeeding activities (significant for finishes), and the project may suffer a total delay equal to the delay period of starting the activity.

The second alternative may still delay the start of the activity if its production rate was not reduced to conform fully with that of the dominant activity. That is, if the rate of B remains higher than A after decreasing it to its minimum limit, a certain amount of delay of the start of B will have to occur in order to balance the difference in both production rates. This is because some activities may not require a duration long enough to conform with the pace of the dominant activity. Costs of performing the activity with such a lower rate than normal may be much higher and may have a financial impact on the project total costs. Such impact may offset the gain in starting the activity at an earlier time period. In some others, it may be impossible to reduce the rate below a minimum value as the crew composition can not be reduced less than a certain required size to perform the job.

For example, the use of only one labor to remove shutters (activity *Strike Formwork*) may still not reduce the pace of the activity to conform with the dominant activity. Yet, it would be impossible to reduce the rate any further. An added disadvantage of this second alternative is the resulting decrease of productivity of the crews. This may have a negative impact on their productivity in future tasks.

To overcome the problem, a third alternative suggests the splitting of the activity, hence sacrificing the continuity of flow. Some activities favor this alternative as resources can be shifted, during the wait period, to perform other work. In addition, delays of starting succeeding activities may be reduced or eliminated while overcoming the problem of the slow pace of the dominant activity.

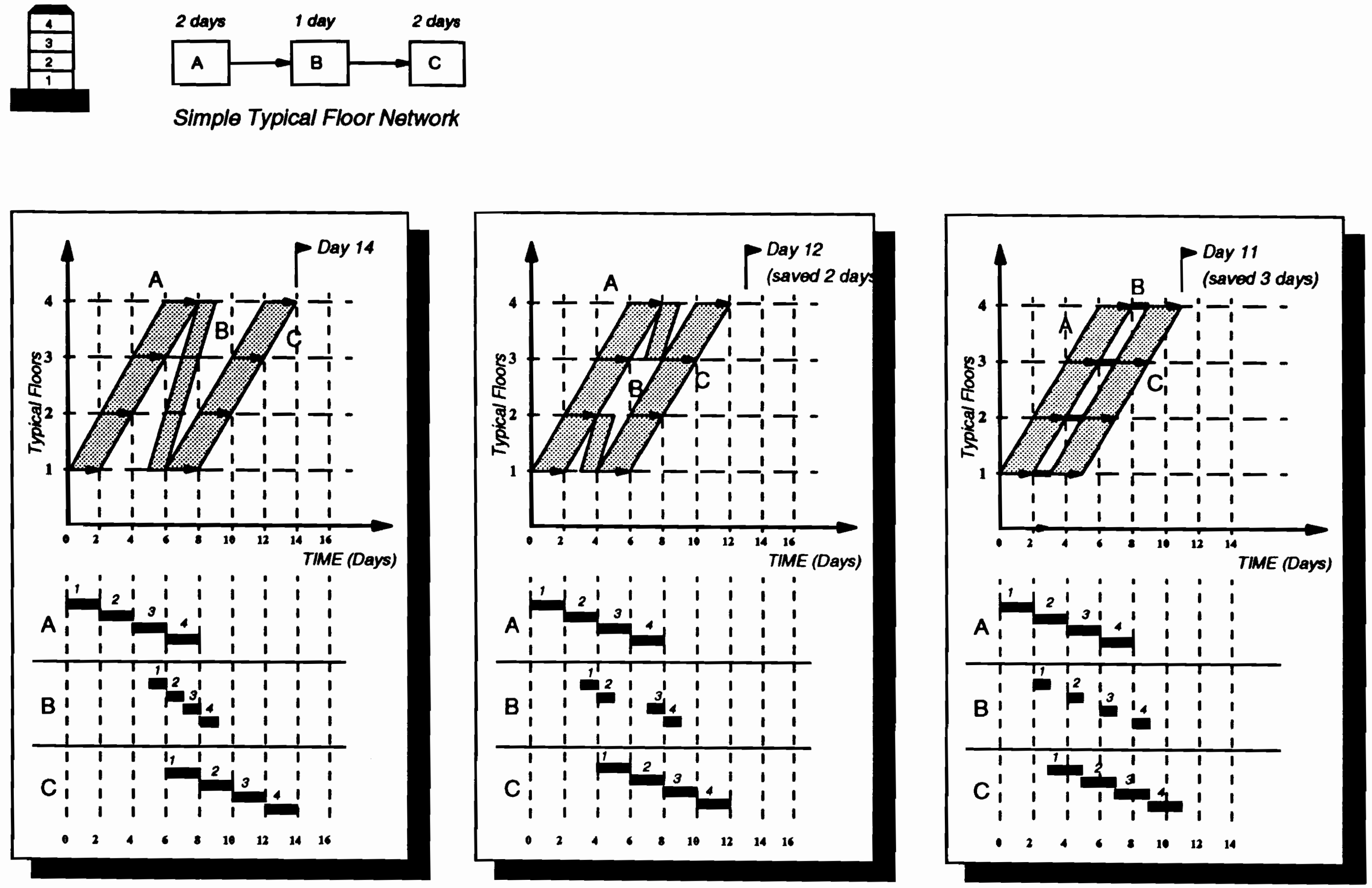
The number of times an activity is split along the floors will vary for each activity and is dependent on the desired start time to be achieved and cost issues. The more number of times an activity is split, the earlier an activity can start. However, increasing the number of splits increases work stoppage that results in higher construction costs.

The effect of the number of splits on total project duration is illustrated in Figure 5.8. In Figure 5.8(a) splitting of activity (B) is not permitted. In Figure 5.8(b) and (c) splitting of the same activity is permitted. The activity is split twice and three times and the resulting schedules are 2 and 3 days shorter, respectively.

5.4 Vertical Logic

In construction of high rise buildings, the execution of some activities affect the start or completion of other activities at different floors. The dependency of the execution of an activity on another activity at a different floor represents a scheduling constraint. Satisfying such a constraint establishes a vertical logic between the two activities. This is opposed to the horizontal logic defined among the activities of the same floor determined by a logical floor network.

The general concept of vertical logic is illustrated in Figure 5.9. As shown in Figure 5.9(a), an activity B can start at any floor only after the completion of activity A at the same floor. This constraint is determined by the horizontal logic network defined among the activities of the same floor. If for some reason activity B can not start at floor$_1$ before the completion of A at floor$_5$, then this constitutes a vertical constraint between the two activities. To satisfy such a constraint, the start of activity B is shifted to start at time t_2.

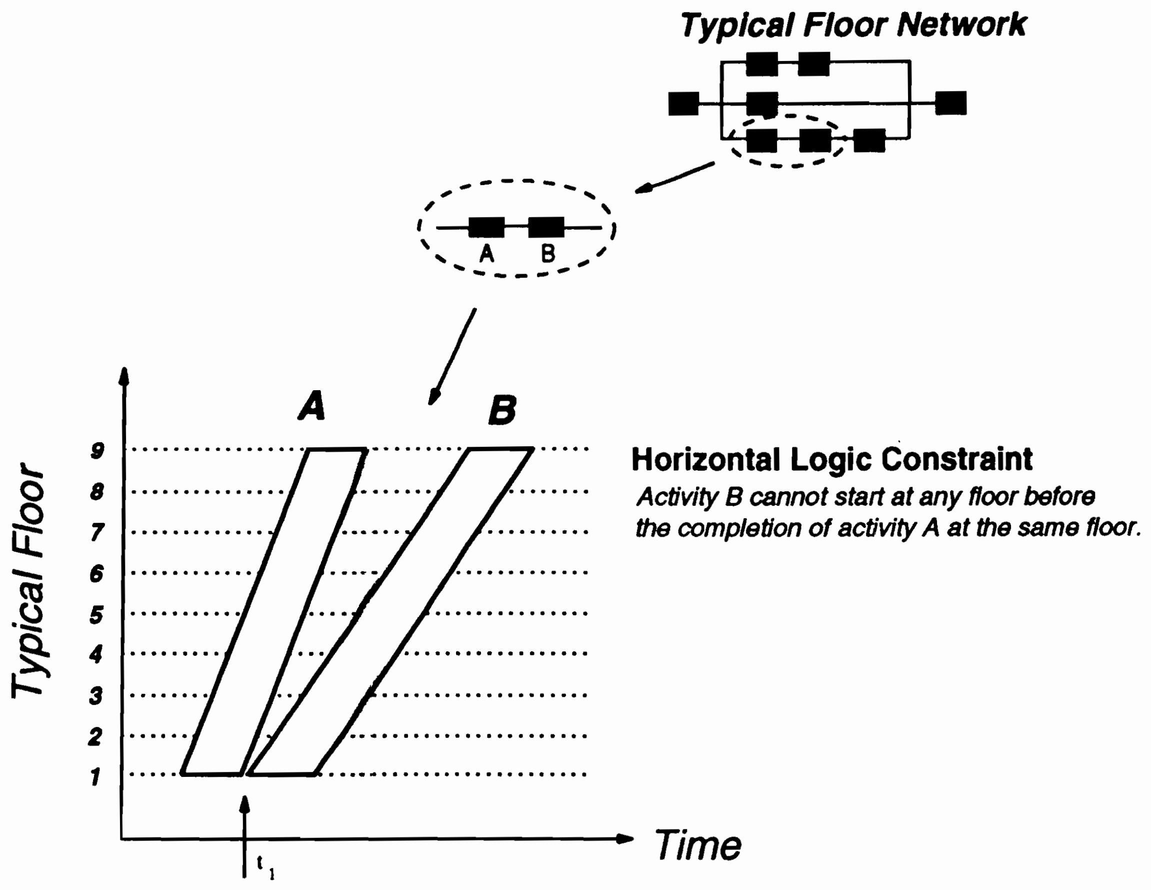


a) Splitting of Act. (B) not permitted

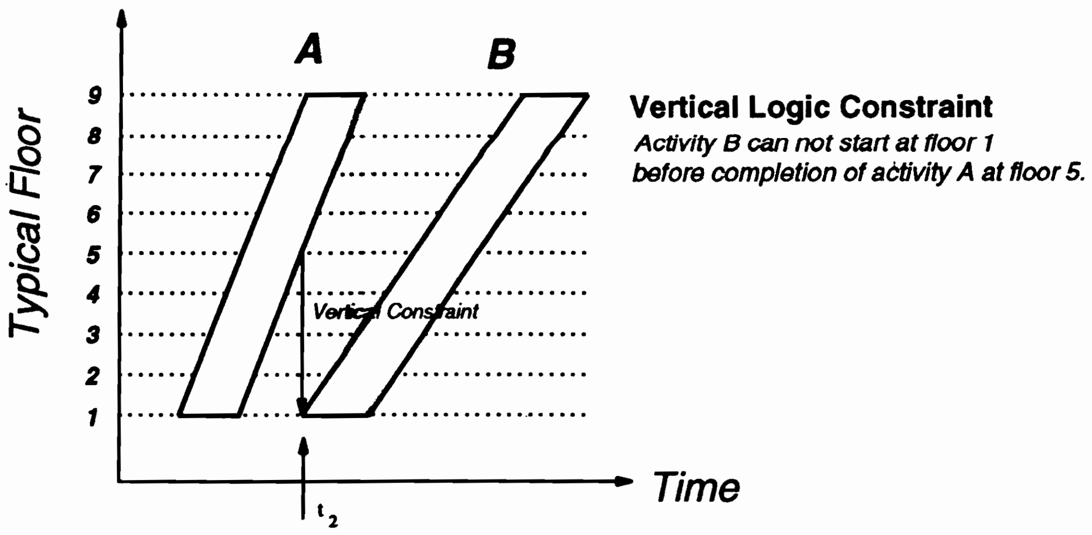
b) Splitting of Act. (B) permitted (activity split twice)

c) Splitting of Act. (B) permitted (activity split three times)

Figure 5.8 - Effect of Activity Splitting on Project Total Duration


Typical Floor Network
A
B
A
B
Horizontal Logic Constraint
Activity B cannot start at any floor before the completion of activity A at the same floor.
Typical Floor
9
8
7
6
5
4
3
2
1
Time
t_1

a) Schedule controled by horizontal logic only.


A
B
Vertical Logic Constraint
Activity B can not start at floor 1 before completion of activity A at floor 5.
Typical Floor
9
8
7
6
5
4
3
2
1
Vertical Constraint
Time
t_2

b) Schedule controled by horizontal and vertical logic.

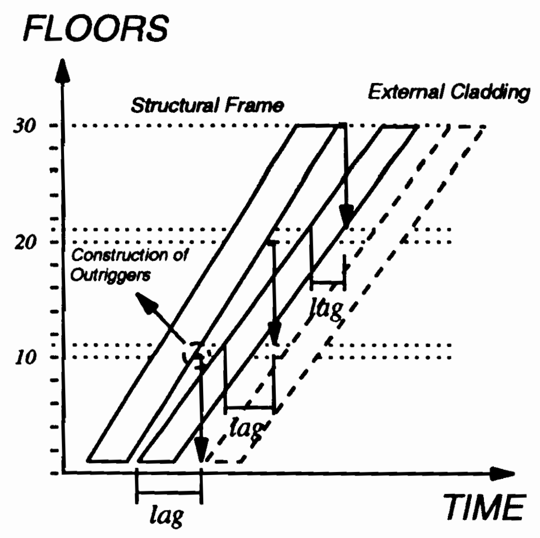
Figure 5.9 - Vertical Logic

This is illustrated in Figure 5.9(b).

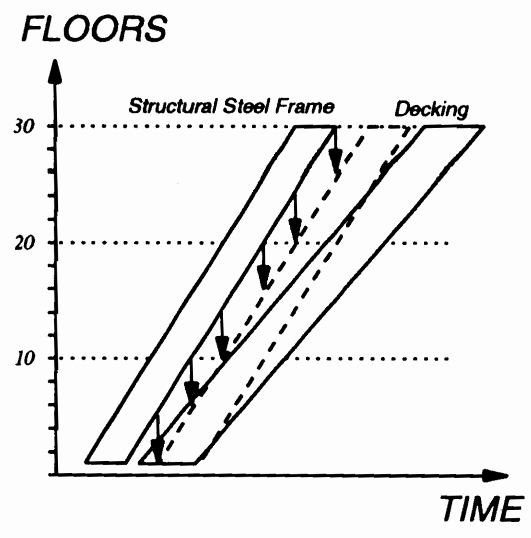
Vertical logic constraints may be a result of many factors such as: construction issues, safety and code regulations, weather conditions, etc. Figure 5.10 illustrates different examples of vertical logical constraints resulting from such factors. Figure 5.10(a) depicts a construction constraint for the external cladding or masonry work. This activity requires scaffolding to be erected along the external side of the building. The key to the erection process is to suspend the scaffolds from outriggers tied to the building frame. Each outrigger would support the scaffolds for a certain number of floors, say 10 floors. That means that scaffolding on the first floor will not be ready for cladding work until the outrigger on the tenth floor is erected. This in turn dictates that the structure frame at the tenth floor should be completed to allow the erection of the outriggers. Similarly, scaffolding at the eleventh floor can not start until the erection of outriggers on the twentieth floor is completed, and so on.

This construction constraint implies that a vertical lag or buffer of 10 floors should be maintained between the cladding or masonry work and the construction of the outriggers and the structural frame. The scheduling procedure should insure that at any floor, the construction of the frame is always ahead of the external cladding and masonry by at least 10 floors to insure this vertical construction constraint.

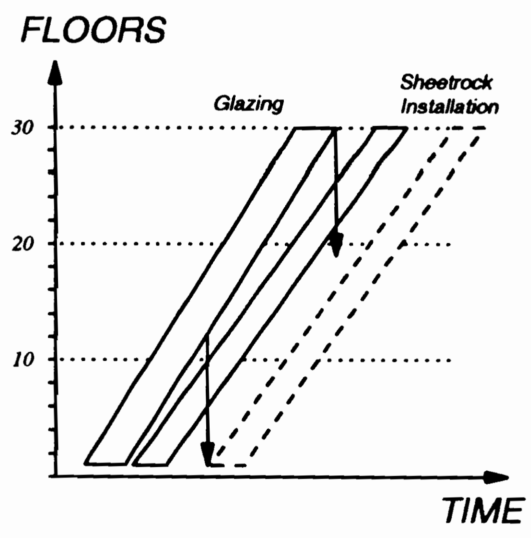
Figure 5.10(b) illustrates a second example of vertical constraint that is a result of code



a) Vertical constraint related to construction issues.



b) Vertical constraint related to safety and code regulations.



c) Vertical constraint related to weather tight requirements.

Figure 5.10 - Examples of Vertical Constraints

regulations imposed on the construction process for safety reasons. Because of safety standards, code regulations requires a horizontal diaphragm following not more than two stories behind the steel erection crew in steel framed structures. In practice, this dictates that the installation of the metal deck should follow not more than two stories behind the erection of the frame. This vertical constraint implies that a vertical lag or buffer of a maximum of two floors should be maintained between the steel frame and the construction of the floor deck.

A third example of vertical logic is illustrated in Figure 5.10(c). This constraint results from restraints due to weather tight requirements. For activity sheetrock to proceed, the building has to be significantly enclosed. Horizontal logic in the typical floor network insures that glazing and brickwork activities are completed before sheetrock installation commences in the floor. This eliminates the problem of water coming horizontally through the building. The other problem is that of water coming vertically through the building from upper floors where glazing and waterproofing are not completed. As a result, the sheetrock installation at any floor is required to be behind glazing and waterproofing by a certain number of floors to prevent the vertical flow of water that may result in damage to the sheetrock.

The scheduling procedure should insure that the glazing and waterproofing works are ahead of the sheetrock installation by a specific number of floors to provide a water tight environment from the vertical direction.

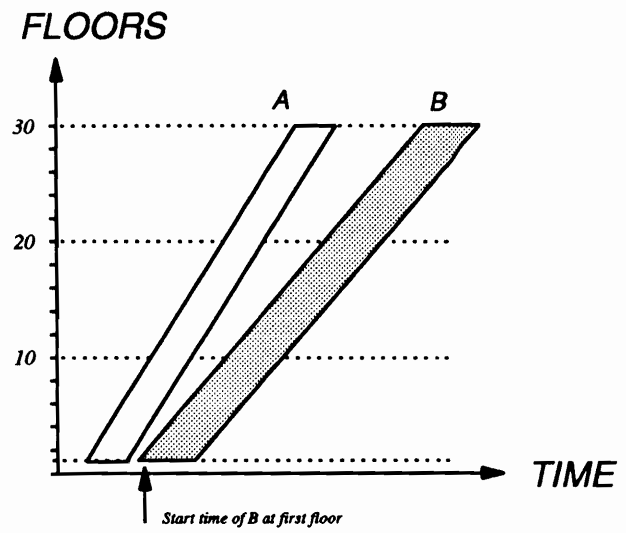
5.5 Workflow Direction

As indicated earlier, construction work in the typical floors of multi-story buildings involves many repetitive units performed consecutively by the same crew. This work repetition characteristic forms chains of activities with each chain performed with the same crew. Each chain of activity posses a spatial orientation or workflow direction.

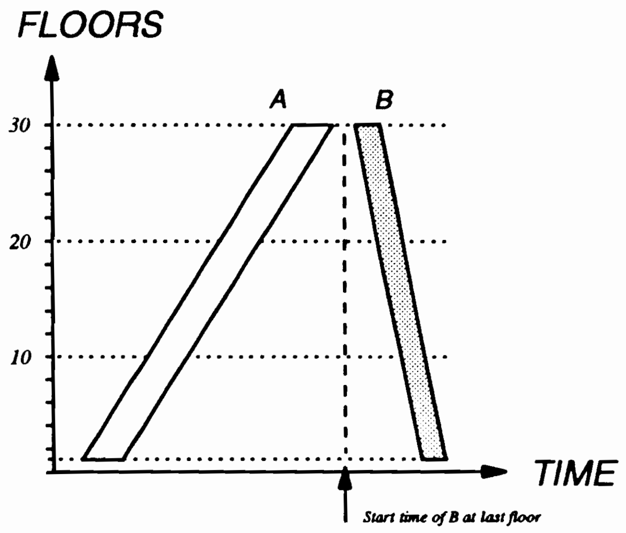
The direction of flow is either upwards or downwards. Most construction work is scheduled in an upward direction to follow the construction of the skeleton. The nature of construction work in the core area mandates an upward scheduling of its activities. Activities such as electrical and plumbing risers work requires that in order for the activity to start at any floor, the work in the floor below must be completed.

Some other activities, however, must be scheduled in a downward direction due to reasons of safety or to prevent damage of the installed work. Final finishing of external blockwork or cladding and clearing and cleaning may be typical examples of such type of activities.

The effect of work flow direction on the scheduling of activities is illustrated in Figure 5.11. As shown in Figure 5.11(a), activity B can start after activity A is completed at the first floor if workflow direction of B is upward. On the other hand, if the workflow direction of B is downward, B can not start until activity A is completed at the last floor.



a) Activity (B) has the same workflow direction as activity (A).



b) Activity (B) has the opposite workflow direction of activity (A).

Figure 5.11 - Effect of Workflow Direction on Activity Scheduling

5.6 Chapter Summary

The chapter presented some of the important issues and characteristics associated with multi-story building construction that influence the planning and scheduling process.

A general review of current planning and scheduling methods for repetitive work was presented. Numerous specific techniques have been developed to schedule linear projects. The general characteristics of these techniques were discussed.

Continuity issues and their importance in scheduling multi-story buildings were discussed. The repetitive nature of this type of construction projects demands the consideration of the continuity of flow in the scheduling process. Insuring continuity reduces idle waiting time of manpower and equipment. Continuity also maximize the learning curve effect associated with repetitive work. As a result, time and cost may be reduced.

The chapter also presented the importance of production rates as a determining factor in generating the schedule. Balancing production rates of activities succeeding a dominant activity provides for minimizing delays of activity starting time and at the same time maintain continuity of the flow. Where balancing of production rates is not possible, activity splitting can be used to minimize such starting delays.

The chapter also discussed the vertical logic as a governing factor affecting the sequencing of activities. In addition to the horizontal logic stated by the logical network among the floor activities, vertical logic enforces a second type of constraint that governs the schedule of some activities along the typical floors. Different examples of vertical logic constraints were presented.

The importance of workflow direction in activity scheduling was also presented. Many repetitive activities performed along the typical floors of multi-story buildings posses a unique workflow direction. The direction of flow of each activity is either upwards or downwards through the building.

6.0 PROJECT SCHEDULING WITH RESOURCE CONSIDERATIONS

Once a plan has been evolved it will be necessary to draw up schedules by which the demands for resources will be satisfied. Many of the constraints hedging in a plan will have origins in the limitation of total resources or in the proportion of total resources that can be committed to a project. In consequence, plan and resource scheduling becomes very mixed to the detriment of the plan which becomes reduced to "what *can* be done" instead of declaring "what *is* to be done".

Time-only or time-based project scheduling techniques, such as PERT and CPM, rely on logical precedence among the activities defined by technological requirements to perform basic scheduling computations. Such scheduling techniques do not consider resource requirements and matching these requirements with resource availability. Resource scheduling techniques consider availability of resources as an integral factor in establishing project schedules. The matching of resource requirements with resource availability is the main task in resource scheduling processes.

This chapter presents the concepts of the resource scheduling problem. The chapter also reviews and discusses the different methods and techniques that have been developed to deal with this scheduling problem. If the reader is familiar with these concepts and methods, he or she may wish to skip this chapter without loosing any continuity.

6.1 The Resource Scheduling Problem

The execution of a project requires the deployment of different resources over a specified time duration. These required resources usually fall into many non-interchangeable categories and the time available may be of limited duration and interrupted by weekends, holidays, etc. The available resources may be fixed in quantity or can only be changed slowly and it may be necessary that they are shared between several projects.

In addition to the large amount of resources involved, methods of use of these resources will vary. Men and equipment cost money whether they are working or not. Materials can stand idle (for a limited time) without adding significantly to project costs. Other resources may only be 'borrowed' for the project (e.g. scaffolding, formwork shutters, etc.) and returned to communal 'pool' after use.

Adding to the problem is the variable flexibility of project completion time. In many projects, the end date is fixed; while in others, completion is usually acceptable over a range of dates.

This interaction between resource required and resource available have been considered in several models. These models can be grouped into two major categories according to the problem addressed:

1) Resource Leveling (Smoothing); and
2) Fixed Resource Limits Scheduling.

6.1.1 Resource Leveling (Smoothing)

This problem arises when it is possible to procure sufficient resources to carry out a project which must be completed by a specified due date, but it is desirable or necessary to reduce the amount of variability in the pattern of resource usage over the project duration. The objective is to smooth, as much as possible, the demand for each specific resource during the life of the project. This type of situation is very important where the cost of hiring, training and/or laying off of personnel or physical resources can be substantial.

6.1.2 Fixed Resource Limits Scheduling

Also often called *Constrained-Resource Scheduling* or *Limited Resource Allocation*, this category of problem is much more common and arises when resources required for performance of project activities are available in fixed limited amounts such that the demands of concurrent activities can not be satisfied. Under these conditions, activity sequencing decisions are required often with a resultant increase in project duration beyond the initial project duration determined from the standard time-only PERT/CPM methods. In this type of resource problem, the objective is to minimize project durations subject to stated constraints on available resources.

This category of problem can be further divided according to whether the fixed limits on resource availabilities are constant at some level, or allowed to vary over activity or project duration. A further useful subdivision is possible according to the method used for solving the problem (i.e. approximate heuristic methods or exact mathematical procedures).

Figure 6.1 depicts the resource scheduling problem.

6.2 Factors to be Considered in Resource Scheduling

Several factors should be considered in project scheduling with resource considerations. These factors include:

1- Resource Requirements

The amount of resource needed for executing a job is known as *resource*

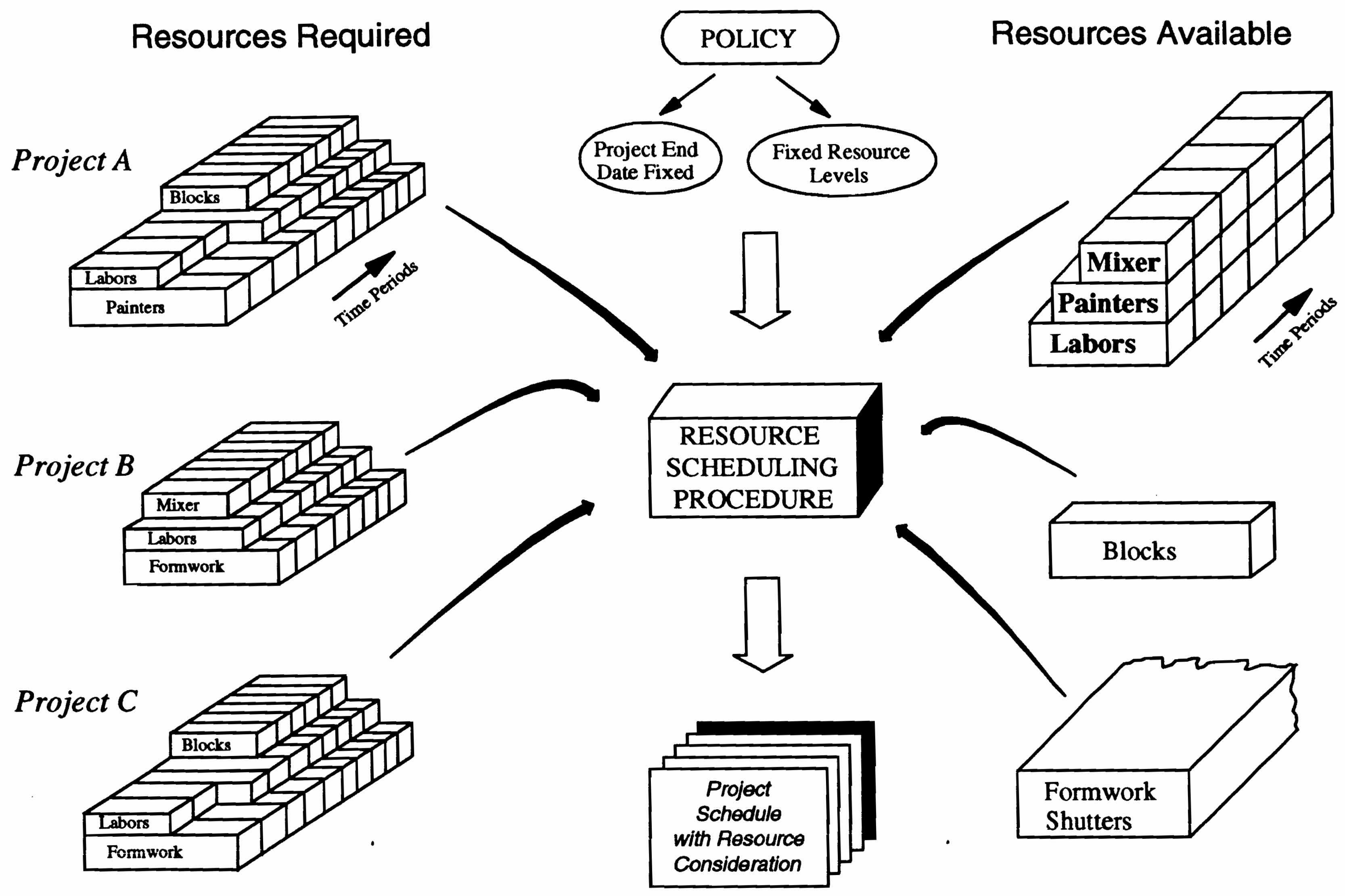


Figure 6.1 - Interaction Between Resources Required and Resources Available.

•

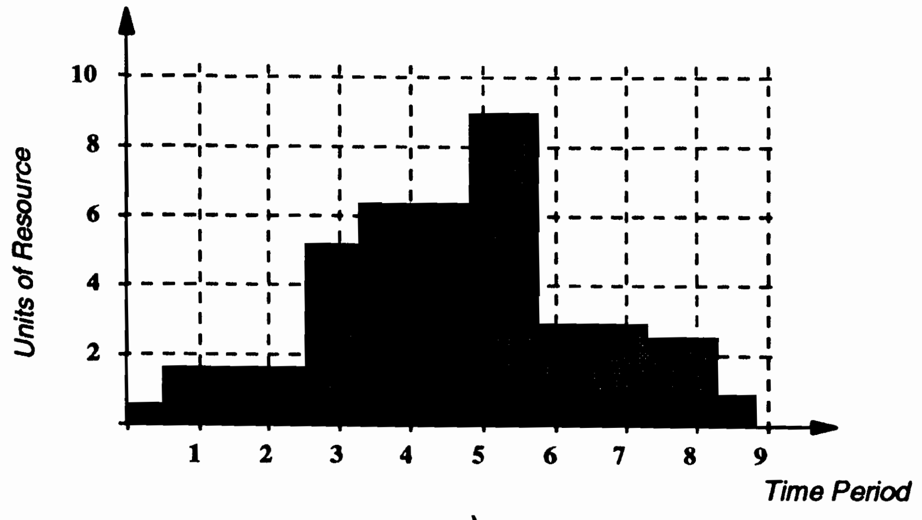
requirements.

When defining the resources required for each activity, it is necessary not only to specify the type of resource (e.g. bricklayer, laborer, crane, etc.), but also whether it is required for all the activity duration or only part of it. Also, it is necessary to define
whether the amount of resources defined is specific (i.e. constant) or whether the requirement will increase if the activity duration is increased (i.e. variable).

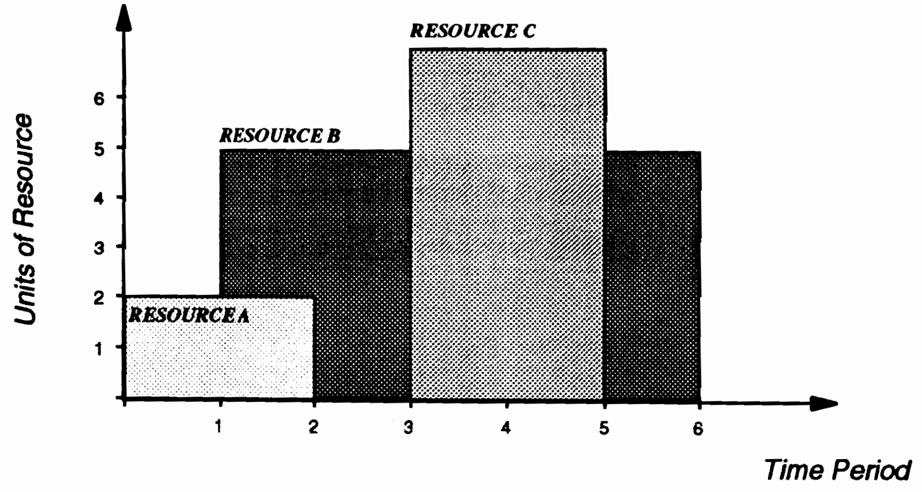
•

When the resource requirement (e.g. number of labors) are uniform over the entire activity duration,, the resource requirements are said to be *rate constant.* This is because if the activity time is increased the same number of labor is still required for the extra time. The total requirement of some resources such as materials, however, will not increase because the activity duration increases and these can be considered as *total constant* resources.

Resource requirements, when variable, are best shown through histograms. A histogram is a graph in which the requirements of a resource are shown with reference to time: time is represented by the x-axis (abscissa) and the resource by the y-axis (ordinate). The requirement for each period is shown through a vertical bar over the relevant time period. Figure 6.2a illustrates an example of a resource requirement histogram.



a)



b)

Figure 6.2 - Example Histograms Showing Period Requirement of Resources.

A further consideration in activity resource requirements arises from the situation that any one activity may require several different resources. Also, each resource specified for any activity may only be required over part of the activity duration.

Diagrammatically, this typical complex resource requirement is illustrated in Figure 6.2b. Each resource is displayed against the part of the activity duration for which they are required. Such information is necessary and essential for resource allocation procedures in order to be able to make decisions as to whether schedule the activity to start when partial or all the required resources are available.

2- Resource Availability

Resource availability is defined as the *amount or quantity of a resource which is available or can be made available.* It indicates the limitation in the availability of the resource(s).

Resource availability may be constant for a period of time. For example, it may be indicated that a maximum of 6 carpenters, 4 fitters and 20 laborers would be available for a particular job for a certain period of time. Most often, resource availability is variable. In this case it may be expressed in terms of distinct time periods or be represented in the form of a histogram.

Two methods of defining resource availabilities can be found. These are: a) normal and threshold limits; and b) pool resources.

Normal and Threshold Resource Limits

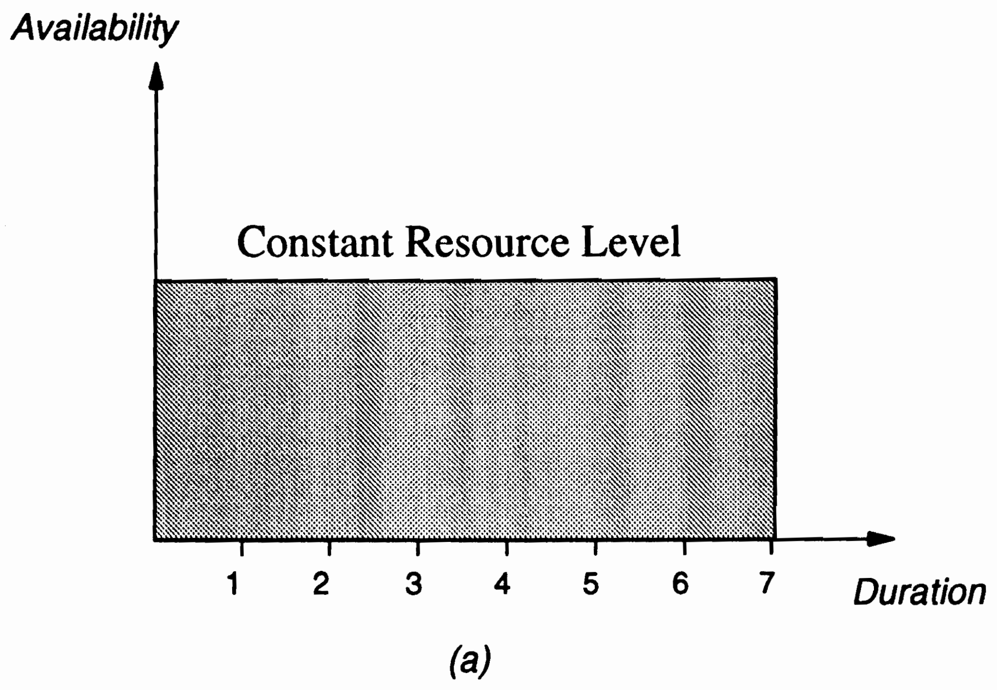
Resource availability can seldom be represented by the constant line shown in Figure 6.3a. Resource levels are usually cyclic with a five to six days' work and two or one day of no work (weekends). In some cases, lower resource availabilities occur on Mondays and Fridays due to absenteeism. Public and annual holidays add further complexity to the scheduling process. Furthermore, resource levels themselves are not finite and absolute (normal versus overtime).

To account for this variability, resource availabilities can be defined by specifying two levels, the normal level, and the maximum or threshold level. This is shown in Figure 6.3b.

Pool Resources

Normal and threshold resource limits are levels set at each particular period of scheduling time. Resources defined in this fashion can not be transferred from one period to the other. Pool resources, on the other hand, represent resource levels which are not conditioned by time. They can be carried forward from one period to the next.

For example, if 10 men are allocated to a project for two periods, the fact that only



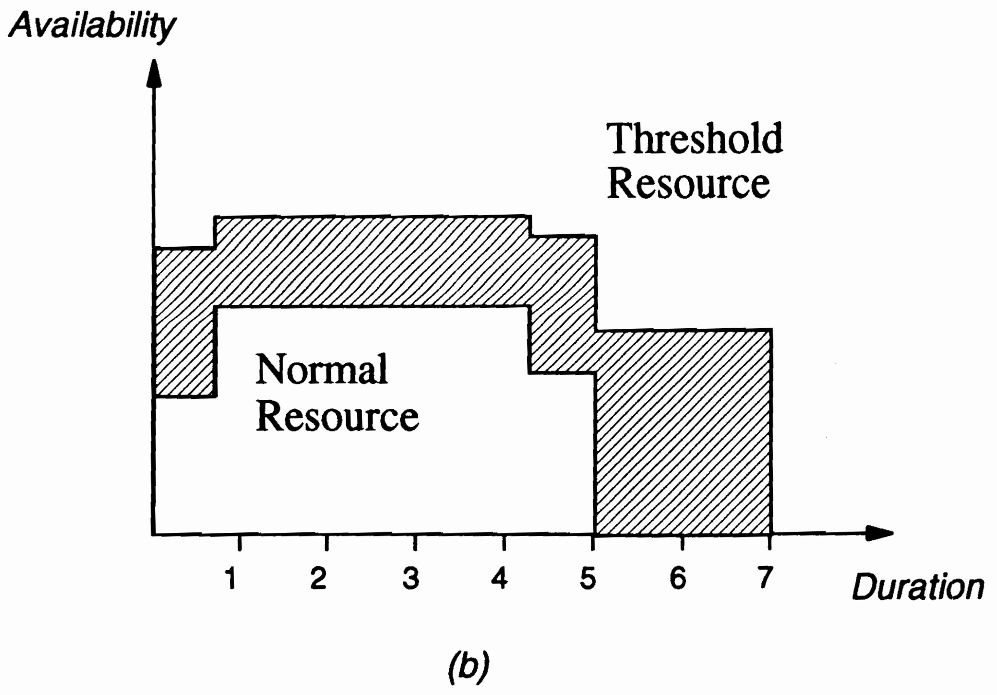


Figure 6.3 Normal and Threshold Resource Availabilities

five are utilized in the first period does not mean that 15 will be available for the second period; on the other hand, if the resource being considered is bags of cement, then 15 would be available in the second period as the unused quantity can be carried forward.

This method of defining resources is particularly suitable for scheduling resources which are transferrable through time such as material and working space.

3- Work continuity

In any scheduling operation, there are always pockets of resources which are available over too short a time period. Where it is possible to carry out different parts of an activity separately, more efficient use can usually be made of the available resources, with a consequential reduction of project time. One factor to be considered is whether an activity may be split up into sub-activities (intermittent activity), or whether once started it must be worked on continuously (continuous activity).

4- Consecutive activities

Similar to the splitting of activities for ease of scheduling, activities may be separated in time from their preceding and succeeding activities if the available float time allows and resource availabilities necessitates it.

In some cases, however, practical considerations demand that successive activities be linked together when the work schedule is constructed. These are generally known as *consecutive activities.* This linking means that dates have to be found where there are sufficient resources available for the continuous scheduling of the series of activities linked in this way.

5- Resource Aggregation

In the majority of construction projects, many activities use common resources. As some of these may be concurrent, it becomes necessary to know the cumulative requirement of each of the resources, for the project taken as a whole, period-by-period. This is obtained by what is known as the *resource aggregation* exercise.

Resource aggregation involves the summing up or totalling of the requirements of a particular resource for all the activities in a project, for each time period (e.g. week/day/hour), of the project duration.

The cumulative resource requirements, obtained through resource aggregation exercise, is represented commonly through histograms. These are also known as *resource profiles*. Resource profiles are of great value in resource scheduling as they show the requirements of a resource with respect to time in an easily understandable way.

6- Resource Utilization

Resource utilization is defined as *the amount of a resource that has been (or is being) utilized.* Utilization of a resource is expressed as the *resource utilization factor*. This is the ratio of the resource required to resource employed. The resource utilization factor can be useful as a pointer to the efficiency of the resource deployment.

7- Project duration threshold

Project completion is not often fixed but may be acceptable over a range of dates. A method of expressing flexible end dates is to set a *preferred completion date* and a *maximum permissible completion date*. The difference between these two dates is known as the *project duration threshold.*

Figure 6.4 shows the alternative areas in which activity scheduling can take place and the boundaries which may be set. With reference to Figure 6.4, normal scheduling will take place in area 1, provided that the amount of resources required does not exceed the normal resource level and that the inability to schedule (because of non-availability of resources) does not delay completion beyond the preferred completion date.

Where either resource requirement or time requirement exceeds the boundary, then alternative schedules are possible. Thus, a schedule may be calculated which uses

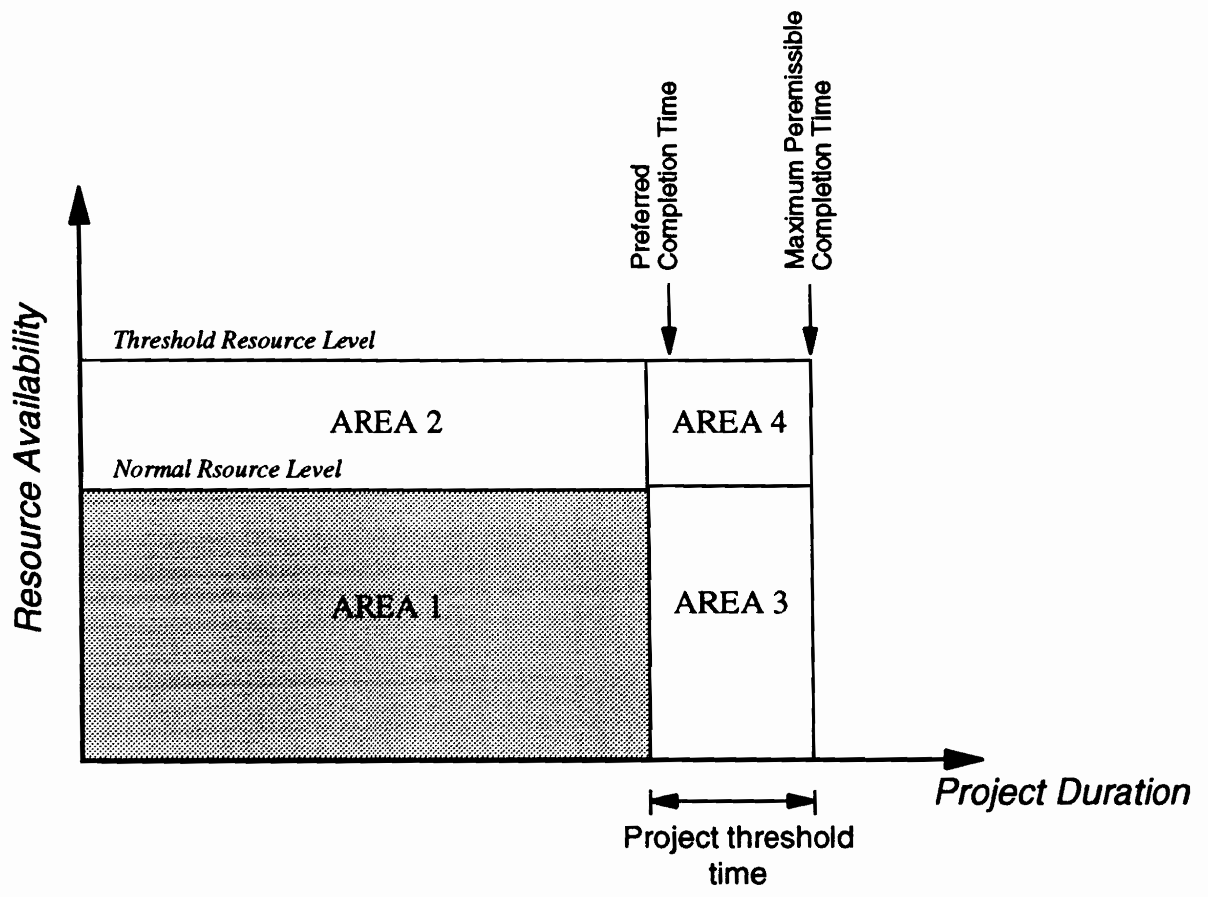


Figure 6.4 - Alternative Scheduling Within Normal and Threshold Project Times.

either areas 1 and 2, or areas 1 and 3. Further, non-availability of resources may cause some of all areas 1, 2, 3, and 4 to be used.

The sequence in which the boundaries of area 1 are exceeded is decided by the type of analysis which is undertaken. For example, a Resource Leveling would use threshold resources before threshold time; while a Fixed Resource Limits Scheduling would use project threshold time before threshold resources.

6.3 Scheduling Techniques with Limited Resource Considerations

The resource scheduling problem falls into a category of mathematical problems known as *combinatorial* problems. This is because, for any given scheduling problem, a very large number of possible combinations of activity start times exist, with each combination representing a project schedule. The number of combinations is extremely large, even for a fairly small networks of 20 to 30 activities and increases rapidly with an increase in the number of activities.

Several approaches are used to solve project scheduling problems with resource-constraints. These approaches may be classified into:

- *analytical (or algorithmic)* methods that search for the best schedule; and
- *heuristic* methods that search for a good schedule.

In the analytical methods, the various constraints in the availability of resources and other conditions (or polices) are expressed in terms of quantified relationships or mathematical equations or inequalities. The values of variables are obtained through solution of the various simultaneous equations.

The combinatorial nature associated with this resource scheduling problem has prevented it from yielding readily to the solution techniques of the analytical methods. An optimal solution, though theoretically possible, may be difficult to achieve using these methods. In many cases, it may be difficult to set appropriate mathematical relationships to represent the various objectives and constraints.

Solution attempts by a variety of such methods have been successful on at most only smaller sized projects. Because of the relative lack of success with these analytical techniques, major efforts in attacking the problem have been expanded in developing heuristic procedures which produce a "good" feasible solutions. Heuristic methods are those in which certain empirical decision rules are specified for arriving at rational allocation of resources.

In limited resource scheduling, where a number of activities are competing for a scarce resource, it becomes necessary to make a choice between the various activities to which the limited resource is to be allocated. One of the decision rules may be about the order in which the resources should be allocated. This could be accomplished by specifying criteria for ranking the activities. For example, it may be specified that the resource would be allocated first to the critical activities, and then to other activities, in order of increasing float.

Likewise, in resource smoothing, it may be indicated that the resources should first be diverted from activities in descending order of free float and after that from other non-critical activities, as per descending order of their total float.

In short, the general approach using these methods is to specify priorities according to which the resources are to be allocated to various activities. Relatively simple criteria such as "shortest job first," or "minimum slack first" are examples of heuristic used to establish activity priorities. There are in existence today hundreds of different heuristic-based procedures. Some have been included in sophisticated scheduling packages [Asad and Wasil, 86].

Heuristic methods do not, in general, yield optimal solutions. However, if the decision rules (criteria) are set properly with the requisite amount of thought, these methods are capable of yielding reasonably good practical solutions.

Analytical methods and heuristic methods are discussed below in more detail in sections 6.3.1 and 6.3.2 respectively.

6.3.1 Analytical-based (Optimization) Methods

This category of resource scheduling procedures for producing optimal solutions has been characterized by relatively less practical progress than the heuristic methods. The optimization procedures which have been developed can be divided into two subcategories:

1. procedures based on linear programming (LP); and
2. procedures based on enumerative and other mathematical techniques.

When first introduced in the early 1960's for solving the constrained-resource scheduling problem, *linear programming (LP)* methods where found to be impractical except for solving small problems of only a few activities. With the increasing capabilities and decreasing costs of computers, researchers have begun to investigate the use of LP procedures in conjunction with other approaches. [Patterson and Huber, 74] combined a minimum bounding procedure with integer LP to reduce the computation time required in arriving at a minimum project duration. Their approach involves starting the optimization procedure off with a "good" lower bound solution, to reduce the domain of possible solutions over which the LP algorithm must search. The "minimum bound" they used to initiate the search procedure is simply an estimate of the minimum project duration implied by the tightest resource constraint, i.e., if the commutative resource requirement of that resource is 80 units, and if 8 units per day are the maximum available, then the minimum project duration = 80/8 = 10 days. Using these combined approach, encouraging results where achieved on small problems, compared to those of other LP approaches.

The LP approaches that have been proposed to date appear to share the common weakness of unpredictability of effectiveness. That is they produce an optimal solution quickly on some problems but not on others. LP approaches still have not progressed to the point where they are capable of routinely solving the type of large, complex problems easily handled by heuristic procedures.

Enumeration Techniques are based on enumeration of all possible activity sequencing combinations (i.e. schedules) and include the so called "branch and bound" procedures. The term "branch and bound" (B&B) refers to a generic type of optimization procedure which involves partitioning a problem into sub-problems (branching) and evaluating these sub-problems (computing bounds). The procedure can be conveniently modeled by the nodes and branches of a tree, to enumerate possible alternatives in arriving at the best solution. This solution tree is sometimes referred to as the "Branch and Bound tree".

A number of branch and bound procedures for the project scheduling problem have been developed by various researchers. One of the most effective to date is that of Stinson, Davis and Khumawala described in [Stinson et al., 78].

All of the enumeration approaches developed to date appear to share the same disadvantage common to LP optimization procedures: unpredictability of computation time from problem to problem.

6.3.2 Heuristic-based Methods

Heuristic-based procedures for resource scheduling are today the only practical means for obtaining workable solutions to large complex problems of the type found in industry. As opposed to the analytical methods, heuristic-based methods claim no "best-solution" for a problem. Rather, they produce "good" feasible solutions.

Heuristic reasoning has been defined by [Polya, 57] as:

> "...reasoning not regarded as final and strict but as provisional and plausible only, whose purpose is to discover the solution of the present problem..."

As indicated earlier, the basic approach to be followed using these methods is to first give priority indices to the various activities and then allocate resources to the activities in order of these rankings. Sometimes, it may happen that two or more activities, competing for the same resource, have the same priority index. A secondary priority index is, therefore,, given to each of the activities.

Hence, the key element is the criterion used to order the activities for scheduling. A change in the order will, of course, change the resulting schedule. Some of the rules/heuristic used for ranking competing activities include [Davis and Patterson, 75]:

1- Minimum Job Slack (MINSLK) which schedules first those activities possessing the least total float. The program would continuously update the float present in an activity, so that any activity sufficiently postponed would eventually become critical.

2- Greatest Resource Demand (GRD) which schedules first those activities requiring the greatest quantity of resources from the outset.

3- Greatest Remaining Resource Demand (GRRD), which schedules first those activities with the greatest amount of work remaining to be completed. This rule characteristicly produces schedules that result in the least amount of resource idle time.

4- Minimum Late Finish Time (LFT) which assigns priorities on the basis of activity LFT (as determined by the initial time-only schedule).

5- Shortest imminent Activity (SIA) which schedules first those activities requiring the least time to complete. It has the property of minimizing the delay beyond the resource free critical path completion time, compared to other rules.

6- Most Jobs Possible (MJP), which schedules first those activities which result in the greatest number of activities being schedule in any interval.

Out of all the different heuristic available, the MINSLK has been found to be superior to others in reducing the total weighted delays in the completion of projects.

The effectiveness of heuristic methods has been studied by [Davis and Patterson, 75] who examined eight different heuristic methods for 83 multi-resource constrained projects. They observed that [Davis and Patterson, 75, p. 952]:

"... none of the heuristic rules tested performed consistently best on all eighty-three problems. However, the MINSLK rule (minimum slack rule), which bases activity priority on activity slack, produced an optimal schedule span most often and exhibited the lowest average increase above optimum of the rules examined..."

Parallel versus Serial Methods

There are two general methods of applying heuristics in project resource scheduling problems. The first, called *parallel scheduling*, makes scheduling decisions after examining all the resource requirements of all activities at a particular time period; while the second method called *serial scheduling* deals with each activity in its entirety before proceeding serially to the next work item. These alternative methods are shown in Figures 6.5 and 6.6 respectively.

In the *parallel scheduling* method shown in Figure 6.5, all activity segments falling within a particular time period are considered together, i.e. in parallel. Within that time period they are ranked as a group in order of priority and each examined and compared against resource availability in the order they are prioritized. When an activity can not be scheduled in a given time period for lack of resources, it is delayed until the next time period. At each successive time period a new rank-ordering of all eligible activities is made and the process continued until all activities have been scheduled. As shown in Figure 6.5, for example, the calculation is at the 6th time interval and the activities will be compared with resource availabilities in the prioritized order 6A, 6C, 6B, 6D.

In the *serial scheduling* method, all activities of the project are ranked in order of priority as a single group, using some heuristic, and then each activity is completely scheduled before considering the next. Thus, activities are said to be scheduled 'serially'. The method is to take an activity from the priority sequenced queue and then to calculate the

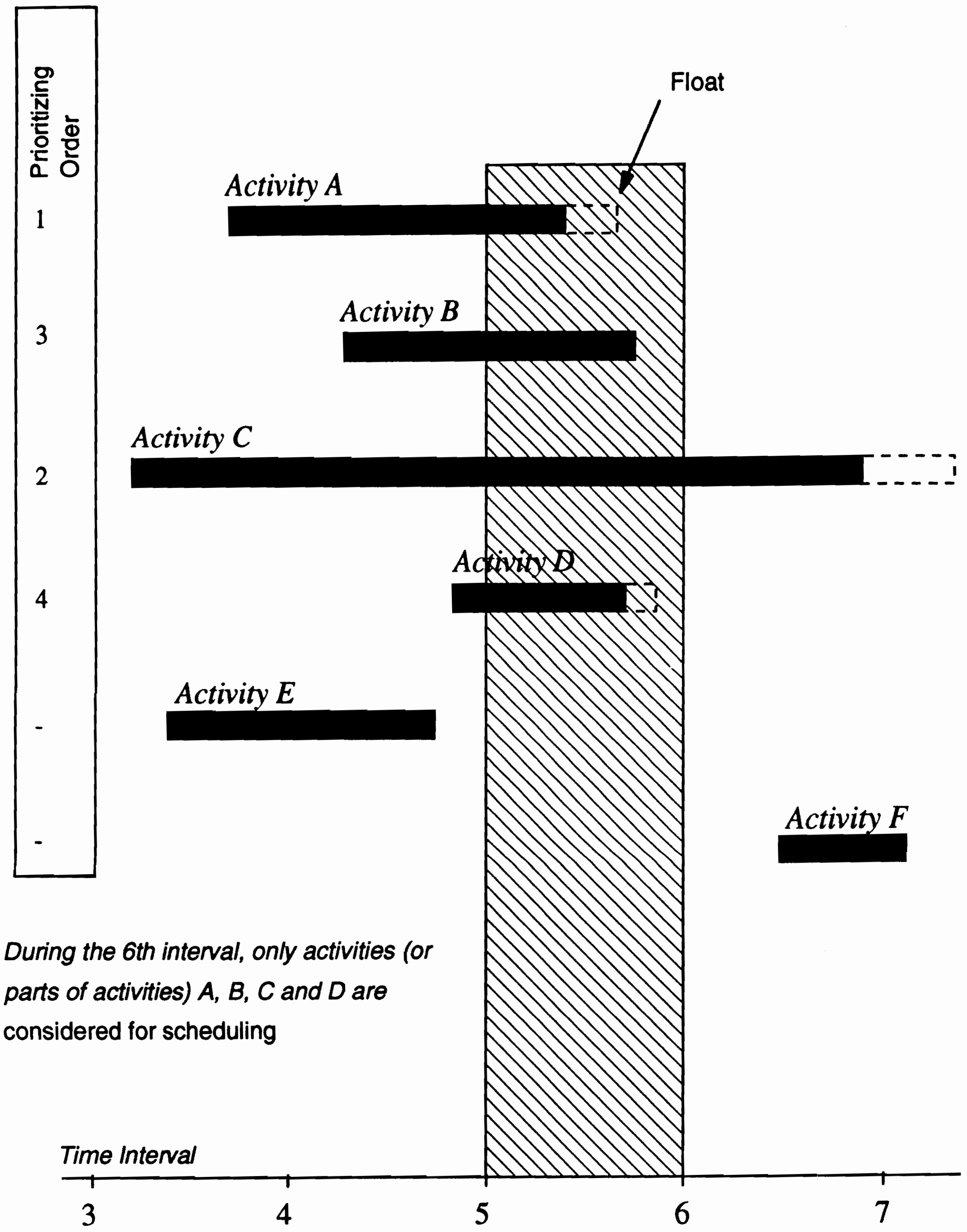

Prioritizing Order
1
3
2
4
-
-
Float
Activity A
Activity B
Activity C
Activity D
Activity E
Activity F
During the 6th interval, only activities (or parts of activities) A, B, C and D are considered for scheduling
Time Interval
3
4
5
6
7

Figure 6.5 - Parallel Scheduling Method

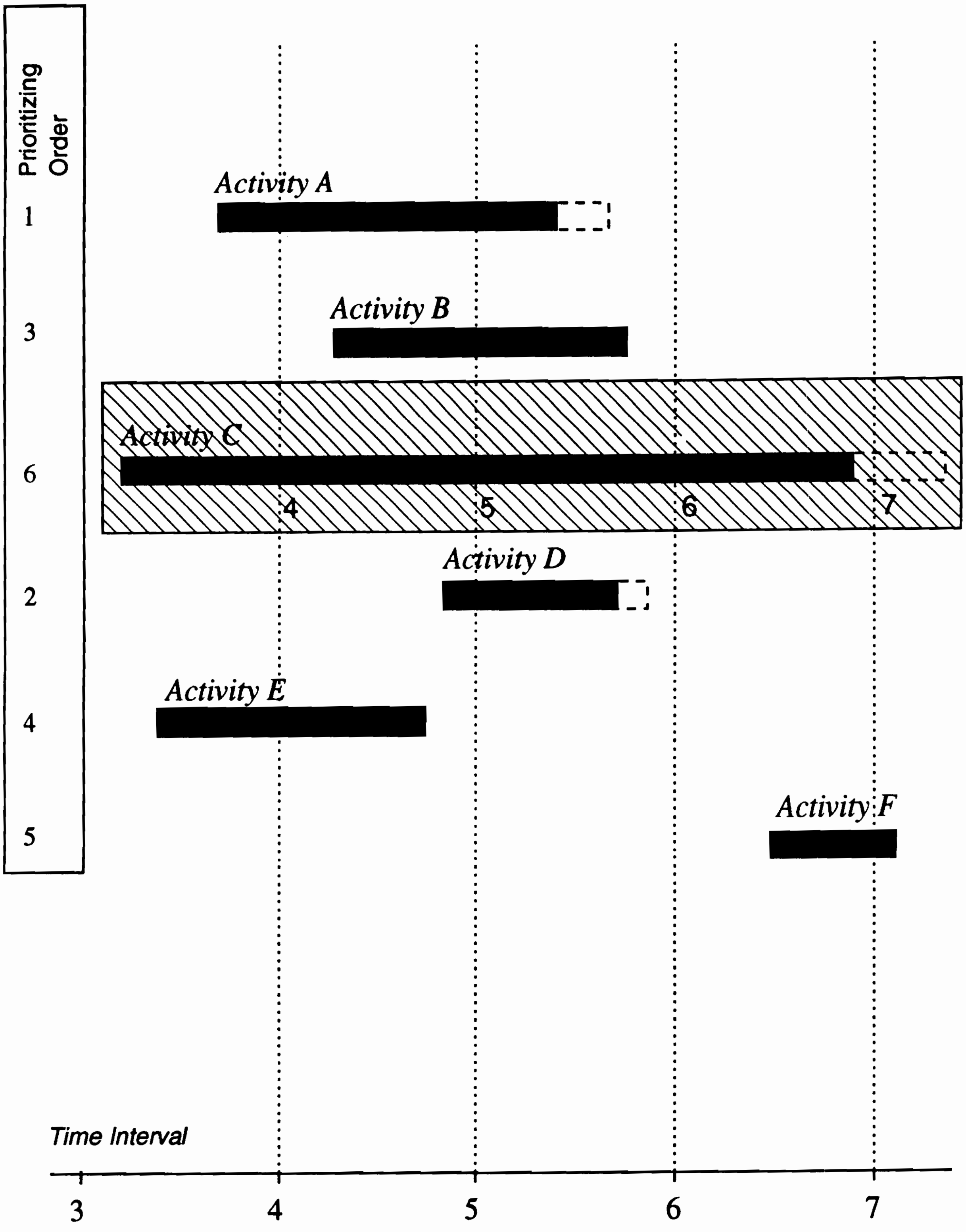


Figure 6.6 - Serial Scheduling Method.

appropriate start and finish dates of the entire activity by examining the resource situation at each time step throughout the activity. Activities that can not be started at their early start time are progressively delayed until sufficient resources are available. This is illustrated in Figure 6.6 where activities are prioritized in the order A, then D, then B, then E, then F, then C. During the scheduling procedure, resource requirements of each activity are examined in time intervals from one end of the activity to the other. In Figure 6.6 the scheduling calculation is shown to be at activity C and the resource requirement will be compared with the resource availability in the order c3, c4, c5, c6, and c7.

Although the parallel method requires more computer time to reorder the eligible activities at each time period or interval, it appears to be the more widely used method. Specific uses are for a number of computer programs for project scheduling [Moder, Phillips and Davis, 83]. A study by [Gordon, 74] comparing the effectiveness of the two approaches indicated that the serial approach could produce shorter-duration schedules for some categories of networks. The study showed that there were also disadvantages with this procedure in terms of the special scheduling conditions (such as activity splitting) which could be handled.

[Woodgate, 77] indicated that one of the advantages of the serial approach is the ability to *look ahead* in order to search for resource availability outside the time period currently being analyzed. Thus, if a shortage of resources occurs at a particular time period, the analysis can proceed along the resource availability list to ascertain whether resources are available at a later date. In the parallel method, the extent of 'look ahead' is limited by the fact that many activities are being analyzed simultaneously.

Some particular types of complex resource problems are best solved by parallel scheduling and others respond more favorably to serial scheduling methods. In general, large 'thin' networks, i.e. those requiring few activities to be scheduled simultaneously, are best handled by the parallel scheduling method whereas short 'fat' networks, i.e. those with more simultaneous working required, are best analyzed by the serial scheduling procedure.

The following sections (6.4 and 6.5) presents example applications of heuristics in resource scheduling.

6.4 Resource Leveling (Smoothing)

The essential idea of resource leveling centers about the rescheduling of activities within the limits of available slack (float) to achieve better distribution of resource usage [Moder, Phillips and Davis, 83]. No constraint is imposed on the availability of resources; but the resource requirements need to be smoothed or evened out. The slack available in each activity is determined from the basic scheduling computations, without consideration of resource requirements or availabilities. Then during the rescheduling, or 'juggling' of activities to smooth resources, the project duration is not allowed to increase.

A method was introduced by [Burgess et al., 62] for leveling resources. This Burgess procedure utilized a simple measure of effectiveness given by the sum of squares of the resource requirements for each "day" (period) in the project schedule. It is easy to show that, while the sum of the daily resource requirements over the project duration is constant for all the complete schedules, the sum of the squares of the daily requirements decreases as the peaks and valleys are leveled. Further, this sum reaches a minimum for a schedule that is level (or as level as can be obtained) for the project in question.

6.5 Limited Resource Allocation

These types of scheduling techniques produce schedules that will not require more resources than are available in any given period, with project durations increased beyond the original critical path length as little as possible.

Initially, the attempt has to be to lower the peak requirements of the resources by staggering the resource input on non-critical activities. However, it may not always be possible to keep the total resource demands within specified (or desired) limits by re-scheduling of the non-critical activities alone. So, it may become necessary to tackle the critical activities and to withdraw the resources from these so as to bring peak demands below the specified levels. This may mean delay in completion of the work; but this may be inevitable for 'resource constrained' situations.

As a result, either the duration of the critical activities has to be increased; or it may become necessary to place some of the concurrent activities in series, so as to reduce the peak demands of the scarce resources. This would give rise to an increase in the project durations.

An example of this type of scheduling procedure is shown in the flow chart in Figure 6.7 (adopted from [Moder, Phillips, and Davis, 83]). The procedure involves a parallel approach. First a set of activities is defined whose predecessors are all scheduled. This set is called the Eligible Activity Set *(EAS)*. Then, only the activities in the *EAS* with their early start (ES) less than or equal to some time value T are considered. These activities are ordered with the least slack first and, with this criterion, with the shortest duration first. This ordered list of activities is referred to as the Ordered Scheduling Set *(OSS)*.

Resources are allocated to each activity in *OSS* in the order specified. Once an activity is scheduled, it is dropped from the list. The procedure is continued until all the activities are scheduled.

6.6 Chapter summary

Traditional network techniques, such as PERT and CPM, tend to formulate the construction logic and schedule independent of project resource requirements. This

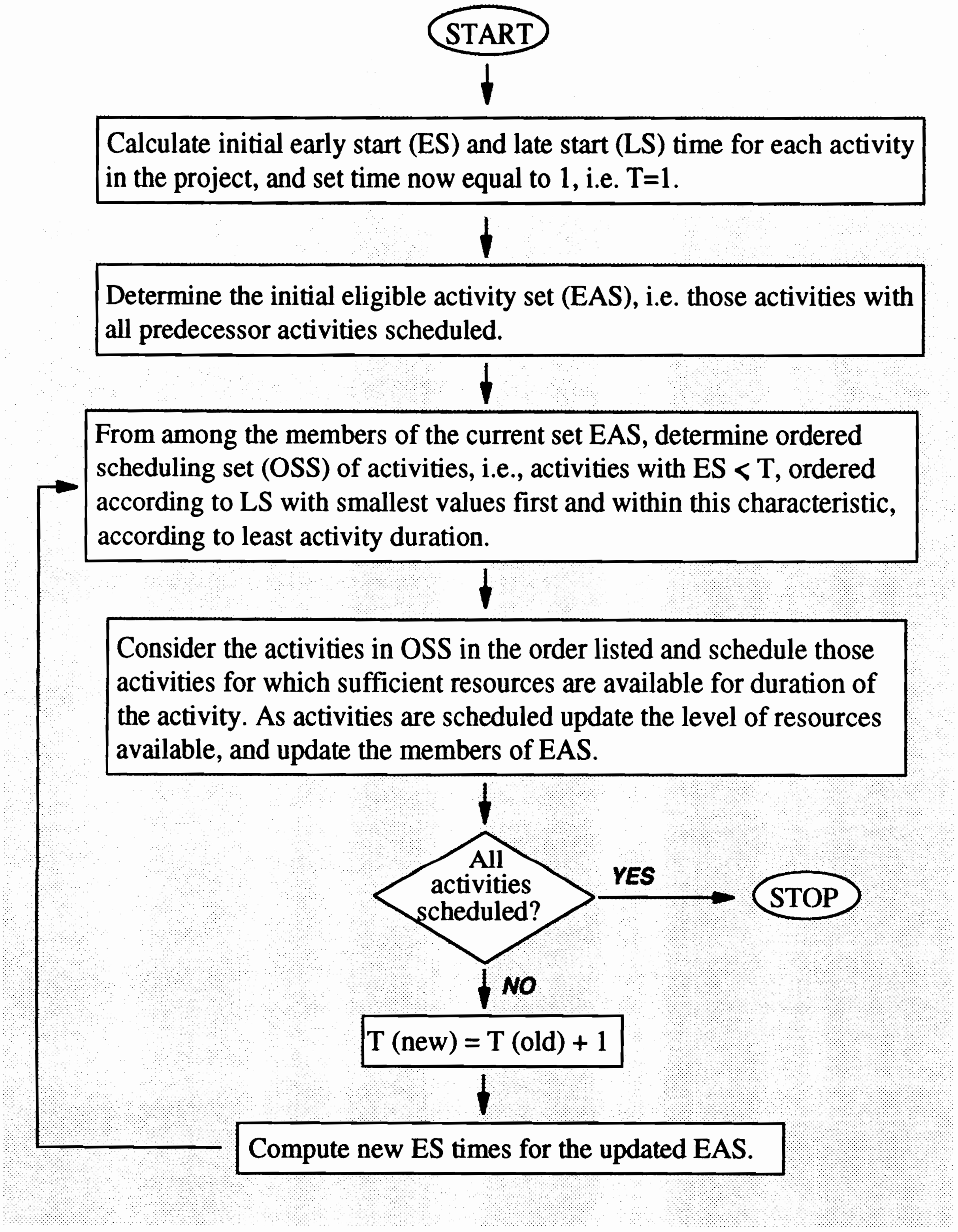

START
Calculate initial early start (ES) and late start (LS) time for each activity in the project, and set time now equal to 1, i.e. T=1.
Determine the initial eligible activity set (EAS), i.e. those activities with all predecessor activities scheduled.
From among the members of the current set EAS, determine ordered scheduling set (OSS) of activities, i.e., activities with ES < T, ordered according to LS with smallest values first and within this characteristic, according to least activity duration.
Consider the activities in OSS in the order listed and schedule those activities for which sufficient resources are available for duration of the activity. As activities are scheduled update the level of resources available, and update the members of EAS.
All activities scheduled?
YES
STOP
NO
T (new) = T (old) + 1
Compute new ES times for the updated EAS.

Figure 6.7 - Example for a Limited Resource Allocation Procedure.

adopted from [Moder, Phillips, and Davis, 83] tendency, coupled with the initial assignment of required resources (implied or otherwise) to each individual activity, gives little scope for resource considerations during the formative stage of the construction plan and schedule. As a result, these techniques have been described as being based on the assumption of infinite resource availability.

Two classical resource problems exist, the resource leveling problem and the fixed resource limits scheduling problem. The *resource leveling problem* adopts the network model and project duration based on the assumption of infinite resources, and searches for the schedule of the project activities that removes peak requirements of resources with no increase in the original project duration.

The *fixed resource limits scheduling* problem commences with a network model and a project duration based on the assumption of infinite resources. The objective is to produce a practical schedule of the activities with minimum time extensions and based on resource requirements that can be met from a defined resource availability while maintaining the construction logic.

Currently, no formal mathematical model exists that considers the interaction between the various possible resource allocations per activity, the construction logic, and defined acceptable resource availabilities. Many heuristic models exist based on arbitrary criteria which obtain feasible solutions by considering a limited view of the problem area and taking advantage of a particular structuring of the limited resource problem.

7.0 WORK SPACE SCHEDULING CONCEPTS FOR MULTI-STORY BUILDINGS

Multi-story building construction falls under a category of projects, termed linear or repetitive projects, that involve many repetitive activities performed consecutively by the same crew. Current available network techniques and linear scheduling methods develop the sequence of activities in such projects by considering technological relationships among the activities and their requirements for labor, material and equipment. Using such techniques, precedence logic among the different activities is first determined based on the technological requirements defined between these activities (hard logic). This logic is further enhanced by recognizing the conflict between the resources that can be made available to a project and those apparently needed as determined from the network model (soft logic).

Although these scheduling routines are widely used in the scheduling process of multi-story buildings, they ignore requirements of activities for work area or space necessary for material storage and movement of manpower and equipment. Any task or activity requires a specific work space for its execution. This space demand is based on the space requirements of each resource allocated to the activity. When such required space becomes limited or unavailable, the activity or task can not be executed or, in some cases, is performed with a lower productivity rate. This is because performance and maneuvering of either crew or equipment may become difficult and sometimes not possible.

In multi-story building projects, limited availability of work space may be a result of many factors. Due to the nature of multi-story buildings where construction is mostly in crowded downtown locations, storage of construction materials on site premises is not always possible because the available area is usually limited. To remedy this problem, construction floors are used for storage of materials required by many activities. The space consumed by material stored in the work areas affects the overall availability of space for activities executed in these areas.

Another factor that contribute to the limited work space problem in the floor is that some activities require a large amount of work space during execution. For example, in mechanical duct installation, materials are first spread and assembled partially on the floor area before they are installed in place. As a result, available work space for other activities sharing the same work area is considerably reduced.

A third factor that adds to the limited work space availability problem is the work area allocation policy among crews of different trades working on the floor. Based on such policy, work areas are solely allocated to one trade at a time. Although space may be available to accommodate crews of other trades concurrently, these crews may not be allowed to occupy or store their equipment and materials in such space. This policy among the different trades is implicitly understood by the crews and, in some cases, explicitly stated by the main contractor or construction manager. As a result, work space is considered theoretically unavailable for concurrent activities until the activity occupying such space is entirely completed.

All these factors contribute towards the limited work space availability problem in this type of construction. Because the current scheduling techniques do not account for work space as a constraint, floor activities that are logically independent and assigned to the same work area are scheduled concurrently although work space may not be sufficient or available to accommodate these activities. The problem is sometimes remedied by the construction staff that manually checks the areas that might become congested and attempts to maintain a specific area per resource for the construction operation. However, such manual procedures are not always applied to all work areas during the initial schedule and are not repeated as the schedule is updated.

Meeting the space needs of resources in the schedule for multi-story construction thus requires a new scheduling model that incorporates work space as a constraint in the schedule generation process. The model should take into consideration the space demand of all resources involved in the project versus the available space for these resources. In addition, the model should allow to integrate work space constraints with other scheduling constraints (e.g. resource constraints) to insure that all constraints can be considered in the generation of the schedule.

This chapter presents a proposed scheduling model that considers work space constraints in scheduling construction activities for repetitive work in multi-story buildings. The chapter first describes a procedure to define and evaluate work space parameters; *work space demand* and *work space availability*. The chapter then describes space-based scheduling decisions that utilizes these space parameters to generate a space-based schedule.

The chapter also provides a worked example to demonstrate the theory of the proposed space-based model in scheduling the repetitive floors. The example generates in a step-by-step procedure a space-based schedule for a 10 story building project.

7.1 Work Space Concept

Work space can be viewed as a special type of resource in the sense that the availability of space is a necessary requirement to perform any set of activities. If space becomes limited, then the scheduled activities that require this space are affected. To implement work space as a constraint in the generation of the schedule, it must first be defined and evaluated. Two parameters define work space constrains: *work space demand* and *work space availability*.

Work Space Demand: This parameter defines the space necessary to accommodate any activity in a the work area. This demand is equal to the total amount of space required by all resources allocated to the activity,

$$\text{Space Demand } \textit{(Activity)} = \sum \text{Space Demand } \textit{(Allocated Resource)} \text{ ----(1)}$$

Space required by any resource can be measured from 3D computer models that represent these resources or can be estimated from site observations and historical data. Knowing the resource requirement data for any one activity, total space demand for the activity can be quantified.

Work Space Availability: The second parameter of work space is work space availability. This parameter defines the amount of space available for any activity during the time period the activity is considered for scheduling. This is dependent on the total available space of the work area and the work usage consumed by other activities concurrent to the activity.

$$\text{Space Availability } \textit{(Activity)} = \begin{array}{l} \text{Total Available Space } \textit{(Work Area)} \\ \text{Space Usage } \textit{(Concurrent Activities)}\text{----(2)} \end{array}$$

7.2 Modeling Work Space Demand

In order to quantify work space demand for an activity, total space demand of all resources necessary to perform the activity must be evaluated. Space demand for any resource is dependent on many factors such as: dimensions of the resource, physical characteristics of the resource (e.g. shape), quantity of the resource, physical characteristics of work area (e.g. floor height), possible storage arrangement in the work area (materials only), and safety issues. Taking these factors into consideration, space demand for any resource can be viewed as a combination of the actual physical three-dimensional space the object resource occupies and an additional space that the resource occupies.

Smith [Smith, 87] proposed a model to define the space occupied by any three dimensional object. Smith suggested that the space demand or usage of any object is a combination of the physical three-dimensional space the body occupies, the space immediately adjacent to the object, and the additional space that is momentarily occupied by moving objects or that remains idle. This is expressed as follows:

$$\mathbf{Space_{(object)} = Space_{(fixed)} + Space_{(imd)} + Space_{(shared)}} \qquad \text{----(3)}$$

where,

$\text{Space}_{(\text{object})}$= total space required by any object (e.g. worker, bags of cement, scaffolding, etc.)

$\text{Space}_{(\text{fixed})}$= is the physical space the body or object occupies. The distance between the outer vertices of the body/object may be used to calculate the volume of space occupied.

$\text{Space}_{(\text{imd})}$= is the space immediately adjacent to $\text{Space}_{(\text{fixed})}$ and is considered as a "safe margin". It is almost never accessed by another body or object except on occasions of very close encounter.

$\text{Space}_{(\text{shared})}$= is the space peripheral to $\text{Space}_{(\text{imd})}$ within the work inclosure in question. It is momentarily occupied by moving objects or remains idle as "float space".

The model proposed in this dissertation defines work space for any resource in terms of two values; a physical value (S_{physical}) , and a surrounding value ($S_{\text{surrounding}}$). Space demand of any resource is thus expressed as follows,

$$\text{Space Demand } \textit{(Resource)} = S_{\text{physical}} + S_{\text{surrounding}} \qquad \text{----(4)}$$

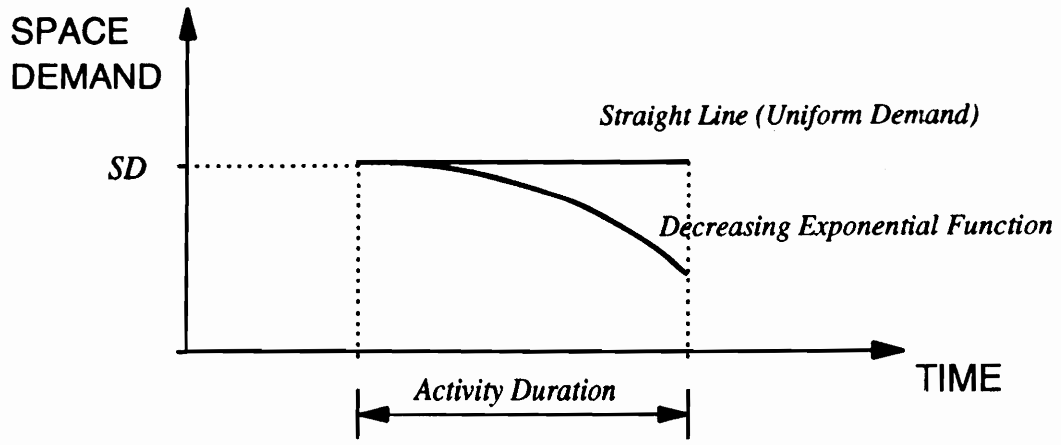
The first value of space demand, $S_{physical}$, denotes the actual three-dimensional space the body of the resource occupies in the work area inclosure. The distance between the outer vertices of the resource may be used to calculate the volume of space occupied. The second value, $S_{surrounding}$, is the space adjacent to the physical space of the body of the resource and is almost never accessed by another resource. The surrounding space is considered as a safety zone and is necessary for the movement of the resource (manpower and equipment only). This space can also be considered as a dead work area that may not be possible to utilize by other resources (e.g. space trapped between stored material and ceiling).

Because of the nature of construction, the significance of these space values will vary from one type of resource to another. For example, for resource 'manpower', although $S_{physical}$ is proportionally small in comparison to the large work space associated with the work floor area, the value of $S_{surrounding}$ plays a more significant role in determining the total amount of space demand required by this type of resource. Due to safety reasons, the value of $S_{surrounding}$ is always bigger in proportion to $S_{physical}$. For resource 'equipment', however, required space is determined by both values; $S_{physical}$, and $S_{surrounding}$ because of volume and mobility characteristics associated with this type of resource. On the other hand, space demand for resource 'material' will mostly be quantified based on the $S_{physical}$ value. This is because of the large physical space occupied by this type of resource when stored within the construction work area in the floor.

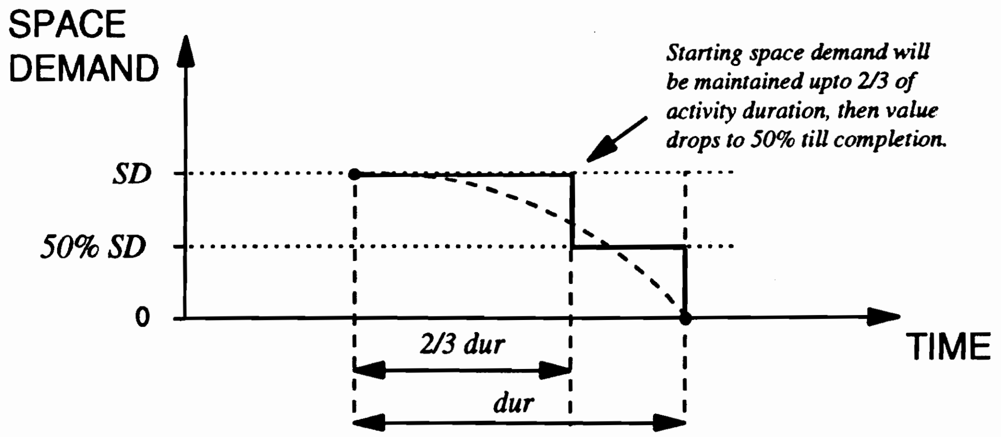
Values of $S_{physical}$ for resources can be measured from 3D computer models that represent these resources or can be estimated or quantified based on physical attributes. Values of $S_{surrounding}$ is estimated based on site observations and historical data.

Level of Space Demand

The level of space demand for an activity is not necessarily constant along its entire duration. One factor affecting the space demand level is the type of resources allocated to the activity. Space demand for resources 'manpower' and 'equipment' could be considered uniform over the entire activity duration and is represented by a constant straight line depicted in Figure 7.1(a). This is because the same amount of space is required as the activity progresses in time. On the other hand, space demand required for


SPACE DEMAND
Straight Line (Uniform Demand)
SD
Decreasing Exponential Function
TIME
Activity Duration

(a) Uniform and Decreasing Level of Space Demand


SPACE DEMAND
Starting space demand will be maintained upto 2/3 of activity duration, then value drops to 50% till completion.
SD
50% SD
0
TIME
2/3 dur
dur

(b) Proposed Step Function

Figure 7.1 - Level of Space Demand

material storage in the work area decreases as time increases. As the activity progresses, material stored in the floor is converted into work in place freeing storage space for other activities. This decrease in space required can be theoretically represented by a decreasing exponential function depicted by the hypothetical curve also shown in Figure 7.1(a).

Another issue affecting the level of space demand is the space allocation policy among the crews of the different trades working on the floor. Because of such policy the total available space of the work area is allocated entirely to a specific activity. Space becomes practically available to other concurrent activities only when the activity is completed. In this case, the level of space demand of all resources is considered constant for the entire duration of the activity.

In order to take these factors into account, activities are classified into 3 categories or classes of space demand: Class *A*, Class *B* and Class *C*.

CLASS A: This class includes those activities whose crews are solely assigned to the work area as a result of the work area allocation policy among different trades. As each activity of this class is scheduled, the total available space of the work area in the floor is allocated to that activity. Other activities that are technically independent and may be executed concurrently with the activity of this class are only scheduled to start after the activity is completed. The space demand level for this class of activities is considered constant for the entire duration of the activity. Activities of this class include any construction activity to which the entire work space is allocated and are specific to the project and management. Example activities are such as: False ceiling works, raised floor works, etc.

CLASS B: Belonging to this class are those activities that accumulate their space demand mainly from the requirement of their manpower and equipment for work space. Such activities only require a small amount of space for storage of their construction material on the floor or do not utilize the floor for material storage.

Similar to class A, activities of this class will also have a constant level of space demand for the entire duration of the activity. However, work space will be allocated to this class only by the amount required. The allocated space will be deducted from the total available space of the work area in the floor in order to determine the remaining space for other competing activities. Concurrent activities may share the work area only if the remaining space is enough to accommodate these activities.

Because the space demand level of this class of activities is considered constant, the occupied space becomes available for other activities only when the activity is completed. Example activities of this class may include: electrical wiring, painting works, accessories works, etc. Such activities do not utilize the floor area for their material storage, or require a small amount of storage space on the floor.

CLASS C: Activities of this class constitute those activities that require a large space area for storage of their construction material in the work area. As work progresses, stored material is converted into work in place and space becomes available for other activities. Space demand for this class will decrease as time increases.

It should be noted, however, that the duration over which the material is converted into work in place in relation to the activity overall duration may vary between activities of this class. That is, for some activities stored material might be converted into work in place from day one of the activity start date, and can continue for the entire duration of the activity. For other activities, their may be an idle time lag at the start of the activity or before the activity is completed where material is not used. This may be attributed to necessary time for setup and preparation works or finishing and cleanup works associated with the execution of these activities.

A simple step function is proposed to model the decrease in space demand value for this class of activities as illustrated in Figure 7.1(b). Based on the proposed function, 50% of space allocated to the activity material storage is assumed to become available after 2/3 of activity duration has elapsed. This freed space will become available for any activity requiring it.

Example activities of this class may include: blockwork, door frames installation works, drywall partitioning works, raised floor works, plumbing works, HVAC equipment and duct installation works, etc.

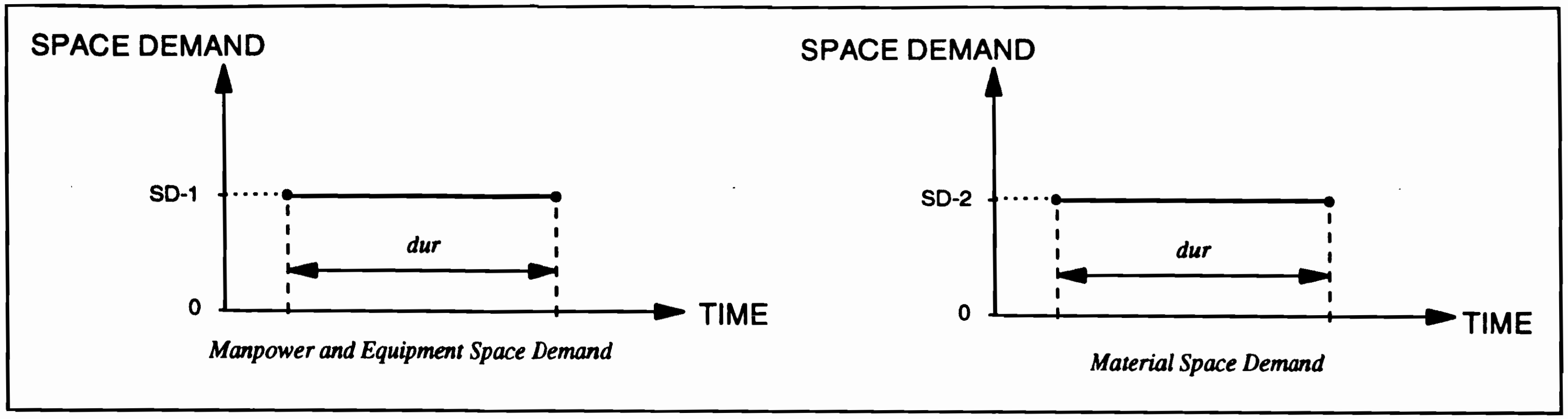
Space Demand Types

Space demand for each activity is divided into two groups or types: manpower and equipment space demand; and material space demand. The distinction allows to monitor each space demand type separately. This is significant because:

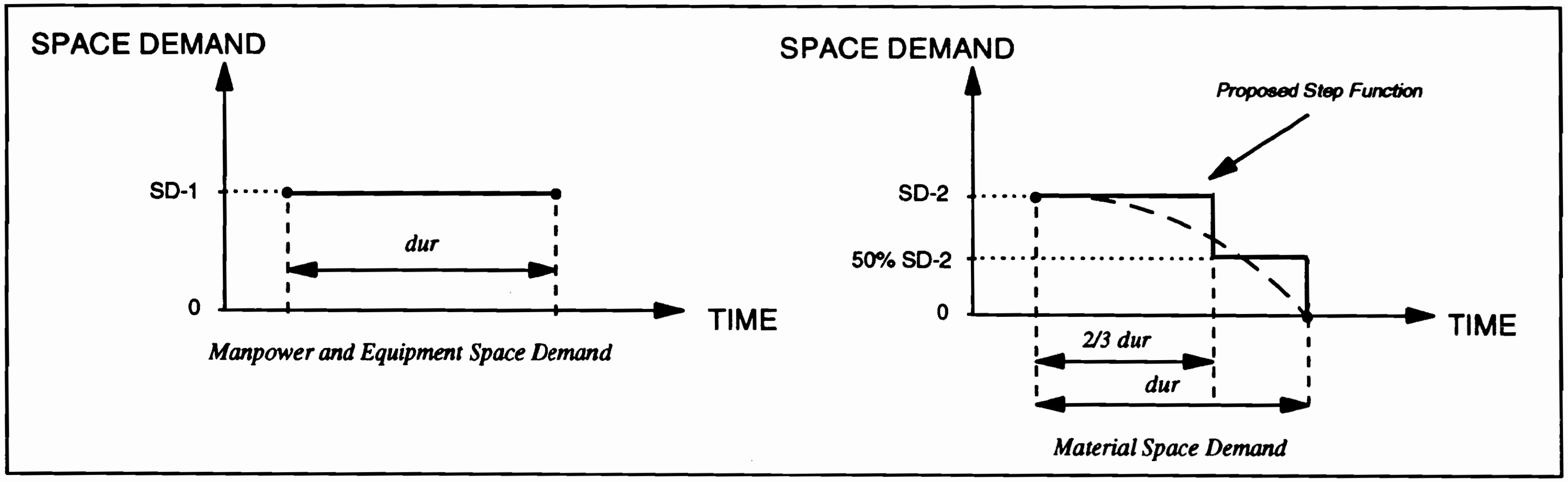
a) In many cases, other activities competing for space occupied by manpower and equipment of an activity are not necessarily the same as those competing for space occupied by its stored material. This is because the area where material is stored on the floor is, in many cases, different from that where manpower and equipment operate.

b) For activities of class C, space demand for manpower and equipment remains constant over the activity duration and does not become available for other activities until the activity is completed. On the other hand, space demand required for material storage decreases gradually as materials are turned into work in place and freed space becomes available to other activities needing it.

Space demand for resource manpower and equipment is denoted SD-1, while space demand for resource material is denoted SD-2. As illustrated in Figure 7.2, space demand value for manpower and equipment (SD-1) is always uniform for activities of any class



(a) Activities of Class A and B



(b) Activities of Class C

Figure 7.2 - Space Demand Requirements for Different Activity Classes

(A, B, or C). However, space demand value for material (SD-2) is uniform for activities of class A, or B only. For class C activities, the value of SD-2 decreases by the proposed decreasing step function. Based on this function, 50% of SD-2 becomes available for other activities after 2/3 of activity duration is elapsed.

Evaluation of Space Demand

Once $S_{physical}$ and $S_{surrounding}$ for all resources are estimated or computed, space demand for any activity is determined. Knowing the required resource pool for any activity, space demand for the activity is be evaluated as follows:

$$\text{Space Demand}_{\textit{Manpower and Equipment}}\ (\text{SD-1}) = \Sigma\ \text{Quantity x } (S_{physical} + S_{surrounding}) \qquad \text{----(5a)}$$

$$\text{Space Demand}_{\textit{Equipment}}\ (\text{SD-2}) = [\ \Sigma\ (\text{Quantity x } S_{physical})\ ] + S_{surrounding} \qquad \text{----(5b)}$$

Equation (5a) evaluates the space demand of manpower and equipment for any activity by multiplying the total space demand for each unit resource ($S_{physical}$ and $S_{surrounding}$) by the quantity of the resource allocated to the activity. This is because the surrounding space demand value for manpower and equipment is associated with each unit resource. On the other hand, because of the stacking characteristics of material during storage in the work area, the surrounding space demand value of the resource is associated with the overall quantity instead of the unit resource. As a result, it is more practical to first determine the three-dimensional space (i.e. $S_{physical}$) consumed by all material quantity stored, and then estimate the surrounding space demand value. This is portrayed by equation (5b).

It should also be noted that quantity of resources determined for each activity during data input procedures are usually associated with normal productivity rates of the activity. However, scheduling techniques utilized for scheduling repetitive work for this type of construction allow for procedures to sometimes modify the normal productivity rate for each activity in order to balance it with productivity rate values of preceding activities. Such procedures are practically adopted in linear scheduling methods such as the Vertical Production Method (VPM) [O'Brien, 75]. As a result, defined quantities of manpower and equipment allocated to each activity should be modified to account for any new productivity rates selected by the scheduling procedure. Consequently, the value of SD-1 for any activity can not be evaluated during the data input stage and can only be determined during the scheduling process when the appropriate productivity rate for each activity is selected.

On the other hand, quantity values of material for any activity are independent of the productivity rate. Rather, material quantities are based on the amount of work put in place. Hence the value of SD-2 for any activity can be determined at the data input stage.

Figure 7.3 depicts the proposed procedure for defining space demand for any activity during the data input stage and the scheduling stage. At the data input stage, resource space demand values for unit resources are defined by the user. Unit values for material are utilized to evaluate total space demand (SD-2) using equation 5(b). For manpower and equipment, demand values for unit resources are input to the scheduling stage. In the

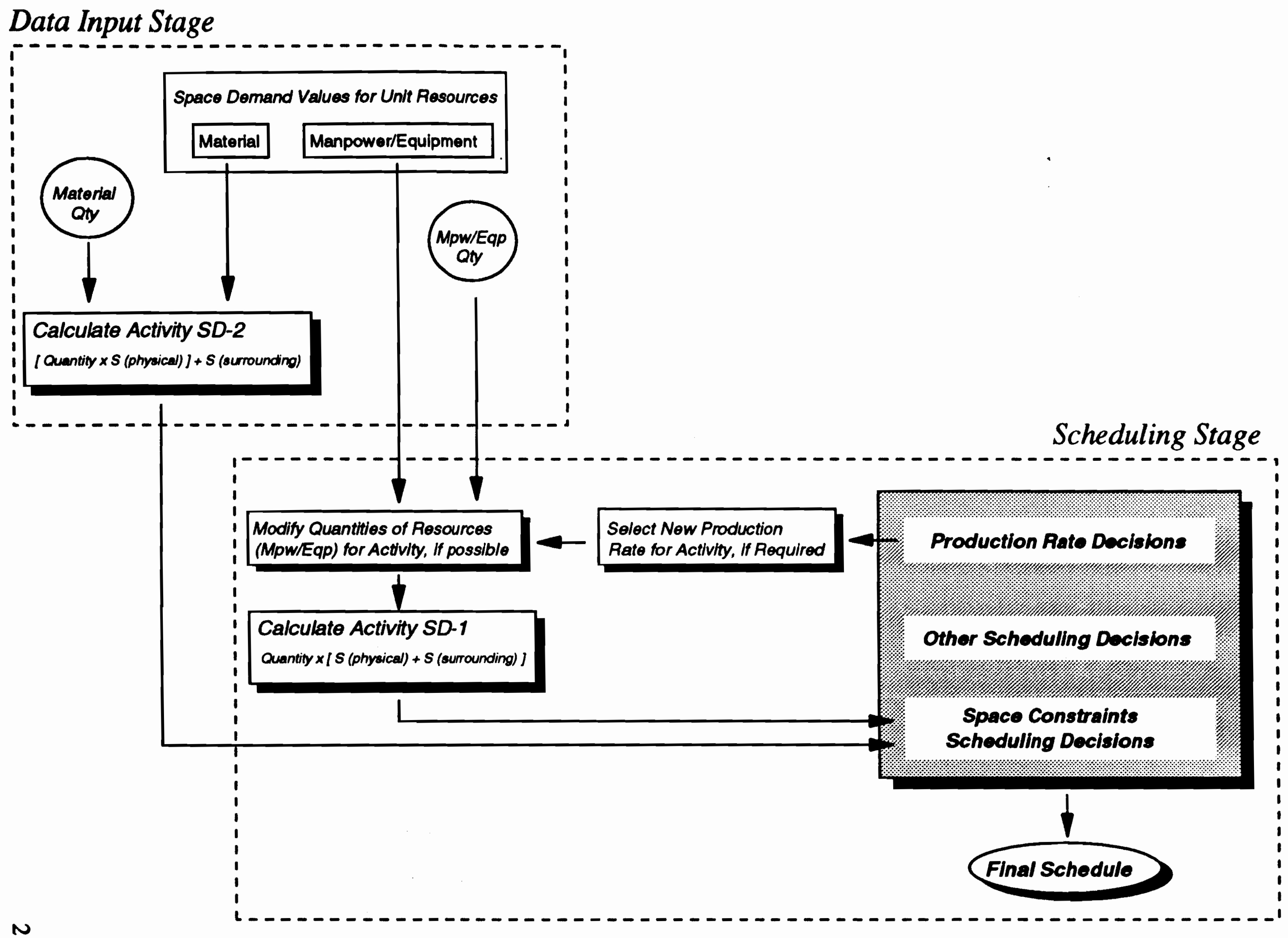

Data Input Stage
Space Demand Values for Unit Resources
Material
Manpower/Equipment
Material Qty
Mpw/Eqp Qty
Calculate Activity SD-2
[Quantity x S (physical)] + S (surrounding)
Scheduling Stage
Modify Quantities of Resources (Mpw/Eqp) for Activity, If possible
Select New Production Rate for Activity, If Required
Production Rate Decisions
Other Scheduling Decisions
Calculate Activity SD-1
Quantity x [S (physical) + S (surrounding)]
Space Constraints Scheduling Decisions
Final Schedule

Figure 7.3 - Procedure of Defining Activity Space Demand

scheduling stage, once quantities of manpower and equipment are determined, total space demand for manpower/equipment (SD-1) is determined using equation 5(a).

Figure 7.4 illustrates the database setup for evaluation of the space demand at the data input stage. To define space demand at this stage, three database files are utilized. These files are: a ***Global Space Demand Data File***, a ***Activity Resource Requirement Data File***, and an ***Resource Space Demand Data File***.

The *Activity Resource Requirement Data File* is a project specific file that contains the resource pool required by each activity of the project as defined by the user based on normal production rates. The *Global Space Demand Data File* is a global database file that will contain records of space requirements for different generic construction resources. Individual values of $S_{physical}$, and $S_{surrounding}$ for each resource will be based on values obtained from previous projects. The objective of these values stored in this global file is to assist the user to make an estimate of $S_{physical}$ and $S_{surrounding}$ for the project under consideration. The *Resource Space Demand data file* will contain space demand values for unit resources extracted from the global file, space demand values estimated by the user and a total space demand value for each resource. The total value will be based on the unit resource for resource 'manpower and equipment' and total value (i.e. S-2) for resource manpower.

As illustrated in Figure 7.4, the sequence of operations in the database environment to

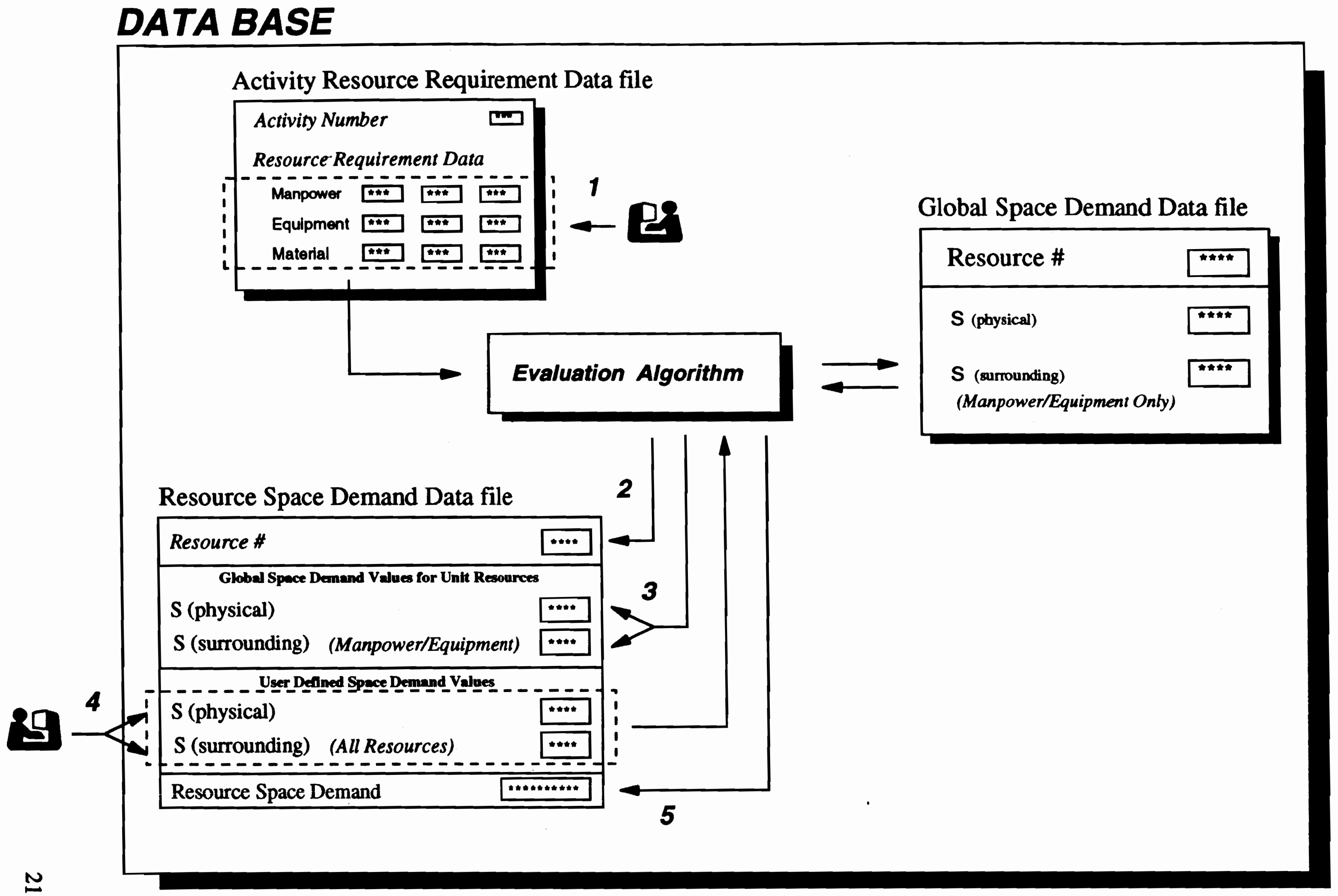

DATA BASE
Activity Resource Requirement Data file
Activity Number
Resource Requirement Data
Manpower
Equipment
Material
1
Evaluation Algorithm
Global Space Demand Data file
Resource #
S (physical)
S (surrounding)
(Manpower/Equipment Only)
Resource Space Demand Data file
2
Resource #
Global Space Demand Values for Unit Resources
S (physical)
S (surrounding) (Manpower/Equipment)
3
User Defined Space Demand Values
S (physical)
S (surrounding) (All Resources)
4
Resource Space Demand
5

Figure 7.4 - Evaluation of Work Space Demand at The Data Input Stage

extract space demand for each resource consists of the following steps:

Step-1 The user will first input different resource types and quantities for each project activity in the *Activity Resource Requirement Data file*. Quantity values will be based on normal production rates (i.e. normal activity duration).

Step-2 Given the required resource types associated with each activity defined in the *Activity Resource Requirement data file*, an evaluation algorithm generates records of resource types and store it in the *Resource Availability data file*.

Step-3 Using the global space demand values for the generic resources stored in the *Global Space Demand Data file*, the evaluation algorithm extracts the space demand values for each resource and store it in the appropriate record in the *Resource Space Demand Data file*.

Step-4 Based on the extracted global values for unit resources, the user will define demand values based on the conditions of the project under consideration. These values are input in the *Resource Space Demand Data file*. As indicated earlier, the user defined values for $S_{physical}$ and $S_{surrounding}$ will be based on unit quantity for manpower and equipment and total quantities for material.

Step-5 Once an estimate is defined by the user for $S_{physical}$ and $S_{surrounding}$, the algorithm calculates the total demand value. This value will be extracted by the scheduling system.

This procedure is discussed in more detail in Chapter 9.

7.3 Modeling Work Space Availability

In order to determine work space available to schedule any activity during a specific time period, it is necessary to first evaluate the total work space of the work area in which the activity is allocated. Knowing the work space consumed by other activities already scheduled in that work area within that time period, available space for the activity is determined using equation 2 as follows,

$$\text{Space Availability } \textit{(Activity)} = \text{Total Available Space } \textit{(Work Area)} - \text{Space Usage } \textit{(Concurrent Activities)} \quad \text{----(2)}$$

Total available space of the work area is dependent on its physical characteristics. Activities sharing the same work area will compete for the total work space in that area only. Activities in other work areas will compete for different work space values associated with these areas. As a result, total space availability varies for different activities and is dependent on the work location and the time period at which these activities will be executed.

For example, early activities such as concrete works and interior blockwork have the entire floor work space available for their execution. Once these activities are completed, the floor becomes segmented into smaller work areas. Succeeding activities such as mechanical piping and electrical works become confined to new work areas with a new lower value of space availability. As construction progresses, the construction work area on the floor continues to change and available space continues to decrease. This is because work areas are further segmented into more work areas with less available space, and space of existing work areas is decreased as a result of work put in place. Consequently, activities executed in a given work area will have a different value for available work space than other activities assigned to other work areas.

In order to determine space availability for any activity, the procedure groups activities sharing the same work area and competing for the total work space of that area, the construction floor is first divided into a number of work blocks as shown in Figure 7.5. A block is composed of a zone and a layer. Each zone will represent a separate location in the typical floor (e.g. west-side). Each layer will represent the status of construction work progress in that zone within a specific time period (e.g. bathroom area in west-side zone between the completion of blockwork and the start of cabinets and fixtures installations). Each block will have a specific available work space value.

Once work blocks are created, construction activities are allocated to one or more block as indicated in Figure 7.5. This grouping of activities under different work blocks allows monitoring of available space amongst activities sharing the same work location during a specific time period (i.e. activities allocated to the same block). Activities executed within the boundaries of a zone and layer block will only be allocated to that block. Activities belonging to the same work block will share the same amount of total available work space of that block. However, if the activity is sharing the space of more than one block (e.g as in the case where materials are stored in one block of the floor while the actual physical work is executed in another), then the activity is allocated to all corresponding blocks.

Work space for each block is quantified utilizing a CAD model of the typical floor work area. Geometric data (x, y, and z coordinates) of the design elements provided in the

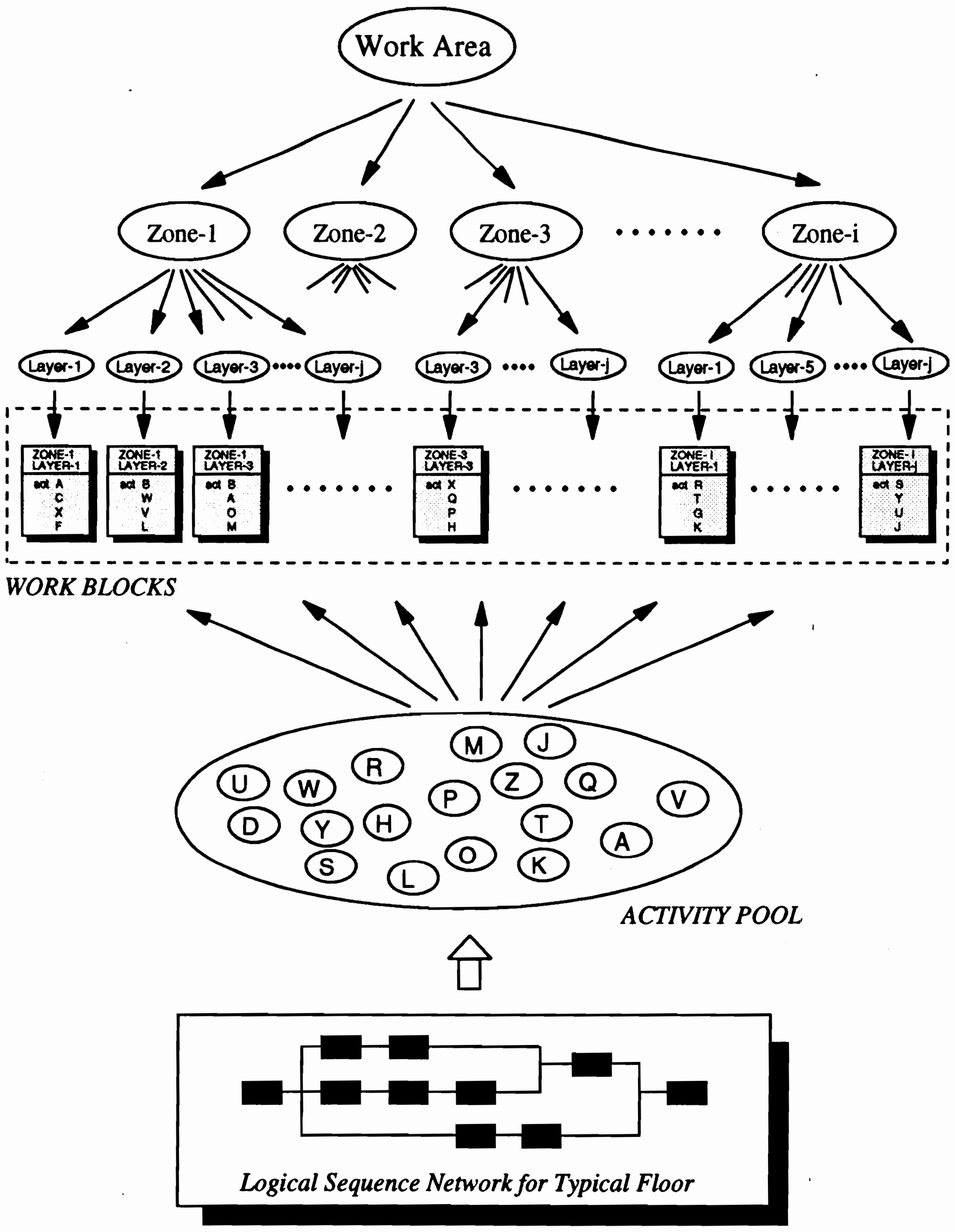

Work Area
Zone-1
Zone-2
Zone-3
Zone-i
Layer-1
Layer-2
Layer-3
Layer-j
Layer-3
Layer-j
Layer-1
Layer-5
Layer-j
ZONE-1 LAYER-1 act A C X F
ZONE-1 LAYER-2 act B W V L
ZONE-1 LAYER-3 act B A O M
ZONE-3 LAYER-3 act X Q P H
ZONE-I LAYER-1 act R T G K
ZONE-I LAYER-j act S Y U J
WORK BLOCKS
M J R U W Z Q P V D Y H T A S O K L
ACTIVITY POOL
Logical Sequence Network for Typical Floor

Figure 7.5 - Allocation of Activities to Block Structure

CAD model is used to calculate the amount of available work space of each block.

Evaluation of Work Block Space Availability

Figure 7.6 depicts the database setup to define the work block structure and evaluate total available space for different work areas. A step-by step to accomplish this process is as follows:

1- Create Work Blocks

The floor is divided into a number of zone and layer blocks. Each activity in the typical floor will correspond to one or more work block. The objective is to group construction activities competing for the same total work space within a specific time period.

To create zones and layers, the typical floor is first divided into different zones. Each zone will represent a separate construction location on the typical floor. The number and area covered by each zone may be based on the architectural layout of the floor. Example zones may include: North-side and South-side, or Zone-1, Zone-2, Zone-3, etc.

Once zones are established, each zone is then divided into one or more different layers. Layers can be viewed as sub-zones. Each sub-zone is created as a result of the execution of a group of activities during a specific time frame resulting in a

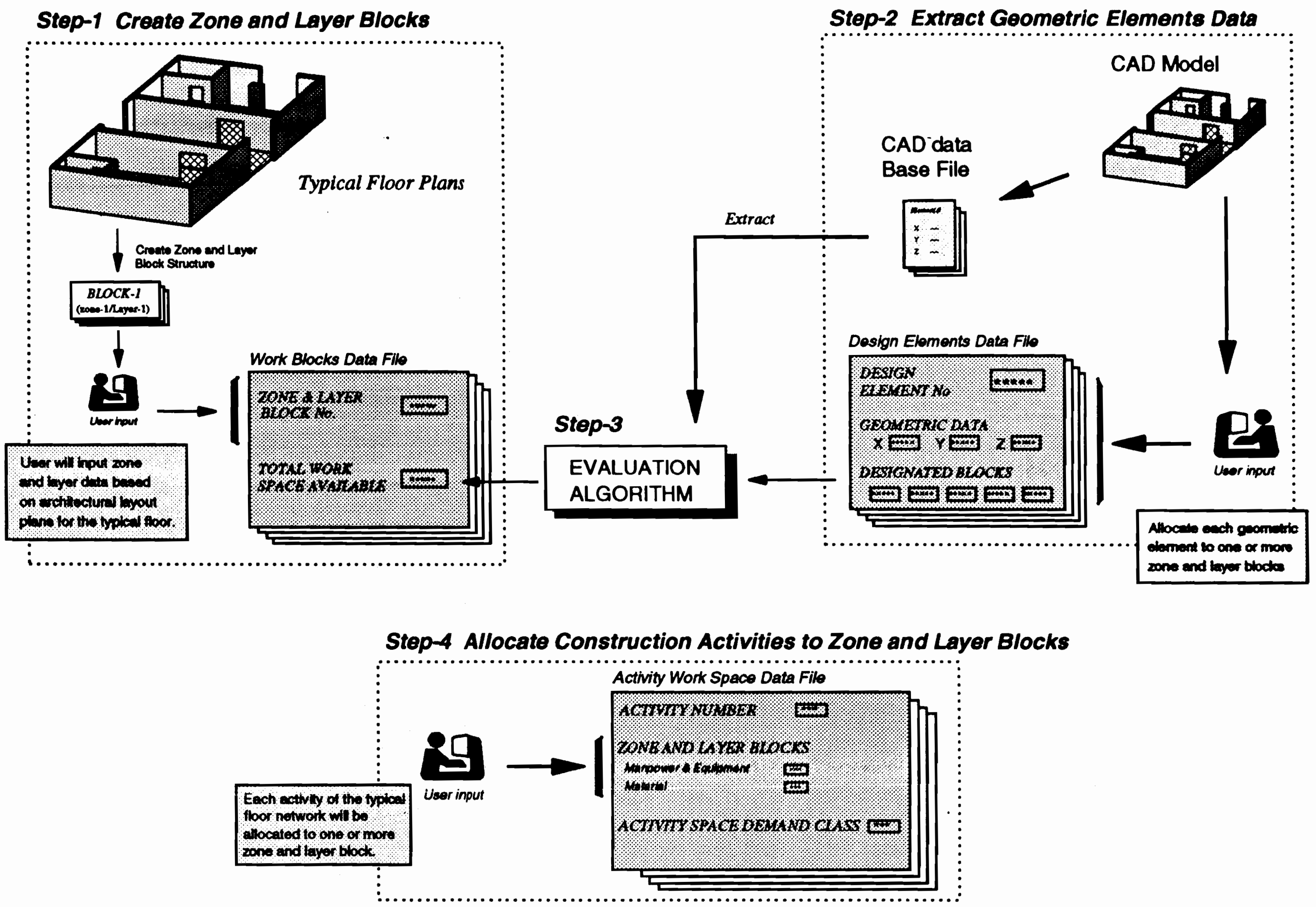
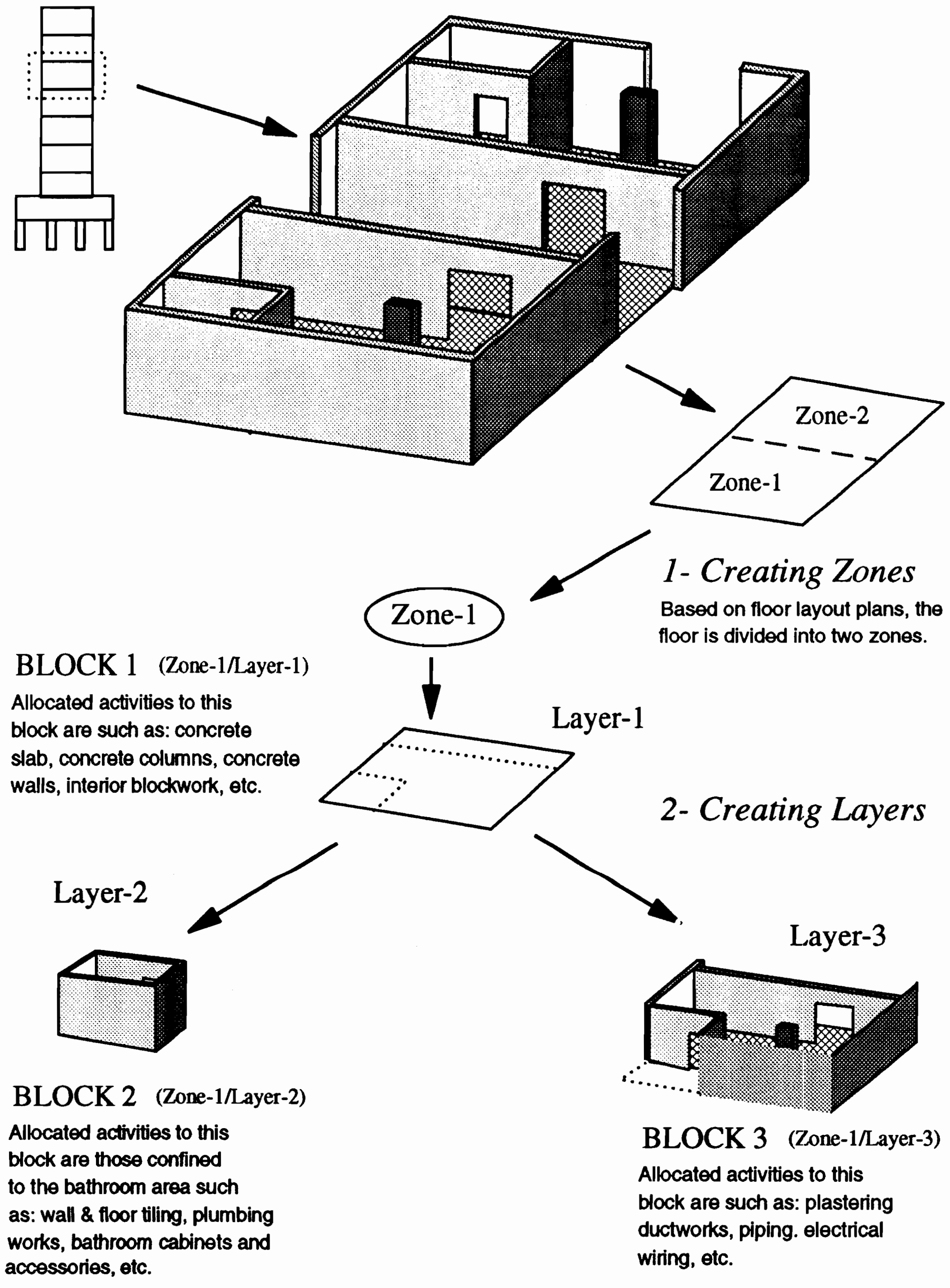

Step-1 Create Zone and Layer Blocks
Typical Floor Plans
Create Zone and Layer Block Structure
BLOCK-1 (zone-1/Layer-1)
User input
User will input zone and layer data based on architectural layout plans for the typical floor.
Work Blocks Data File
ZONE & LAYER BLOCK No.
TOTAL WORK SPACE AVAILABLE
Step-2 Extract Geometric Elements Data
CAD Model
CAD data Base File
Extract
Design Elements Data File
DESIGN ELEMENT No
GEOMETRIC DATA
X Y Z
DESIGNATED BLOCKS
User input
Allocate each geometric element to one or more zone and layer blocks
Step-3
EVALUATION ALGORITHM
Step-4 Allocate Construction Activities to Zone and Layer Blocks
Activity Work Space Data File
ACTIVITY NUMBER
ZONE AND LAYER BLOCKS
Manpower & Equipment
Material
ACTIVITY SPACE DEMAND CLASS
User input
Each activity of the typical floor network will be allocated to one or more zone and layer block.

Figure 7.6 - Evaluation of Work Space Available

continuous reduction of the total available space of the previous sub-zone.

Figure 7.7 illustrates the concept of zone and layer as applied to a typical floor of a multi-story building. The floor is first divided into two zones: zone-1 and zone-2. Each zone will represent a separate construction location and will constitute different layers. Layer-1 of zone-1 (block-1) represents the status of construction on the floor in zone 1 between day 1 and until all concrete and blockwork elements are completed. Activities allocated to this block will include those starting activities such as: concrete works and interior blockwork. Available space of block-1 will be the entire work space of the zone. Block-2 (zone-1/layer-2) will represent the bathroom area. Activities allocated to this block may include: bathroom plastering, ceramic wall and floor, bathroom cabinets, etc. Once cabinets are installed, a new block (Block-4) may be defined with new available space. Activities belonging to this block may include bathroom accessories and clean-up. Block-3 (zone-1/layer-3) constitutes the living room work area of the zone and will establish available space for activities in that area. This segmentation of the floor may be continued until all possible zone and layer blocks are established.

Zone and layer block numbers are stored in the *Work Blocks Data file* as shown in Step-1 of Figure 7.6.



1- Creating Zones

Based on floor layout plans, the floor is divided into two zones.

2- Creating Layers

BLOCK 1 (Zone-1/Layer-1)

Allocated activities to this block are such as: concrete slab, concrete columns, concrete walls, interior blockwork, etc.

BLOCK 2 (Zone-1/Layer-2)

Allocated activities to this block are those confined to the bathroom area such as: wall & floor tiling, plumbing works, bathroom cabinets and accessories, etc.

BLOCK 3 (Zone-1/Layer-3)

Allocated activities to this block are such as: plastering ductworks, piping. electrical wiring, etc.

Figure 7.7 - Segmentation of Floor into Zones and Layers

2- <u>Extract geometric data from design elements</u>

From a CAD model of the typical floor, all design elements geometric data are extracted. Element numbers and geometric data are stored in the Geometric Elements Data File. Each element is then linked to the zone and layer blocks defined in the Work Block Data file based on its location in the CAD model. If an element is located inside the boundary of a block, it will only be allocated to that block. On the other hand, if the design element shares the boundaries of two or more blocks, it will be allocated to all these blocks. This is illustrated in Step-2 of Figure 7.6.

For example, if two blocks share a concrete wall, the wall element will be allocated to both blocks. If the wall is inside the boundaries of one block and does not lie in the boundaries of any other block, the wall element will be allocated only to that block.

3- <u>Work Space Evaluation</u>

After the work block structure is defined and design elements are allocated to each block, the objective is to quantify total available space of each block. Available space for each block is calculated using a geometric algorithm utilizing geometric data of design elements allocated to that block. This is shown in Step-3 of Figure 7.6.

The procedure adopted by the algorithm first determines all design elements belonging to each zone and layer block. Once all elements of a specific block are established, boundary elements of the work block are identified. Using geometric data of the boundary elements, space enclosed within these elements is computed. To determine the net total available space of the work block, space consumed by all design elements inside the boundary of the block is deducted from the gross value.

4- Allocate Activities to Zone and Layer Blocks

Each activity is allocated to one or more blocks. The objective is to identify all activities belonging to a specific block. Given the total work space of the block, this allows to compute the space availability for any activity knowing the space consumed by other activities sharing the block. This is depicted in Step-4 of Figure 7.6.

Manpower and equipment allocated block is defined separately from the material block. This allows the verification of space demand of theses resources independently with available space of these blocks during scheduling. As discussed previously, this is because resource 'material' may occupy a different work location than that where resource 'manpower and equipment' are operating.

The proposed geometric algorithm and the database setup to evaluate work space availability are discussed in Chapter 9.

7.4 Space Capacity Factor

Scheduling activities in work areas with limited space availability may constitute unsafe working conditions and result in interruptions, delays and sometimes quarrels and fights among crews of the different trades. This, obviously, will have a negative impact on crew productivity rates as crews will become less efficient. Stated another way, crew productivity rate will decrease with decrease in work space availability as theoretically illustrated in Figure 7.8.

The impact of congestion on productivity could be viewed as analogous to the relationship between speed and traffic density (or concentration) on highways. It is found that, even with ideal road conditions, traffic volume tends to reach a maximum point at a relatively low speed [Wright, 79]. This phenomenon results from the fact that spacing allowed by the average driver when tailing another vehicle increases non-linearly with increases in speed. In other words, speed decreases exponentially with increases in traffic density as depicted in Figure 7.9.

The decrease of crew productivity in work areas with limited space availability can be utilized during scheduling of such activities. Instead of delaying an activity, the activity is scheduled with a lower production rate than the normal value. For example, assume that the normal productivity rate of crews allocated to an activity is 10 units/day. If available work space in any activity work block is less than the total space demand of the activity, then crew productivity is decreased (say by 20%) to account for the lack of work space. The activity is scheduled with a productivity rate of 8 units/day. The reduced productivity rate is used along those floors where space is determined to be insufficient.

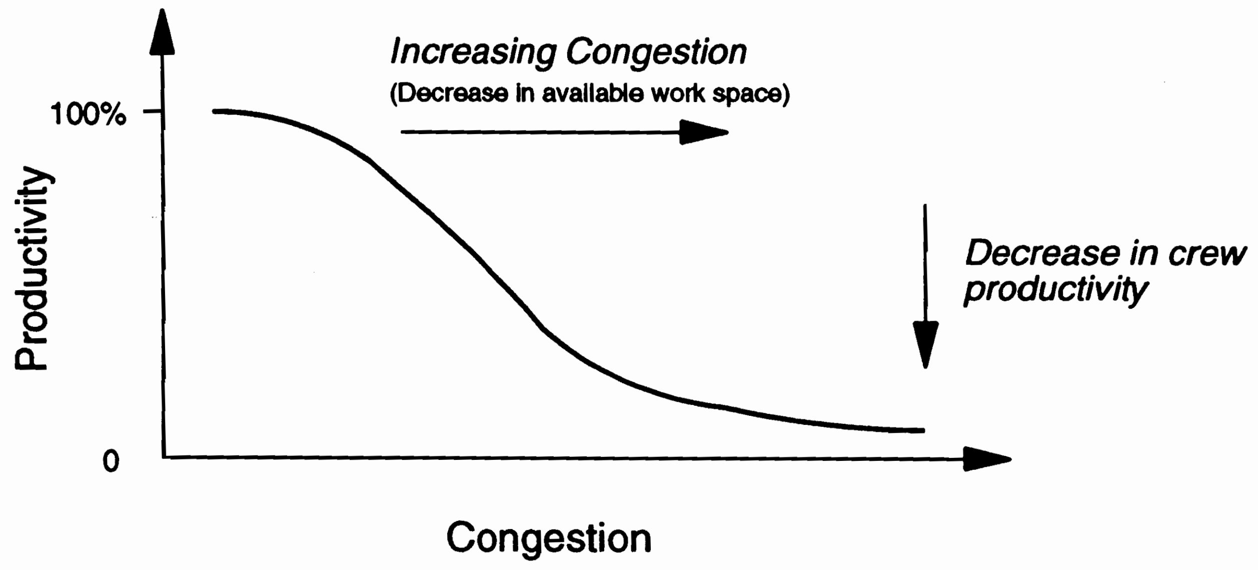


Figure 7.8 - Productivity versus Congestion

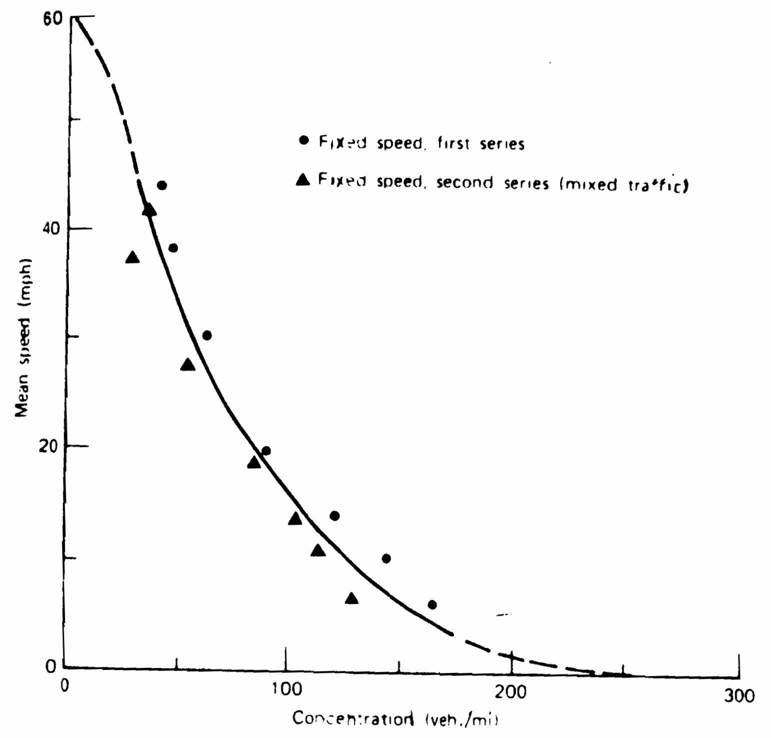


Figure 7.9 - Speed-Concentration curve for straight track at British Road Research Laboratory (courtesy Transportation Research Board)

(adopted from [Wright, 79])

Normal productivity rate of 10 units/floor is used to schedule the activity along other floors with sufficient work space.

To quantify the decrease in productivity rate because of limited work space a *Space Capacity Factor (SCF)* is proposed to measure the degree of congestion in any given work block of the floor. The factor computes the ratio of space demand required by the activity to the current available space of the work block. The proposed space capacity factor (SCF) is expressed as follows:

$$\text{Space Capacity Factor (SCF)} = \frac{\text{Space Demand for Activity}}{\text{Current Space Availability}} \quad \text{----(6)}$$

where: current space availability for the activity is the amount of available space of the work block as determined by equation (2). This is equal to the total computed work space of that block less any space occupied by concurrent activities already scheduled within the same time period of the activity.

Based on equation (6), if available space is equal to or greater than activity space demand, the factor will indicate a value of 1.0 or less. If available space is less than the demand, the factor will be greater than 1.0.

The space capacity factor can be plotted against different productivity rates for construction activities. Substituting SCF for congestion in Figure 7.8 results in the theoretical Productivity-SCF relationship illustrated in Figure 7.10. Figure 7.10 depicts an example Productivity-SCF curve. The curve theoretically reflects the variation (decline) in productivity for a hypothetical construction crew under variable space availability

$$\boldsymbol{SCF} = \frac{\text{Space Demand for Activity}}{\text{Current Space Available in Work Block }^{*}}$$

* *Current Available Space = Total Available Space of Work Block - Space Consumed by Other Activities Already Scheduled in Work Block*

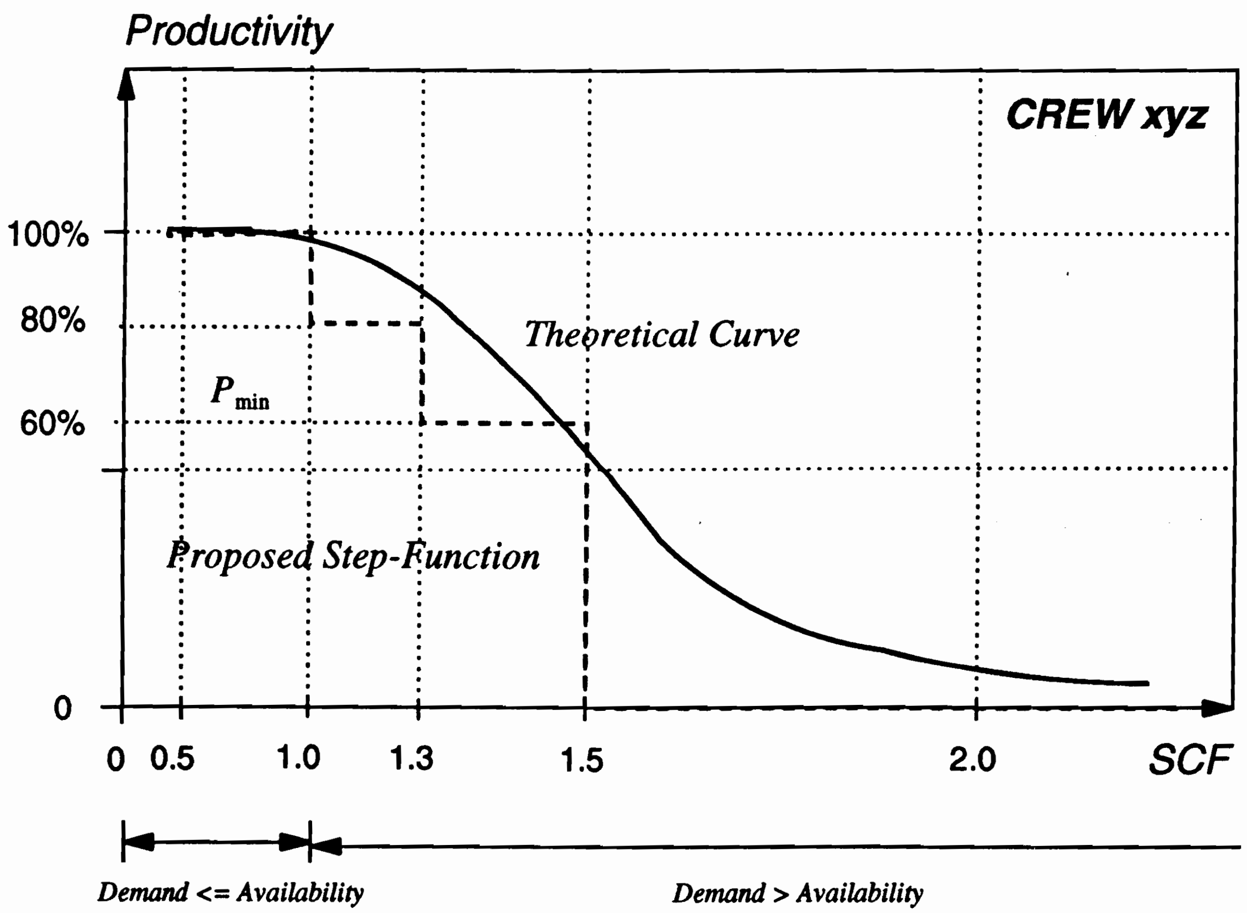


Figure 7.10 - Productivity-SCF RelationShip

conditions. As shown in Figure 7.10, when work space demand is less than or equal to the amount of available space for activity (i.e. SCF <= 1.0), the productivity of the crew remains unaffected (i.e. 100%). As available work space decreases, crew productivity decreases and drops to about 50% its normal value as available space becomes less than 2/3 of that required (i.e. SCF = 1.5).

Once lack of space is determined for an activity in a given work block, a space capacity factor is calculated for these floors. Using different SCF-Productivity graphs, the reduced rate of production for the activity is determined. The activity will be scheduled along the crowded floors using the new rate of production. The activity will be rescheduled using the normal rate once the space becomes available.

A simple step function is utilized to in the developed scheduling procedure of the model. As illustrated in Figure 7.10, the function defines four values of crew productivity for different ranges of SCF. The function is hypothetical for the purpose of the research and is utilized for all the activities to be scheduled. Actual graphs showing the relationship between the SCF and productivity rate for different activities or group of activities must be developed to reflect the actual variation of crew productivity under various limited space conditions. Such graphs may be constructed based on site observations and empirical equations. Developing these graphs is not within the scope of this research and is considered as future work.

7.5 Space-Based Scheduling Procedure

For each activity considered for scheduling, the scheduling procedure compares its space demand with the available space in each block designated to the activity. Values of space demand for manpower/equipment (SD-1) and space demand for material (SD-2) are compared to the space available in their corresponding blocks.

If sufficient space is available, the activity is scheduled to start at that period. To schedule an activity that is non-splittable or continuous, space must be available in the corresponding work blocks for the entire typical floors. If the activity will be split into two or more segments along the floors, each segment is considered for scheduling individually. Space is verified for each segment along the portion of the floors corresponding to that segment only.

If demand exceeds availability, scheduling decisions are made to take into account the lack of space. These decisions are made by considering various factors about the activity and taking the appropriate actions for the situation defined by these factors. Based on these actions some activity parameters are modified to allow the scheduling of the activity to start at the current time period, or delay the activity to a following time period.

Decision Factors

As illustrated in Figure 7.11, there are seven decision factors considered by the space scheduling model:

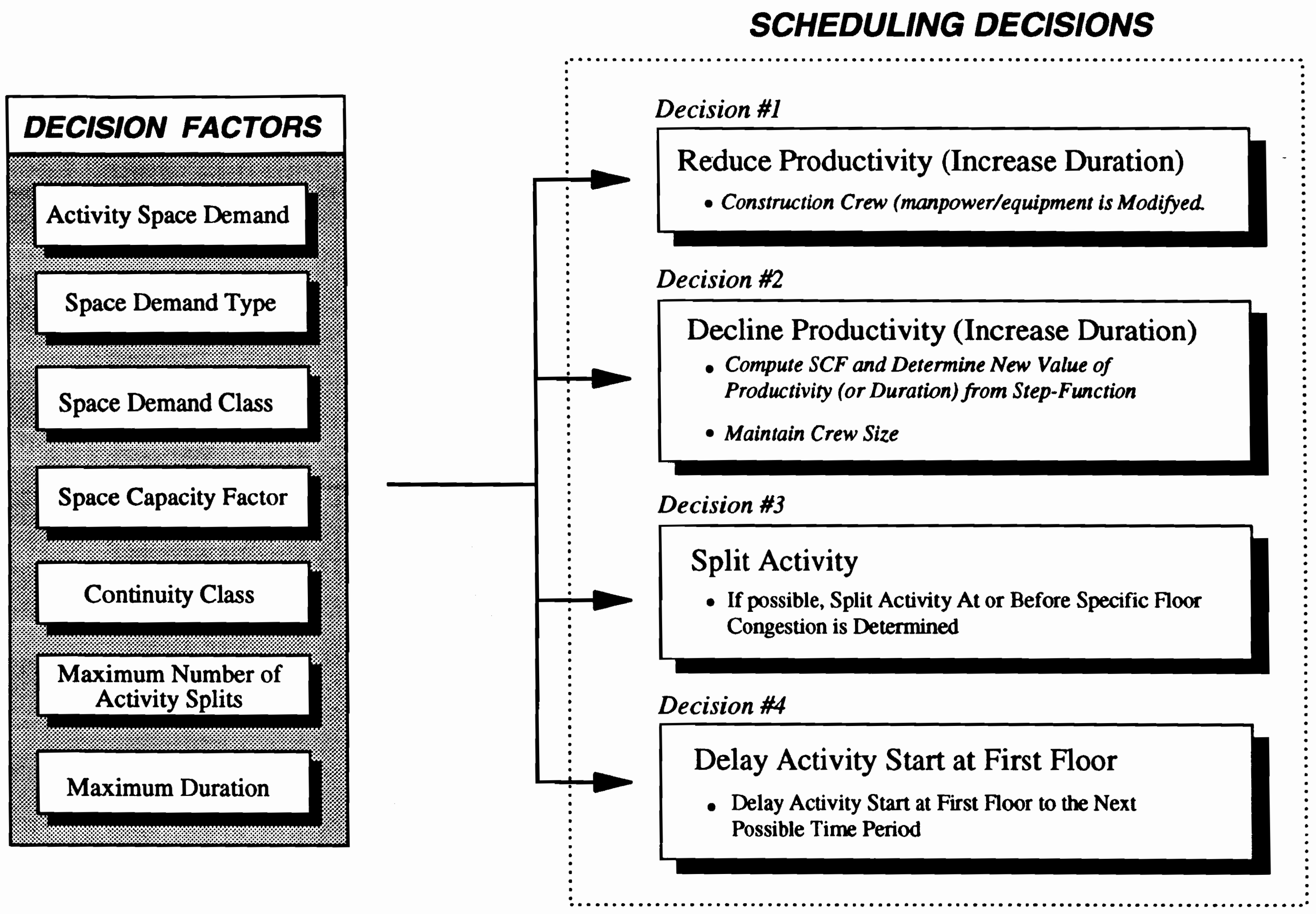

SCHEDULING DECISIONS
DECISION FACTORS
Activity Space Demand
Space Demand Type
Space Demand Class
Space Capacity Factor
Continuity Class
Maximum Number of Activity Splits
Maximum Duration
Decision #1
Reduce Productivity (Increase Duration)
• Construction Crew (manpower/equipment is Modifyed.
Decision #2
Decline Productivity (Increase Duration)
• Compute SCF and Determine New Value of Productivity (or Duration) from Step-Function
• Maintain Crew Size
Decision #3
Split Activity
• If possible, Split Activity At or Before Specific Floor Congestion is Determined
Decision #4
Delay Activity Start at First Floor
• Delay Activity Start at First Floor to the Next Possible Time Period

Figure 7.11 - Possible Decisions for Space-Based Scheduling

1- *Activity space demand.* This factor defines the amount of space required by the activity's allocated resources.

2- *Activity Space Demand Type.* Two space demand types are defined; space demand for manpower and equipment (SD-1), and space demand for material (SD-2).

3- *Activity space demand class (A, B or C).* Activities belonging to class A will not share the available space with other activities. Activities of class B and C will share the work space, if available, with other activities.

4- *Space capacity factor.* The factor is utilized in conjunction with the Productivity-SCF curves to determine the required decrease in productivity rate (i.e. increase in duration) for any activity.

5- *Activity continuity status.* The continuity status will determine whether an activity can be split into two or more segments along the typical floors or must be scheduled continuously.

6- *Maximum number of activity splits.* This number indicates the maximum number of times an activity can be split into segments along all the typical floors.

7- *Maximum Duration* (minimum productivity rate). Activity normal duration value reflects the activity's production under normal working conditions. A maximum activity duration value defined by the user will reflect the slowest rate at which the activity crew can operate.

Scheduling Decisions

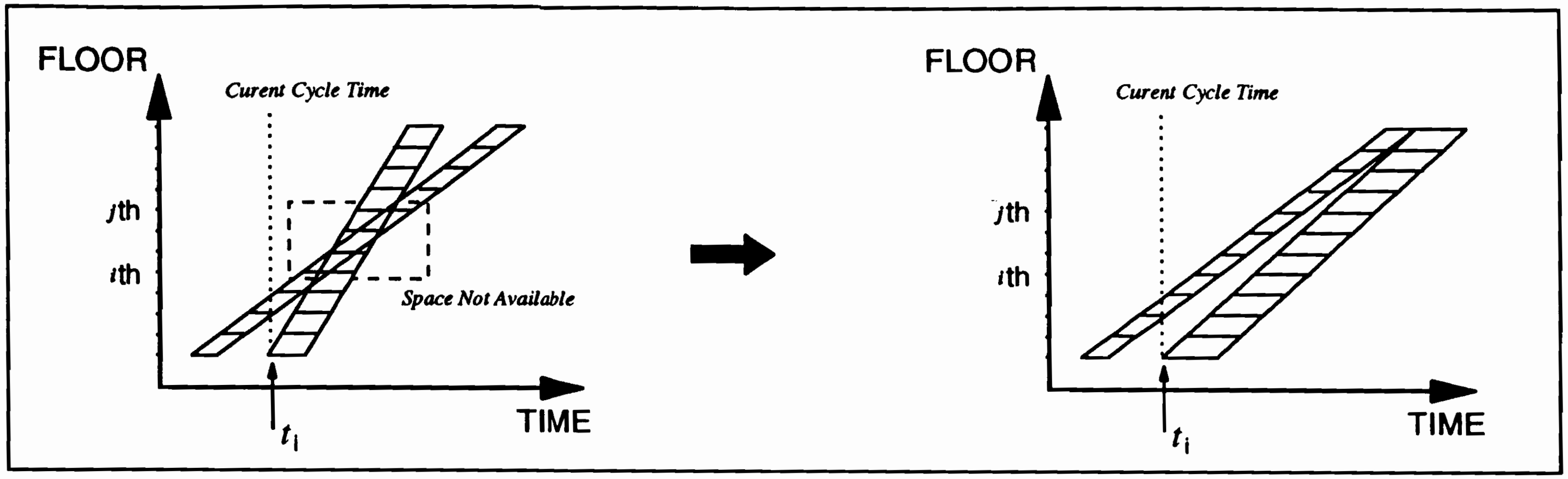
Based on these factors, decisions are made to either '*schedule*' or '*delay*' the activity during any time period. As illustrated in Figure 7.11, there are four decisions considered by the proposed scheduling model to account for lack of available work space during scheduling of each activity:

Decision-1: Decrease Production Rate The activity is scheduled by increasing activity duration in an attempt to steer the activity schedule outside the time period were work space is not available in the work area. The start date of the activity segment is maintained while its duration is increased. Activity resources (manpower and equipment) are modified, if possible, to reflect the new duration. The modified duration must not exceed the maximum value defined by the user. This decision option is depicted in Figure 7.12(a).

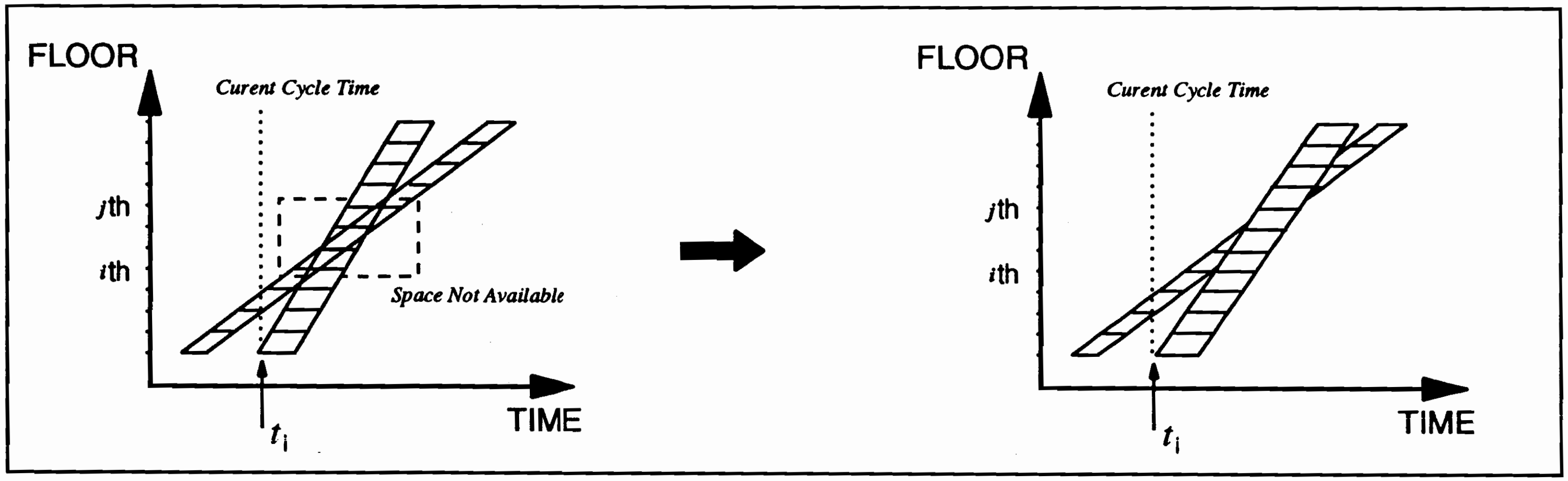
Decision-2: Decline Production Rate The activity is scheduled during the current period despite the lack of space. However, the activity production rate is decreased along the floors where available space is unsatisfactory to reflect the effect of congestion on crew productivity. Modified values of productivity rates is determined from Productivity-SCF step function curve based on a computed space capacity factor. As opposed to the previous decision option, activity resource are not modified. Figure 7.12(b) illustrates this decision option.

This scheduling action is only applicable for those activities with a space demand class B or C. Activities of class A can not be scheduled to share space with other activities. If space is not available for this class of activities, they must be delayed to a time period were no other activity is already scheduled.

It should also be noted that decreasing productivity rate is only considered if limited work space is present in construction areas where manpower and equipment operate. That is, when the value of work space available is less than space demand for resource manpower and equipment (SD-1) only. If required space for material storage



a) Decision #1: Reduce Productivity (Increase Duration)
(Modify Construction Crew Size)



b) Decision #2: Decline Productivity (Increase Duration)
(Maintain Construction Crew Size)

Figure 7.12 - Scheduling Decisions Considered by The Procedure

(SD-2) is not satisfactory, this option will not be applicable to account for limited work space during scheduling the activity. In this case, the scheduling procedure must consider one of two other actions discussed next.

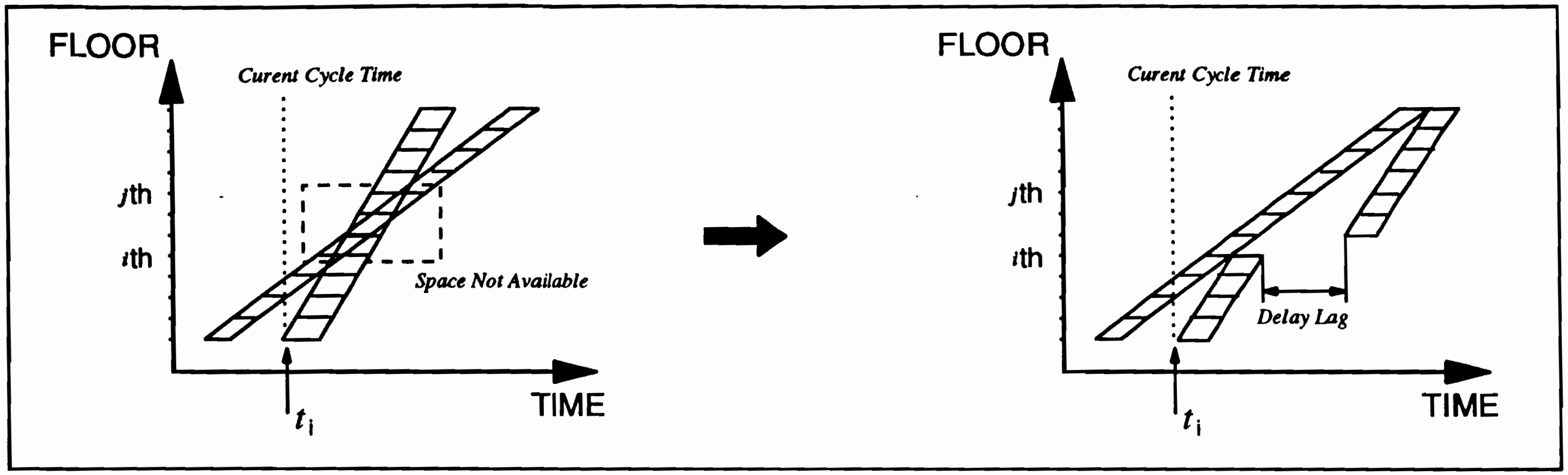
Decision-3: Split Activity

As illustrated in Figure 7.13(c), this option allows interruption of activity work flow at or before the specific floor where required work space is unavailable. The floor(s) at which work space is unavailable is first determined. Work flow is then interrupted by splitting the activity at or before these floors. The portion of the floors with available space is scheduled to start at the current time period. The unscheduled portion of the activity along the remainder of the floors is delayed to a following time period. This procedure is repeated, if necessary, with splitting the activity at or before the specific floor where work space is found unavailable.

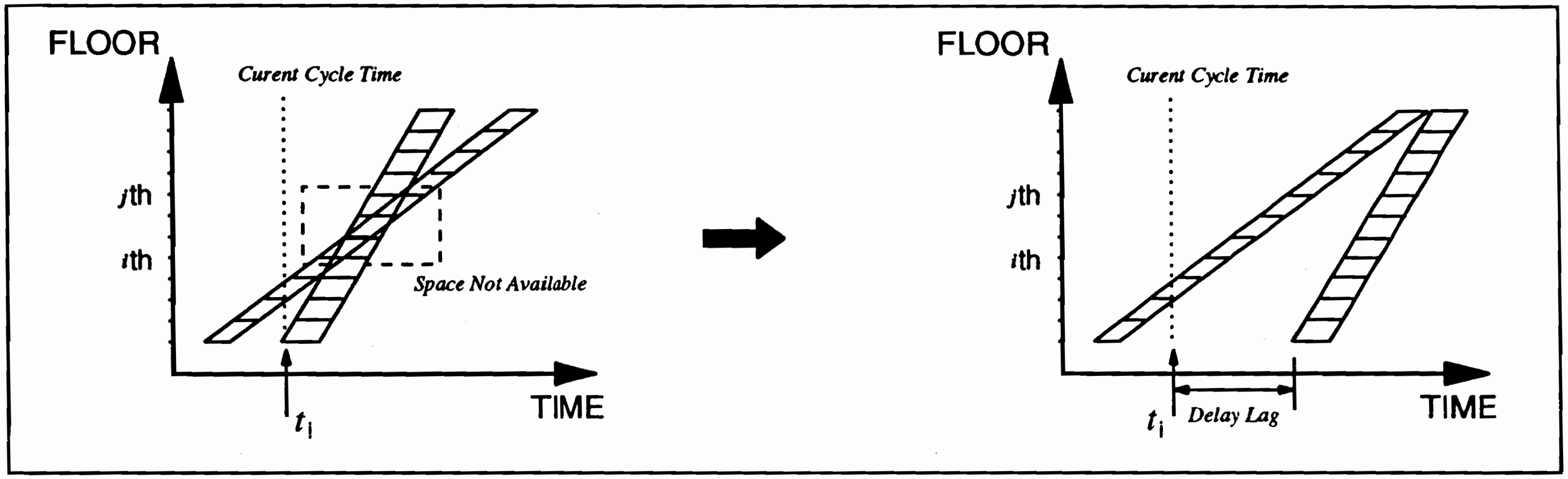
This option is dependent on the continuity status of the activity as defined by the user. Interruption of work flow is only be possible for activities that can be split during scheduling.

Decision-4: Delay Activity Start at First Floor

In this option, the procedure delays the start of the entire activity along all the floors to the next possible time period. This is depicted in Figure 7.13(d). The main disadvantage of this option is it will delay the start of the entire activity along the



c) Decision #3: Split Activity



d) Decision #4: Delay Activity Start at First Floor

Figure 7.13 - Scheduling Decisions Considered by The Procedure

typical floors. This in turn may delay the start of all succeeding activities and may result in increasing overall project duration.

Work Space Scheduling Procedure

The flowchart illustrated in Figure 7.14 depicts the logic employed by the scheduling procedure of the model. A step-by-step description of this logic is as follows:

Step-1 *Define set of eligible activities to schedule.* As the scheduling process is started and the cycle time is defined (t_i), the procedure determines the set of eligible activities that can be considered for scheduling at time t_i.

Step-2 *Verify other constraints.* As an activity is selected for scheduling from the set of eligible activities, the procedure first verifies other constraints and insures that they are satisfactory. Such constraints may include: imposed dates, weather constraints, constructability constraints, resource constraints, safety issues and code regulations, etc.

If these constraints are not satisfactory, the activity is dropped from the scheduling cycle (2a). The next possible time to reconsider the activity for scheduling is determined.

If all constraints are satisfactory, preliminary start/finish dates are computed (2b) and

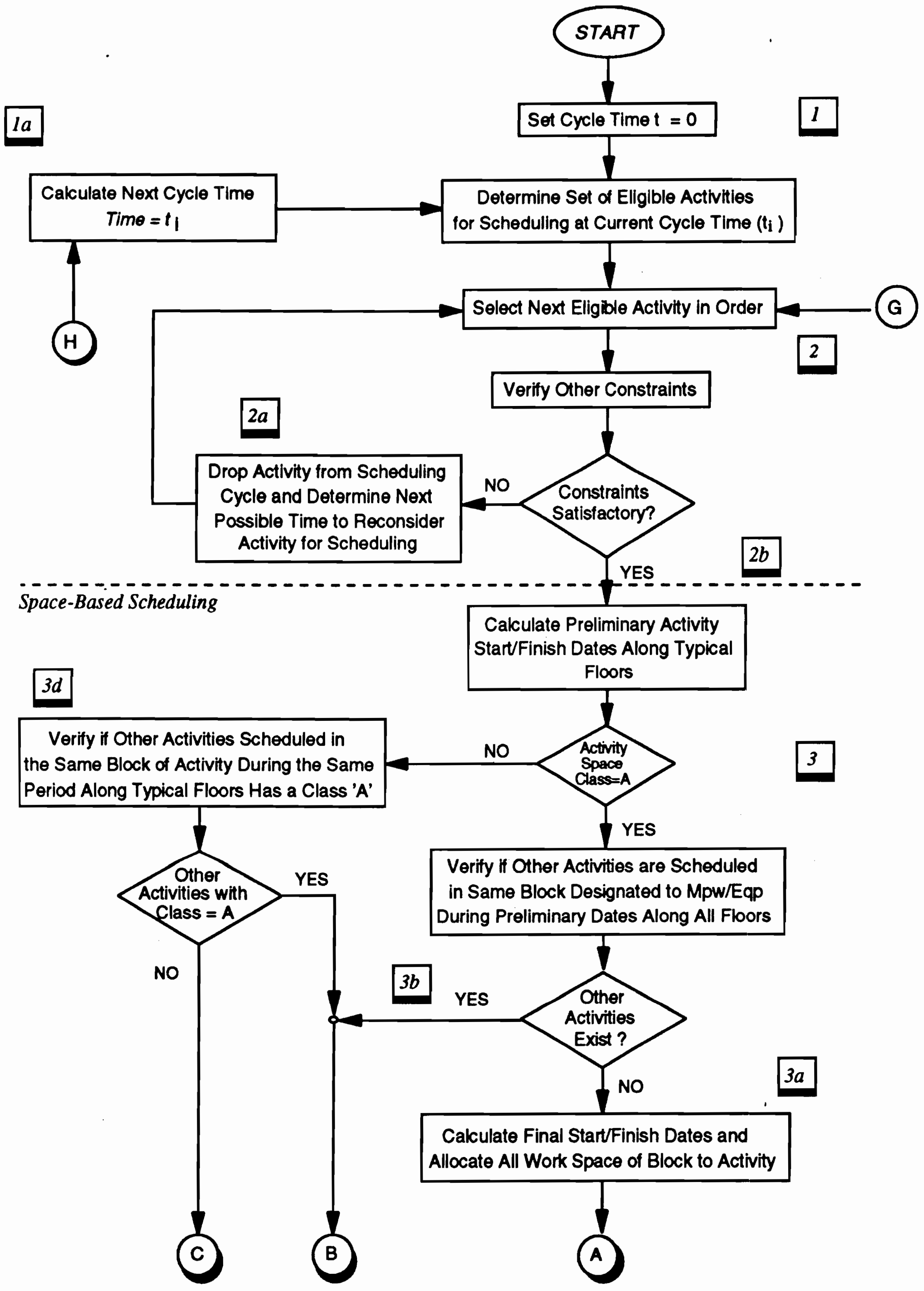

START
Set Cycle Time t = 0
1
1a
Calculate Next Cycle Time
Time = t_i
Determine Set of Eligible Activities for Scheduling at Current Cycle Time (t_i)
Select Next Eligible Activity in Order
G
H
2
Verify Other Constraints
2a
Drop Activity from Scheduling Cycle and Determine Next Possible Time to Reconsider Activity for Scheduling
NO
Constraints Satisfactory?
2b
YES
Space-Based Scheduling
Calculate Preliminary Activity Start/Finish Dates Along Typical Floors
3d
Verify if Other Activities Scheduled in the Same Block of Activity During the Same Period Along Typical Floors Has a Class 'A'
NO
Activity Space Class=A
3
YES
Verify if Other Activities are Scheduled in Same Block Designated to Mpw/Eqp During Preliminary Dates Along All Floors
Other Activities with Class = A
YES
NO
3b
YES
Other Activities Exist ?
NO
3a
Calculate Final Start/Finish Dates and Allocate All Work Space of Block to Activity
C
B
A

Figure 7.14 - Space-Based Scheduling Procedure

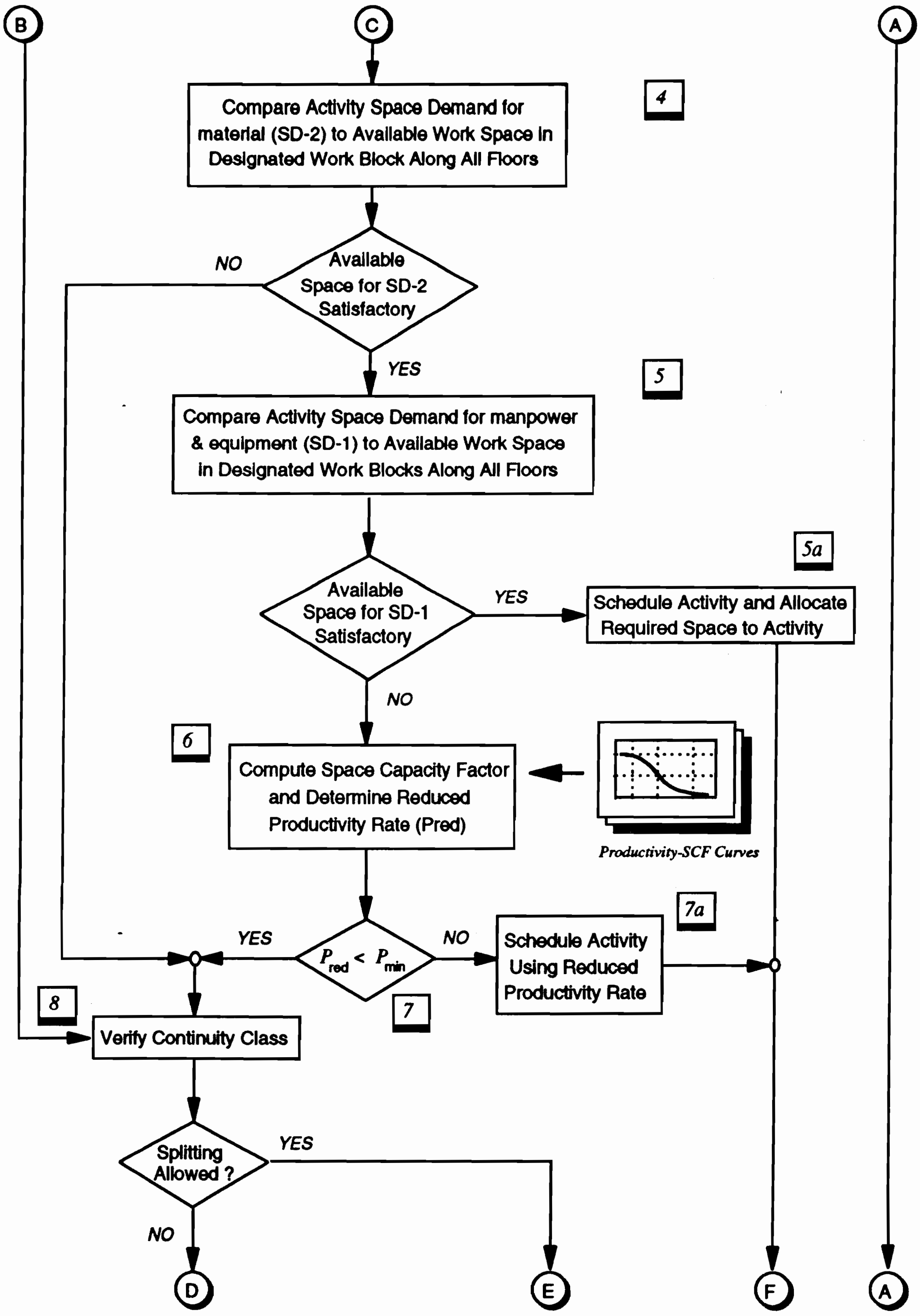

B
C
A
Compare Activity Space Demand for material (SD-2) to Available Work Space in Designated Work Block Along All Floors
4
NO
Available Space for SD-2 Satisfactory
YES
5
Compare Activity Space Demand for manpower & equipment (SD-1) to Available Work Space in Designated Work Blocks Along All Floors
5a
Available Space for SD-1 Satisfactory
YES
Schedule Activity and Allocate Required Space to Activity
NO
6
Compute Space Capacity Factor and Determine Reduced Productivity Rate (Pred)
Productivity-SCF Curves
7a
YES
$P_{red} < P_{min}$
NO
Schedule Activity Using Reduced Productivity Rate
8
7
Verify Continuity Class
Splitting Allowed ?
YES
NO
D
E
F
A

Figure 7.14 - Continued

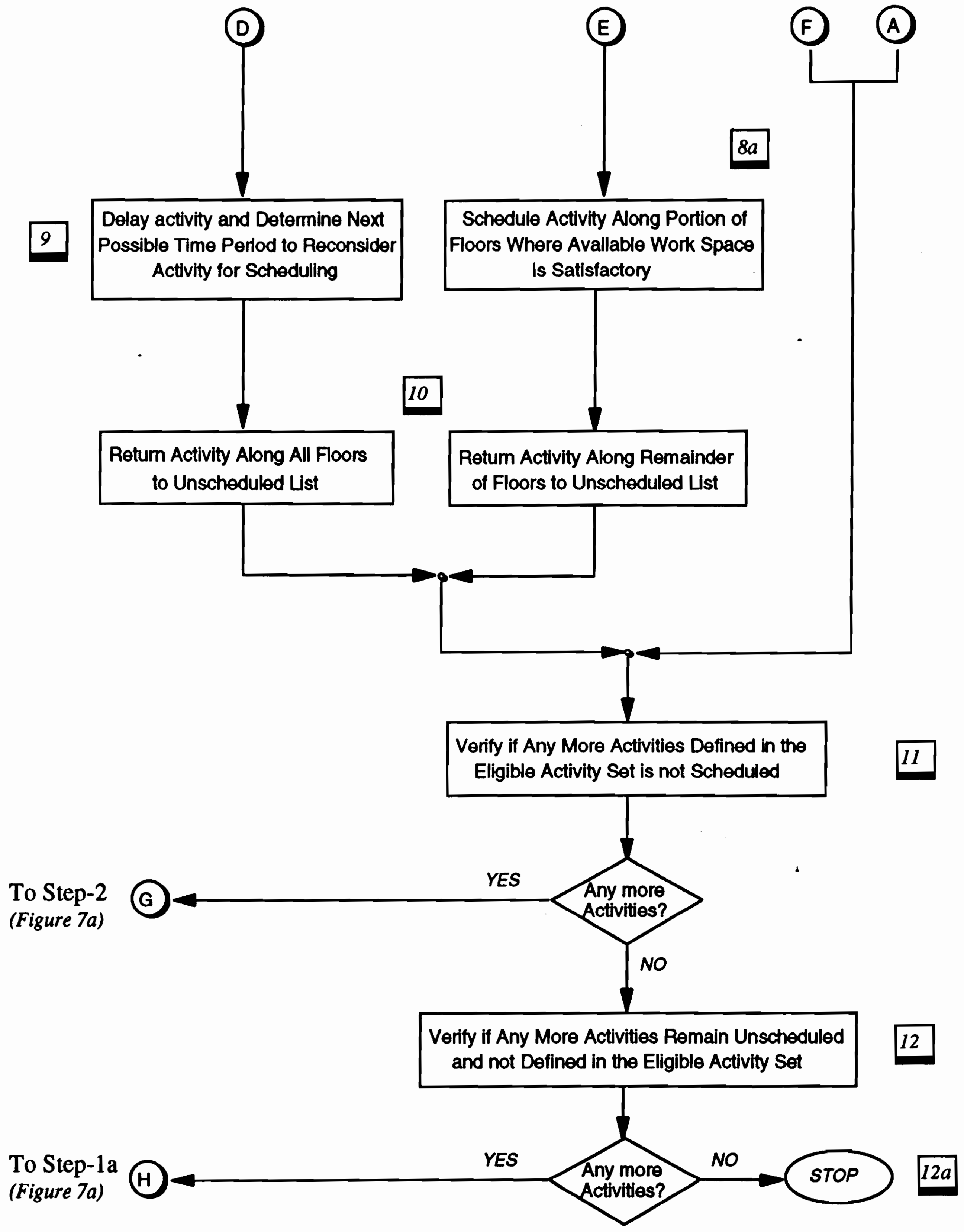

D
E
F
A
8a
9
Delay activity and Determine Next Possible Time Period to Reconsider Activity for Scheduling
Schedule Activity Along Portion of Floors Where Available Work Space is Satisfactory
10
Return Activity Along All Floors to Unscheduled List
Return Activity Along Remainder of Floors to Unscheduled List
Verify if Any More Activities Defined in the Eligible Activity Set is not Scheduled
11
To Step-2 (Figure 7a)
G
YES
Any more Activities?
NO
Verify if Any More Activities Remain Unscheduled and not Defined in the Eligible Activity Set
12
To Step-1a (Figure 7a)
H
YES
Any more Activities?
NO
STOP
12a

Figure 7.14 - Continued

activity is checked for work space constraints.

Step-3 *Verify space demand class for activity.* The space demand class for activity is identified. If activity has of a class A space demand, the procedure verifies that no other activity is already scheduled in the same block (for manpower and equipment) along all the floors during the calculated preliminary start and finish dates. If no activities are scheduled in the block during the same time periods at any floor, the activity is scheduled and total space of work blocks in each floor is allocated to the activity (3a). Once the activity is scheduled, the procedure verifies if there are any more activities to be considered for scheduling during the current cycle (follow A to step-11).

If one or more activities are already scheduled in the block at one or more floors during the scheduled preliminary dates (3b), the procedure checks if the activity can be split at or before the floor where other activities are found already scheduled (follow B to step-8).

If the activity is not of class A (i.e. class B or C), the scheduling procedure verifies that no other activities of class A are already scheduled in the same block during the preliminary dates (3c). If other activities of a class A are already scheduled in the same block at any floor, the activity can not be scheduled. The procedure attempts to split the current activity at or before the floors where other class A activities are already scheduled (follow B to step-8).

If no other activities of class A are scheduled in the current activity block during the same time periods along any floor, the activity can be scheduled if work space is available. At this point the procedure moves to determine space availability in each block and compares it with activity space demand (follow C to step-4).

Step-4 *Verify space availability for resource 'material'*. The procedure first compares space demand for materials (SD-2) with available space in the designated material work block. If available space is satisfactory, the procedure moves to compare space demand for manpower and equipment (SD-1) with available space in the designated manpower/equipment work block (step-5).

On the other hand, if available space for material is not satisfactory decisions are made to either split the activity or delay its start to the next possible time period. The procedure first checks the possibility of splitting the activity at or before the floor where space becomes unavailable for SD-2 (step-8). If this option is found unsuccessful, the procedure delays the start of activity along all typical floors and drop it from scheduling cycle (step-9).

Step-5 *Verify space availability for resource 'manpower/equipment'*. Space is verified for manpower and equipment at the designated work block. If available space is satisfactory, the activity is scheduled and work space is allocated to the activity in the amount required (5a). At this point, the scheduling procedure checks if there is any other activities defined in the eligible activity set is to be considered for scheduling during the current cycle (follow F to step-11).

If available space is not satisfactory for manpower and equipment, the procedure first computes the space capacity factor for the block and attempts to schedule the activity using a reduced crew productivity rate (step-6). If not possible, the procedure then attempts to split the activity at or before the floors where space becomes unavailable (step-8). If this action is also not applicable, the procedure delays the start of the activity along all floors to the next possible time period (step-9).

Step-6 *Compute Space Capacity Factor (SCF).* To account for unavailability of work space for resource 'manpower/equipment', the procedure first considers reducing the normal productivity rate of activity. Space capacity factor is calculated for each floor where available space is not satisfactory. For each floor with a SCF value less than 1.0 the procedure determines the reduced crew productivity rate (P_{red}) using Productivity-SCF relationship curves. If the value of P_{red} varies for each floor, an average value for all floors is computed.

Step-7 *Compare P_{red} to minimum crew productivity rate (P_{min}).* If the average calculated value of P_{red} is equal to or greater than P_{min} defined by the curve based on the maximum SCF allowed, the activity is scheduled along the floors. Start and finish dates are calculated based on the new reduced productivity rate and space is allocated to activity by the amount required (7a). At this point, the procedure checks if there is any other activities defined in the set of eligible activity set to be considered for scheduling at the current cycle (follow F to step-11).

If the value of P_{red} fell below P_{min}, this option is not considered and the activity attempt to split the activity at or before the floors where available space is not satisfactory (step-8).

Step-8 *Check possibility to split the activity.* The second possible action considered by the procedure is to check whether the flow of work could be interrupted. If the activity is splittable, the procedure splits the activity at or before the floors at which work space becomes limited or unavailable. The activity is scheduled along the portion of the floors with available space. The portion of the activity along the remainder of floors is delayed and returned to the queue to be reconsidered for scheduling in the next possible time period (follow E to step-8a).

If the activity is non-splittable and must be scheduled continuously along all typical floors, activity can not be split and must be delayed. The activity is returned to the scheduling queue to be reconsidered for scheduling in a following time period (follow D to step-9).

Step-9 *Delay activity start to next possible time period.* Activity is not scheduled and is delayed to a following time period. If possible, the procedure computes the next possible time period for the activity to be reconsidered for scheduling.

Step-10 *Return activity to scheduling queue.* Return activity to list of unscheduled activities to be reconsidered for scheduling at a following time period.

Step-11 *Consider next activity from eligible activity set.* Once the current activity is 'scheduled' or 'delayed', the procedure checks the list of eligible activities to verify if any more activities remain unscheduled during the current time period (t_i). If the list of eligible activities is not empty the entire process (i.e. step-2 through step-10) is repeated (follow G back to step-2). Otherwise, the procedure terminates the current scheduling cycle and determines the next time period for the following cycle (step-12).

Step-12 *Modify list of eligible activities and start a new scheduling cycle.* Once all activities defined in the eligible activity list are considered for scheduling and the cycle is completed, the procedure checks if any more activities remain unscheduled. If there is any unscheduled activities, the procedure determines the new cycle time, modifies the set of eligible activities, and starts a new scheduling cycle (follow H to step-1a). Once all the typical network activities are scheduled, the scheduling process is terminated (12a).

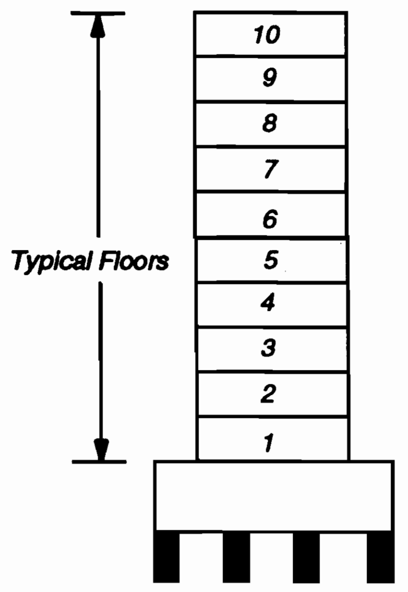
The proposed space-based scheduling procedure is integrated with other procedures to verify all constraints considered by the SCaRC scheduling system. The procedure is discussed in more detail in Appendix C.

7.5 Worked Example

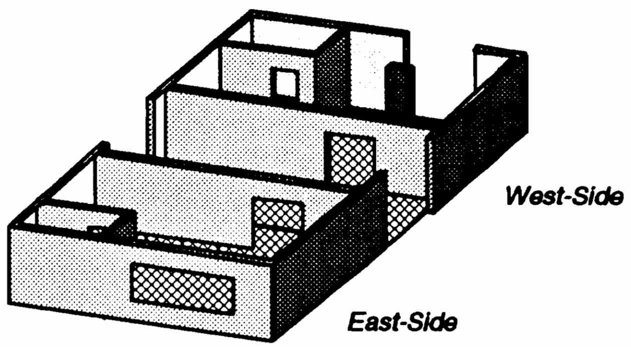
This section provides a worked example to demonstrate the proposed model for scheduling repetitive floors utilizing work space as a construction constraint. In a step-by-step procedure, the example generates a space-based construction schedule based only on limited space availability. Other construction constraints (e.g. resources, vertical logic, etc.) are not considered in the example.

The proposed space-based procedure is applied to high rise project shown in Figure 7.15. The figure shows a 3D model of the floor along with a plan view with the floor dimensions. The example high rise consists of 10 typical floors. A typical high rise building usually exceeds 20 stories. However, since the purpose of this example is to demonstrate the theory of the proposed model, only a 10 story hypothetical building is considered.

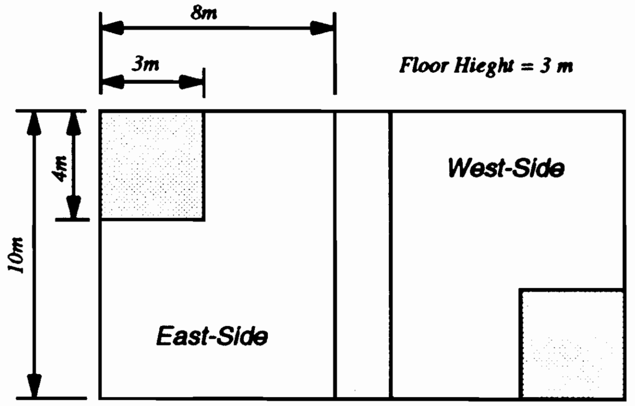
The typical floor logical network with schedule calculation for one floor is depicted in Figure 7.16. The figure also shows a basic schedule for all typical floors. The schedule is generated with no consideration to any continuity issues or any scheduling constraints.



a) Typical Floors



b) CAD Model of Typical Floor



c) Typical Floor Plan

Figure 7.15 - High Rise Example Project

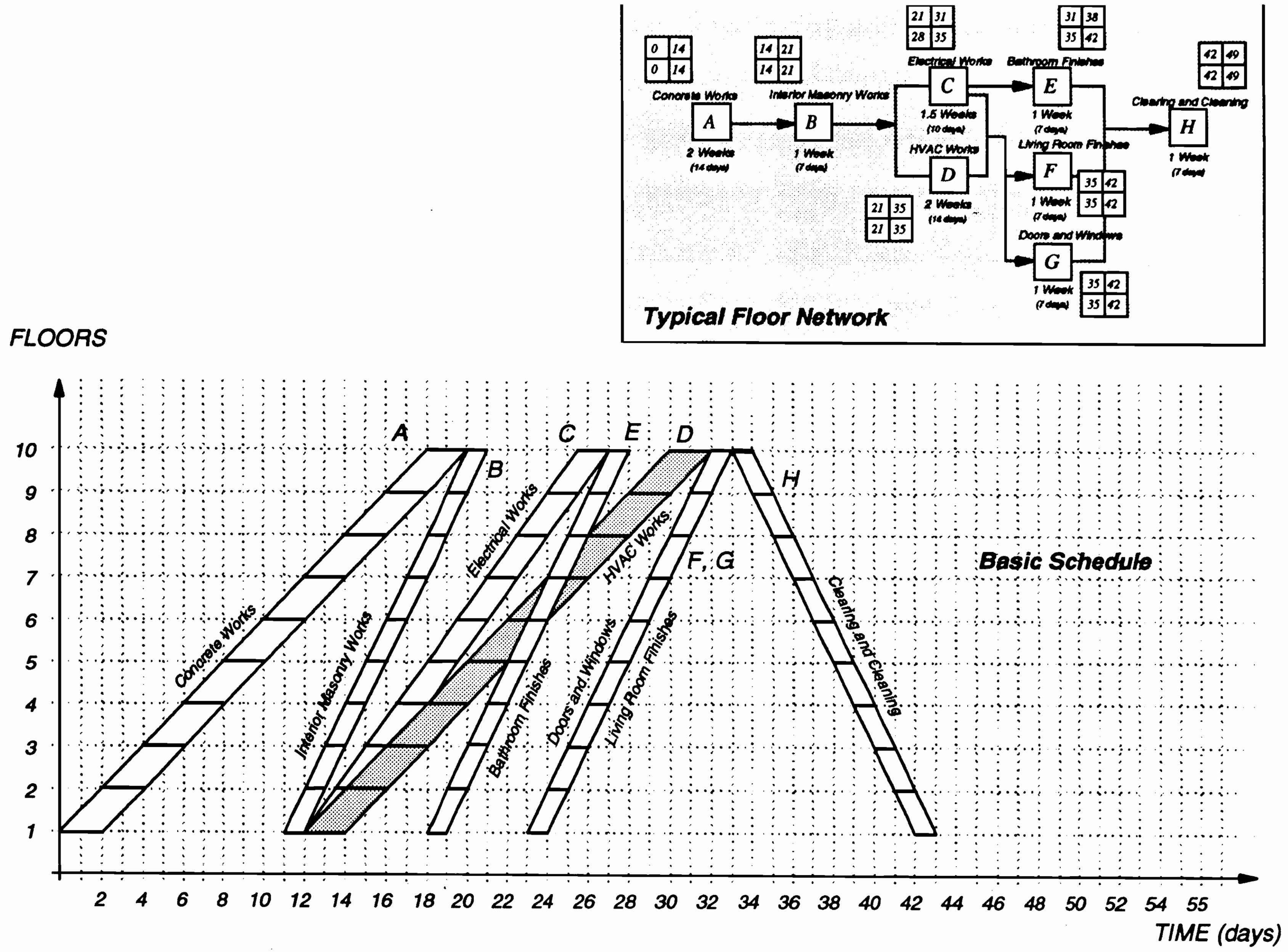


Figure 7.16 - Activity Network and Basic Schedule

Comparing this basic schedule to the space-based schedule generated at the end of this example emphasizes the effect of limited work space availability on the schedule generation process.

Activity attributes and scheduling parameters are given in Figure 7.17(a). A hypothetical SCF-Productivity curve is shown in Figure 7.17(b). Based on the proposed step function, the productivity rate of the construction manpower is 100% of the normal rate defined by user for values of SCF equal or greater than 1.0. The productivity rate drops to 80% for values of SCF between 1.0 and 1.3. Between values of 1.3 to 1.5 of SCF, the productivity is 60%. For SCF values over 1.5, no productivity for the manpower is considered (i.e productivity = 0). Heuristic rules defining the ranking criteria to be used during the scheduling process are provided in Table-3 in Figure 7.17(c).

Figure 7.18 illustrates the database setup for evaluating work space demand for each activity as described earlier in section 7.2. Using the *Activity Resource Demand data file* and the *Global Space Demand data file*, the space demand for each resource of the project is determined as follows:

1- For each resource type defined by a record in the Activity Resource Demand Data file, the corresponding record that contains the same resource type in the Global Space Demand data file is determined.

2- Once the required record is located, the global space demand values ($S_{physical}$ and $S_{surrounding}$) for the unit resource are extracted.

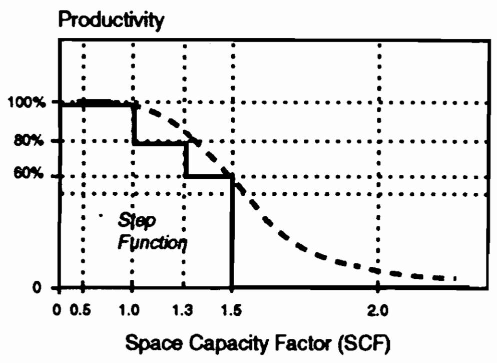
3- The extracted values are stored in the corresponding fields of the search record in

Activity Number		A	B	C	D	E	F	G	H
Workflow Dir		UP	UP	UP	UP	UP	UP	UP	DOWN
nor-act-dur (DAYS)		14	7	10	14	7	7	7	7
max-act-dur (DAYS)		14	7	10	14	7	7	7	7
Typical Floor Network	Prec-act	-	A	B	B	C	C,D	C,D	E,F G
	Succ-act	B	C,D	E,F G	F,G	H	H	H	-
Continuity Class		A	A	B	A	B	A	A	B
Max-splits		-	-	2	-	2	-	-	10
Space Demand Class		A	A	B	A	B	B	B	B

A) Table-1 Activity Scheduling Data

Table-2 Values are Based on Step Function

SCF		Productivity
from	to	
0	1.0	100 %
1.0	1.3	80 %
1.3	1.5	50 %
1.5	~	0



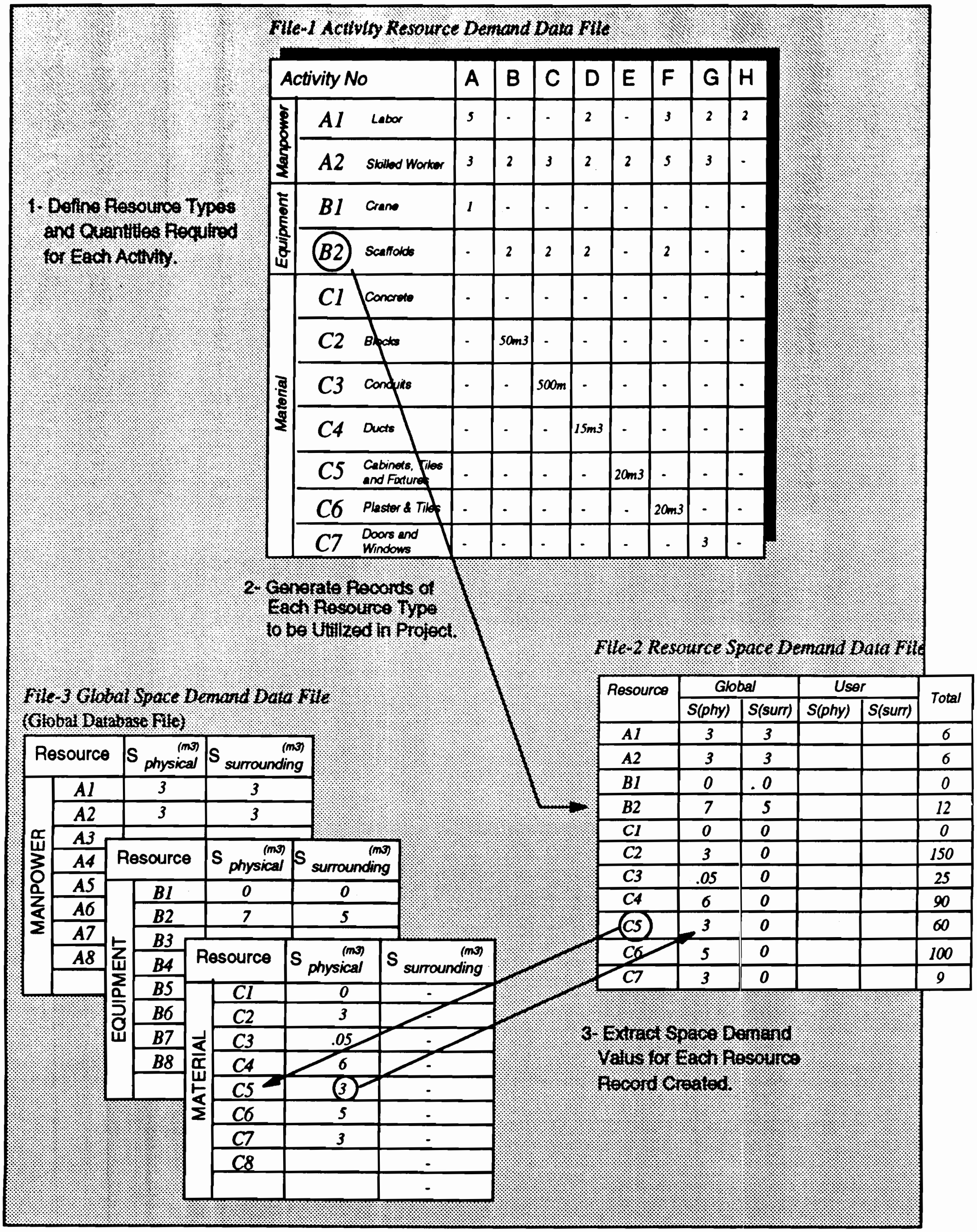
b) Hypothetical SCF-Productivity Curve and Step Function.

Table-3 Ranking Criteria

Order	Heuristic
1	Minimum Late Start
2	Minimum Normal Duration
3	Minimum Maximum Duration

c) Ranking Criteria Table

Figure 7.17 - Activity Data



File-1 Activity Resource Demand Data File

Activity No			A	B	C	D	E	F	G	H
Manpower	A1	Labor	5	-	-	2	-	3	2	2
	A2	Skilled Worker	3	2	3	2	2	5	3	-
Equipment	B1	Crane	1	-	-	-	-	-	-	-
	B2	Scaffolds	-	2	2	2	-	2	-	-
Material	C1	Concrete	-	-	-	-	-	-	-	-
	C2	Blocks	-	50m3	-	-	-	-	-	-
	C3	Conduits	-	-	500m	-	-	-	-	-
	C4	Ducts	-	-	-	15m3	-	-	-	-
	C5	Cabinets, Tiles and Fixtures	-	-	-	-	20m3	-	-	-
	C6	Plaster & Tiles	-	-	-	-	-	20m3	-	-
	C7	Doors and Windows	-	-	-	-	-	-	3	-

1- Define Resource Types and Quantities Required for Each Activity.

2- Generate Records of Each Resource Type to be Utilized in Project.

File-2 Resource Space Demand Data Fil

Resource	Global		User		Total
	S(phy)	S(surr)	S(phy)	S(surr)	
A1	3	3			6
A2	3	3			6
B1	0	. 0			0
B2	7	5			12
C1	0	0			0
C2	3	0			150
C3	.05	0			25
C4	6	0			90
C5	3	0			60
C6	5	0			100
C7	3	0			9

File-3 Global Space Demand Data File
(Global Database File)

Resource		S physical (m3)	S surrounding (m3)
MANPOWER	A1	3	3
	A2	3	3
	A3		
	A4		
	A5		
	A6		
	A7		
	A8		

Resource		S physical (m3)	S surrounding (m3)
EQUIPMENT	B1	0	0
	B2	7	5
	B3		
	B4		
	B5		
	B6		
	B7		
	B8		

Resource		S physical (m3)	S surrounding (m3)
MATERIAL	C1	0	-
	C2	3	-
	C3	.05	-
	C4	6	-
	C5	3	-
	C6	5	-
	C7	3	-
	C8		-
			-

3- Extract Space Demand Valus for Each Resource Record Created.

Figure 7.18 - Defining Space Demand for Activity Resources

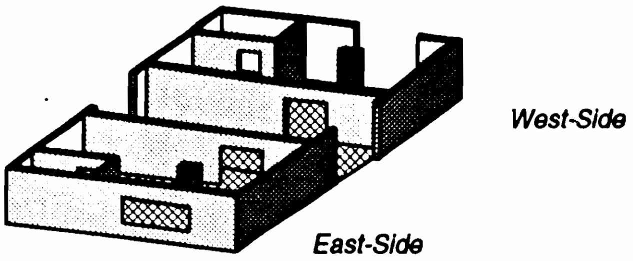
the Resource Space Demand Data file.

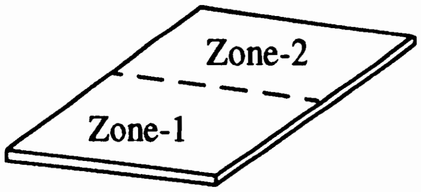
4- Values may be estimated by the user based on the extracted global values and a total value for the space demand of the resource id defined.

Steps 1 through 4 are repeated for all resource records defined in the Activity Resource Requirement data file.

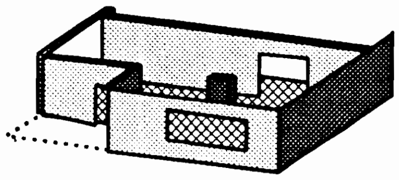
To define space availability, the typical floor is divided into three work blocks as illustrated in Figure 7.19. For the purpose of this example, total available space values for each block is calculated using the dimensions of each block defined in Figure 7-15. A separate algorithm is developed to compute work space for each block using the CAD model of the floor as discussed previously in section 7.3. Activities are allocated to each based on the work nature of the activities and the work area where each activity is assigned. Activity manpower and equipment is allocated separately from activity material. Table-4 of Figure 7.19 illustrates the block numbers designated to each activity. As shown in Table-4, manpower and equipment for some activities are allocated to different blocks than the material. This indicates that the material for those activities is stored in a different block than that where the actual activity is performed.

Example Scheduling Procedure: The scheduling of the activities in this example uses a parallel approach. The scheduling procedure moves through time period-by-period. For each time period scheduling decisions are made to schedule all activities that are eligible for scheduling after examining the availability of work space.

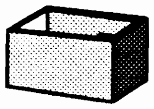




Block 1
(zone 1, Layer 1)
Total Avialble Space = 8 x 10 x 3 = 240 m3



Block 2
(zone 1, Layer 2)
Total Avialble Space = 240 - (3 x 4 x 3) = 204 m^3



Block 3
(zone 1, Layer 3)
Total Avialble Space = 3 x 4 x 3 = 36 m3

Table-4 Construction Activities to Work Blocks Allocation

Activity Resource	**Activity**							
	A	B	C	D	E	F	G	H
Manpower & Equipment Block	1	1	2	2	3	2	2	2
Material Block	-	1	2	2	2	2	2	-

Figure 7.19 - Block Structure for Typical Floor

Activities competing for work space at any time period are prioritized using the heuristic ranking criteria given in Table-3 in Figure 7.17(c). During each time period (t_i), a set of activities is defined whose predecessors are all scheduled; this is called the ***Eligible Activity Set*** (EAS). At any time period (t_i), only the activities in EAS with their early start value (ES) less than or equal to the scheduling time (t_i) are considered for scheduling. These activities are ordered according to the ranking criteria specified. This ordered list of activities is referred to as the ***Ordered Scheduling Set*** (OSS). At the end of each scheduling cycle, once an activity is scheduled it is dropped from the EAS and the OSS. If other activities become eligible for scheduling they are added to the EAS. If any activity is not scheduled at any time period for any reason (e.g. lack of available space), it is dropped from the OSS but remain in the EAS to be considered for scheduling in the next possible time period. At each successive time period, a new rank ordering of all eligible activities is made and the process is continued until all activities have been scheduled.

The following is a detailed calculation of the scheduling of the example project.

Time t=0
EAS A
ES 0
LS 0
nor-dur 14
max-dur 14
OSS A

Schedule Activity A

• From Figure 7.18, space demand value for activity A is
SD-1 = (5x6) + (3x6) = 48 m3
SD-2 = 0

• From Figure 7.19, activity A is allocated to Block 1 with total space availability of 240 m3.

• Calculate preliminary start and finish dates for activity A. Since the work flow direction of A is upwards, start A at the first floor.

From Floor 1 Start 0 (week 0) Finish 14 (week 2)
Total Duration = 14(days) x 10(floors) = 140
To Floor 10 Start 126 (week 18) Finish 140 (week 20)

• From Figure 7.17(a), activity A has a class A space demand and a continuity class A. Activity must be scheduled continuously with no other activity in the same block already scheduled during the same time periods. Since no other activity is scheduled yet, then schedule activity A continuously to start at the first floor at time t=0 and finish at the tenth floor at time t=140 (week 20).

• Remove activity A from the ordered scheduled set (OSS) and the eligible activity set (EAS).

• Because activity A is scheduled, add succeeding activity B to the EAS.

• The status of the schedule is as shown in Figure 7.20.

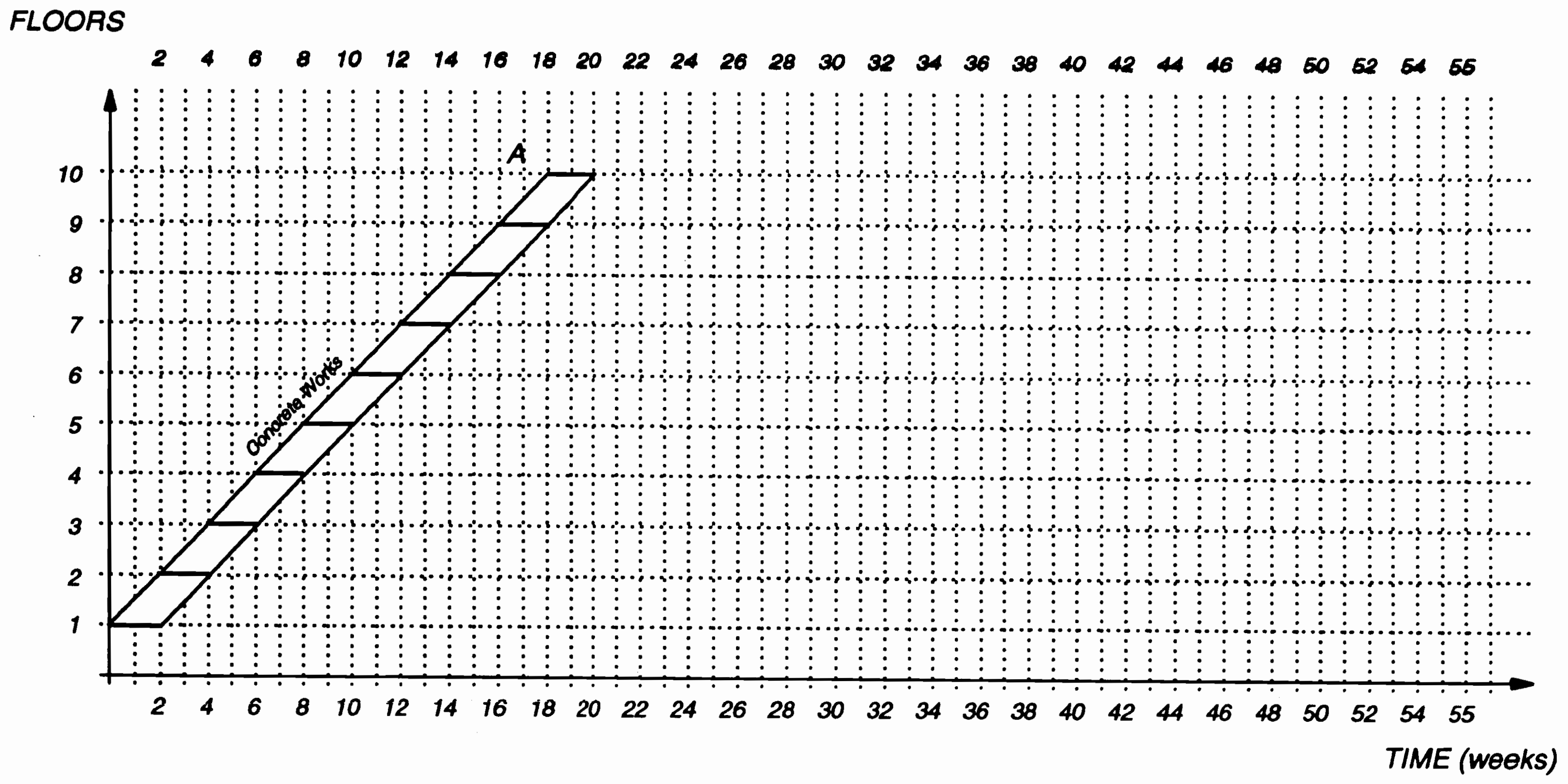


Figure 7.20 - Status of Schedule After A is Scheduled

Time	t=1
EAS	B
ES	14
LS	14
OSS	None - Activity B can not be scheduled since its early start value is bigger than t=1. Go to t=14.

Time	t=14
EAS	B
ES	14
LS	14
OSS	B

Schedule Activity B

• From Figure 7.18, space demand value for activity B is
SD-1 = (2x6) + (2x12) = 36 m3
SD-2 = 50x3 = 150 m3

• From Figure 7.19, activity B is allocated to Block 1 with total space availability of 240 m3.

• Calculate preliminary start and finish dates for activity B.

From Floor 1 Start 14 (week 2) Finish 21 (week 3)
Total Duration = 7(days) x 10(floors) = 70
To Floor 10 Start 77 (week 11) Finish 84 (week 12)

• From Figure 7.17(a), activity B has a class A space demand and must be continuous. Activity must be scheduled continuously with no other activity in the same block already scheduled during the same time period.

• Based on the preliminary dates, activity B can not be scheduled to start at t=14. This is because the production rate of B is faster than the preceding activity A. As a result, B will finish before A at higher floors thus violating the logical relationship defined by the floor network (i.e. A must be completed to start B at any floor).

• Because B must be scheduled continuously, by projecting B backwards from floor 10 the earliest time it can start at floor 1 without violating the logic of the network is at time t=77 (week 11). Modify early start (ES) value of B to 77.

• Remove activity B from the ordered scheduled set (OSS).

• At this point, the status of the schedule is still as shown in Figure 7.20.

Time	t=15
EAS	B
ES	77
LS	14
OSS	None - Activity B can not be scheduled since its early start value is bigger than t=15. Go to t=77.

Time	t=77
EAS	B
ES	77
LS	·14
OSS	B

Schedule Activity B

• Calculated space demand value for activity B is
SD-1 = 36 m3
SD-2 = 150 m3

• From Figure 7.19, activity B is allocated to Block 1 with total space availability of 240 m3.

• Calculate preliminary start and finish dates for activity B.

From Floor 1 Start 77 (week 11) Finish 84 (week 12)
Total Duration = 7(days) x 10(floors) = 70
To Floor 10 Start 140 (week 20) Finish 147 (week 21)

• From Figure 7.17(a), activity B has a class A space demand and must be continuous. Activity must be scheduled continuously with no other activity in the same block already scheduled during the same time periods. Since no other activity is scheduled in Block 1 during the same time period along all typical floors, then schedule activity B continuously to start at time t=77 (week 11) and finish at the tenth floor at time t=147 (week 21).

• Remove activity B from the ordered activity set (OSS) and the eligible activity set (EAS).

• Because activity B is scheduled, add succeeding activities C and D to the EAS. Because both activities can only start after activity B is completed at the first floor, modify early start (ES) value for both activities to 84 (week 12).

• At this point, the status of the schedule is as shown in Figure 7.21.

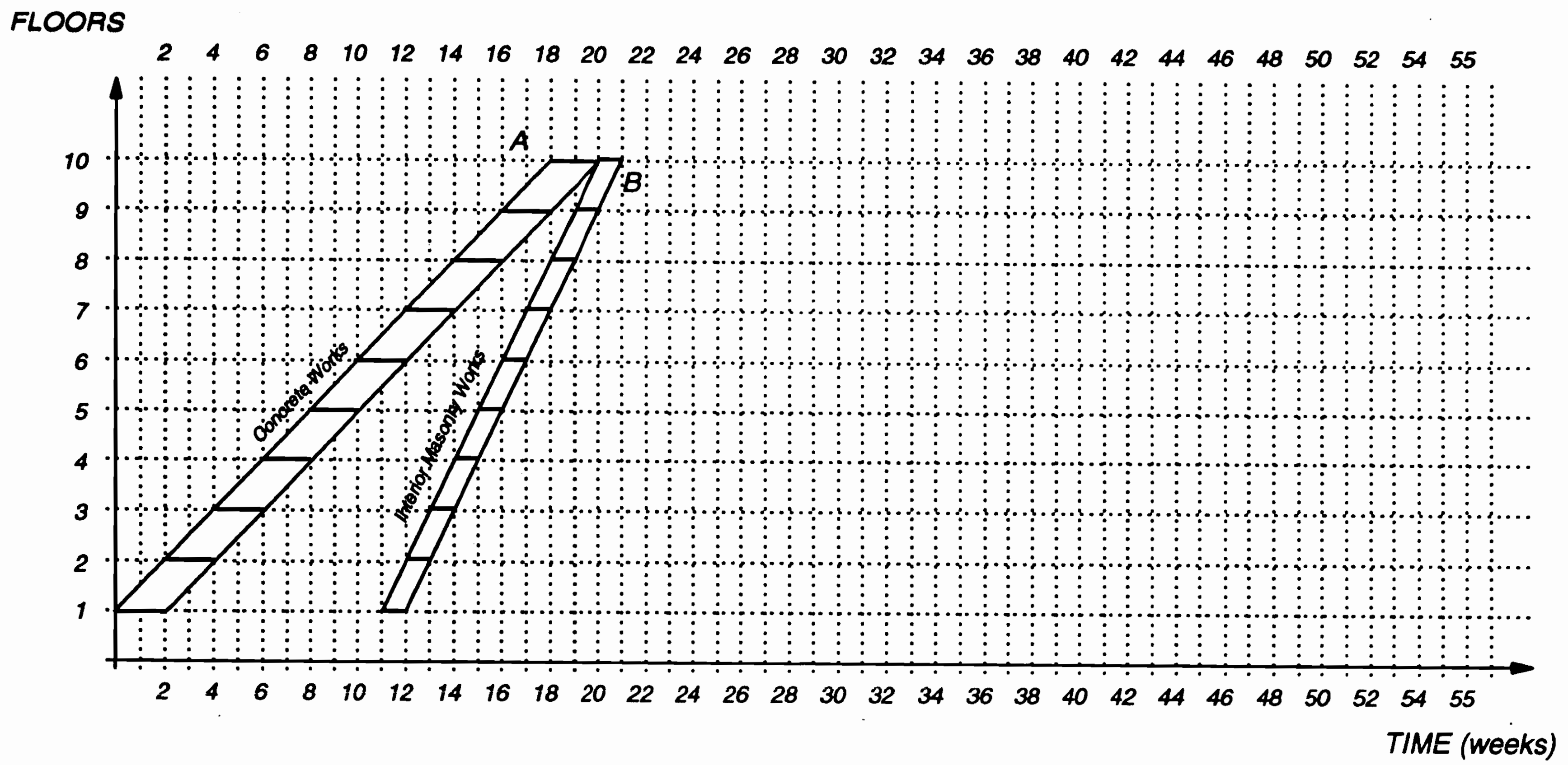


Figure 7.21 - Status of Schedule After B is Scheduled

Time	t=78
EAS	C D
ES	84 84
LS	28 21
OSS	None - no activities can be scheduled since early start values of all eligible activities are bigger than t=78. Go to t=84.

Time	t=84
EAS	C D
ES	84 84
LS	28 21
OSS	D C (Ranking is based on least late start value)

Schedule Activity D

• From Figure 7.18, space demand value for activity D is
SD-1 = (2x6) + (2x6) + (2x12) = 48 m3
SD-2 = (15x6) = 90 m3

• From Figure 7.17, activity D is allocated to Block 2 with total space availability of 204 m3.

• Calculate preliminary start and finish dates for activity D. Since the work flow direction of D is upwards, start D at the first floor.

From Floor 1 Start 84 (week 12) Finish 98 (week 14)
Total Duration = 14(days) x 10(floors) = 140
To Floor 10 Start 210 (week 30) Finish 224 (week 32)

• From Figure 7.17(a), activity D has a class A space demand and must be continuous. Since no other activity is already scheduled in Block 2 during the same time period, schedule activity D to start at the first floor at time t=84 (week 12) and finish at the tenth floor at time t=224 (week 32).

• Remove activity D from the ordered activity set (OSS) and the eligible activity set (EAS).

Schedule Activity C

• From Figure 7.18, space demand value for activity C is
SD-1 = (3x6) + (2x12) = 42 m3
SD-2 = (500x.05) = 25 m3

• From Figure 7.17, activity C is allocated to Block 2 with total space availability of 204 m3.

• Calculate preliminary start and finish dates for activity C. Since the work flow direction of C is upwards, start C at the first floor.

From Floor 1 Start 84 (week 12) Finish 94 (week 13.5)
Total Duration = 10(days) x 10(floors) = 100
To Floor 10 Start 179 (week 25.5) Finish 189 (week 27)

• Activity C can not be scheduled through the periods 84-189. This is because activity C is allocated to Block 2 where activity D is already scheduled. Preliminary dates calculated for C indicate that the activity will share block 2 with D at floors 1, 2 and 3. Activity D has a class A space demand and must solely occupy the work space. No other activities can be scheduled in the same block while D is in progress.

• To schedule activity C without interfering with D, the start time of C should be moved to day 130 (week 18.5). This will allow C to be scheduled continuously along all floors without sharing block 2 with D at any time. The start date is computed by projecting activity C backward from the tenth floor after D is completed at time t=224 (week 32).

• However, as indicated in Figure 7.17, activity C has a continuity class B. To start activity C earlier, the activity can be split along the typical floors. From Figure 7.17, maximum number of splits for activity C is 2. Hence, split the activity into two segments along the typical floors. Floors 1-5 will be executed in one segment and floors 6-10 will be executed in another segment. Splitting activity C will allow the first segment of the activity along floors 1-5 to start as early as day 114 (week 16.3) without sharing Block 2 with activity D at any floor between floors 1-5. Again, this time is calculated by projecting the first segment of activity C backward from the fifth floor after D is completed at time t=154 (week 22).

• Modify early start (ES) of C to 112 (week 16) and drop activity from OSS.

• At this point, the status of the schedule is as shown in Figure 7.22.

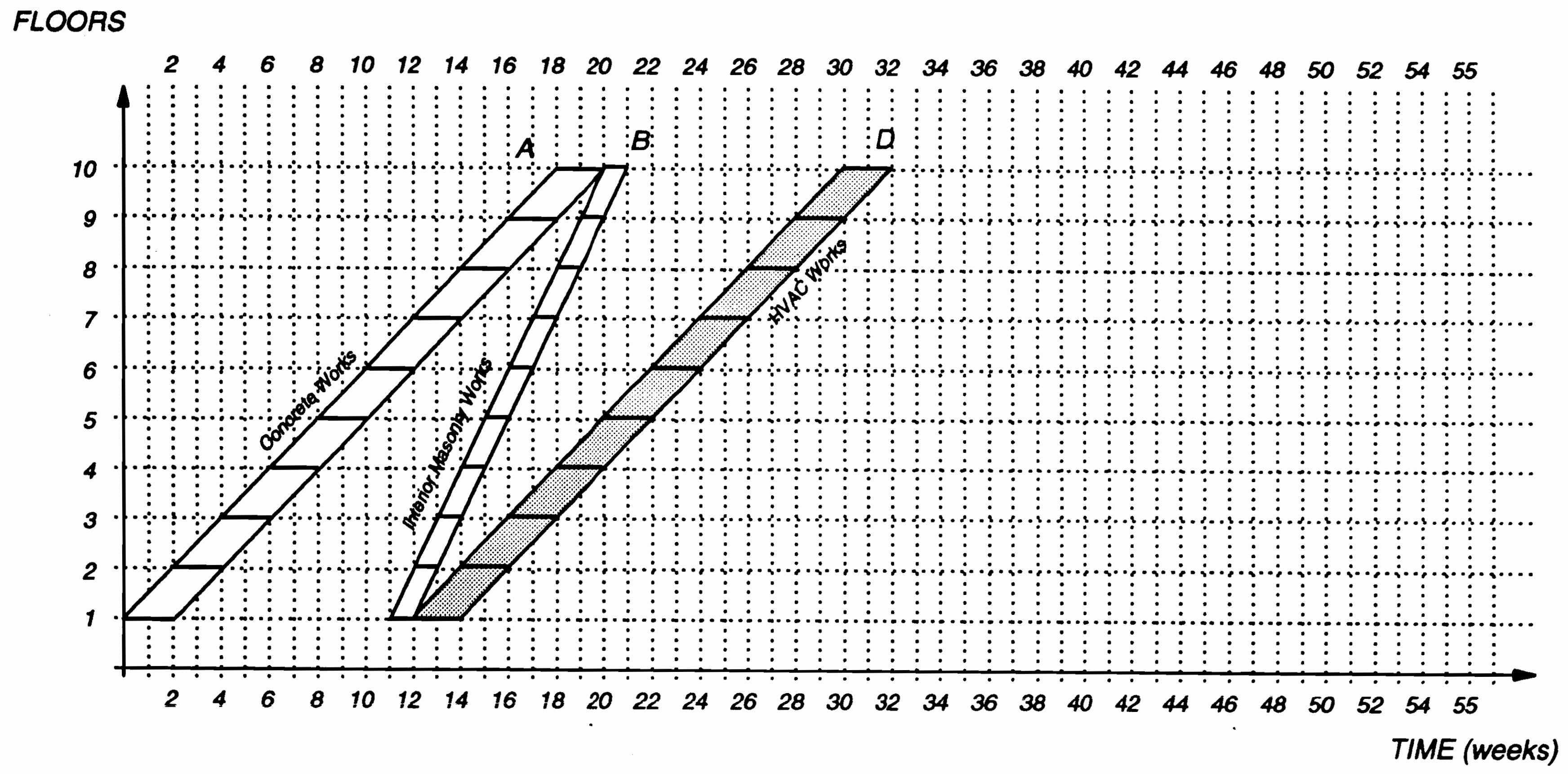


Figure 7.22 - Status of Schedule After D is Scheduled

Time	t=85
EAS	C
ES	114
LS	28
OSS	None - Activity C can not be scheduled since its early start value is bigger than t=85. Go to t=114.

Time	t=114
EAS	C
ES	112
LS	28
OSS	C

Schedule Activity C

• Calculated space demand value for activity C is
SD-1 = 42 m3
SD-2 = 25 m3

• From Figure 7.19, activity C is allocated to Block with total space availability of 204 m3.

• Calculate preliminary start and finish dates for first segment of activity C along floors 1-5.

From Floor 1 Start 112 (week 16) Finish 122 (week 17.5)
Total Duration = 10(days) x 5(floors) = 50
To Floor 5 Start 154 (week 22) Finish 164 (week 23.5)

• Activity C is allocated to Block 2 and 3. Since no other activity is already scheduled in Block 2 and 3 during the same time period along floors 1-5, available space exceeds the demand. Schedule the first segment of activity C to start at floor 1 at time t=114 (week 16.3) and finish at floor five at time t=164 (week 23.5).

• The second segment of C on floor 6 can only start after the completion of the activity on the fifth floor (i.e. at time t=164). Modify the early start (ES) value for activity C to 164.

• Remove C from the OSS.

• The status of the schedule at this point is as shown in Figure 7.23.

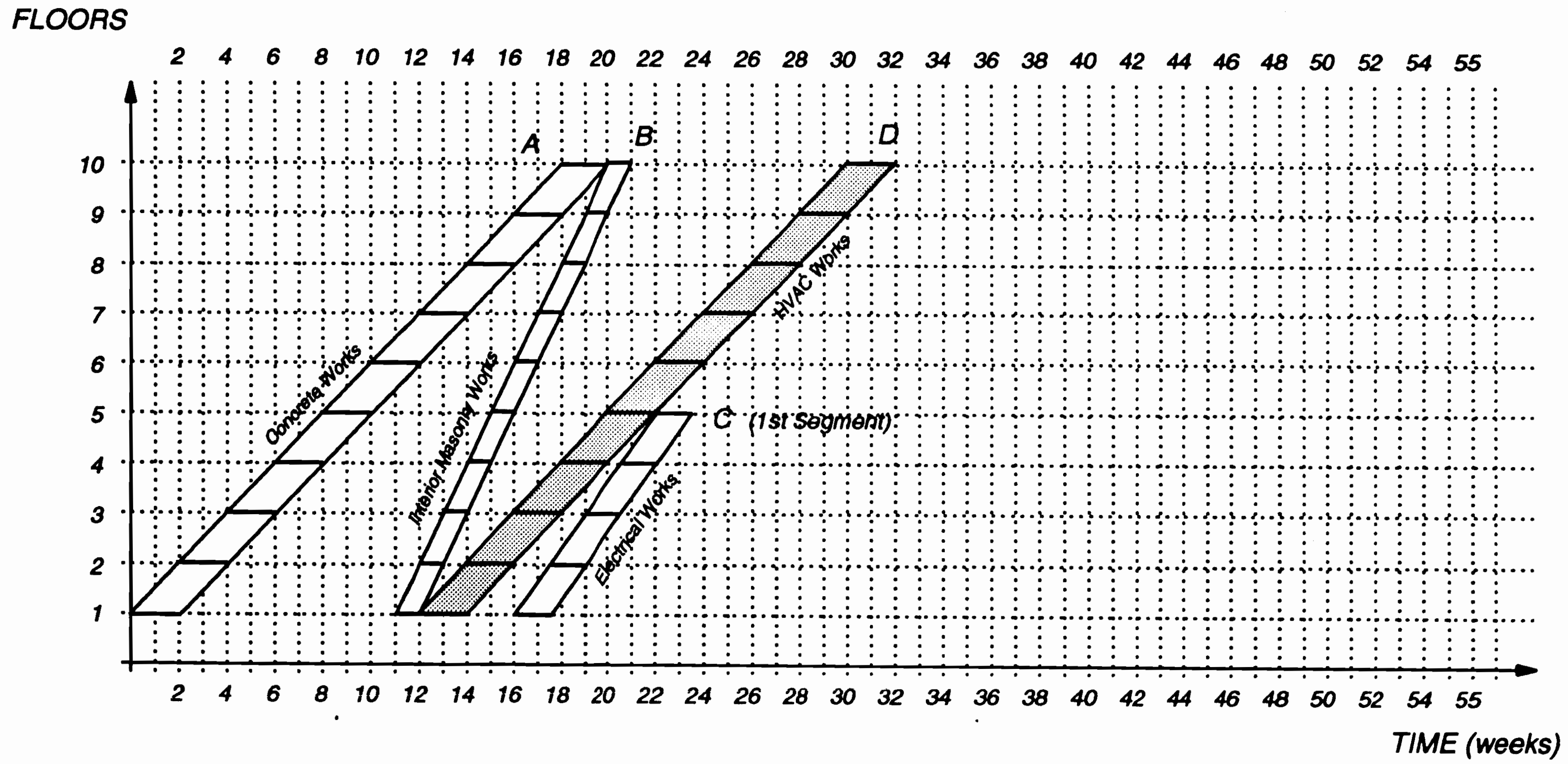


Figure 7.23 - Status of Schedule After 1st Segment of C is Scheduled

Time	t=114
EAS	C
ES	164
LS	28
OSS	None - Activity C can not be scheduled since its early start value is bigger than t=114. Go to t=164.

Time	t=164
EAS	C
ES	164
LS	28
OSS	C

Schedule Activity C

• Calculated space demand value for activity C is
SD-1 = 42 m3
SD-2 = 25 m3

• From Figure 7.19, activity C is allocated to Block 2 with total space availability of 204 m3.

• Calculate preliminary start and finish dates for second segment of activity C along floors 6-10.

From Floor 6 Start 164 (week 23.5) Finish 164 (week 25)
Total Duration = 10(days) x 5(floors) = 50
To Floor 10 Start 204 (week 29.5) Finish 214 (week 31.5)

• Activity C can not be scheduled through the periods 164-214. This is because activity C is allocated to Block 2 where activity D is already scheduled. Preliminary dates calculated for C indicate that the activity will share block 2 with D at floors 6 through 10. Activity D has a class A space demand and must solely occupy the work space. No other activities can be scheduled in the same block while D is in progress.

• To schedule the second segment of activity C without interfering with D, the start time of C should be moved to day 182 (week 26). This will allow the second segment of activity C to be scheduled along floors 6-10 without sharing block 2 with D at any time. The start date is computed by projecting the second segment of activity activity C backward from the tenth floor after D is completed at time t=224 (week 32).

• Modify early start (ES) value of activity C to 184 and remove C from the ordered scheduled set (OSS).

• The status of the schedule at this point is still as shown in Figure 7.23.

Time	t=165
EAS	C
ES	185
LS	28
OSS	None - Activity C can not be scheduled since its early start value is bigger than t=165. Go to t=184.

Time	t=184
EAS	C
ES	184
LS	28
OSS	C

Schedule Activity C

• Calculated space demand value for activity C is
SD-1 = 42 m3
SD-2 = 25 m3

• From Figure 7.19, activity C is allocated to Block 2 with total space availability of 204 m3.

• Calculate preliminary start and finish dates for second segment of activity C along floors 6-10.

From Floor 6 Start 184 (week 26) Finish 194 (week 27.7)
Total Duration = 10(days) x 5(floors) = 50
To Floor 10 Start 224 (week 32) Finish 234 (week 33.5)

• Activity C is allocated to Block 2. Since no other activity is already scheduled in Block 2 during the same time period along floors 6-10, available space exceeds the demand. Schedule the second segment of activity C to start at floor 6 at time t=184 (week 26.3) and finish at tenth floor at time t=234 (week 33.5).

• Remove C from the ordered scheduled set (OSS) and the eligible activity set (EAS).

• Since activity C is scheduled, and activity D is already scheduled, add activities E, F and G to the eligible activity set (EAS). Since these activities can only start after activities C and D are completed at the first floor, modify early start value for each activity to 122 (week 17.5) that corresponds to the completion of activity C at the first floor.

• Return the scheduling time counter to t=122 (week 17.5).

• The status of the schedule at this point is as shown in Figure 7.24.

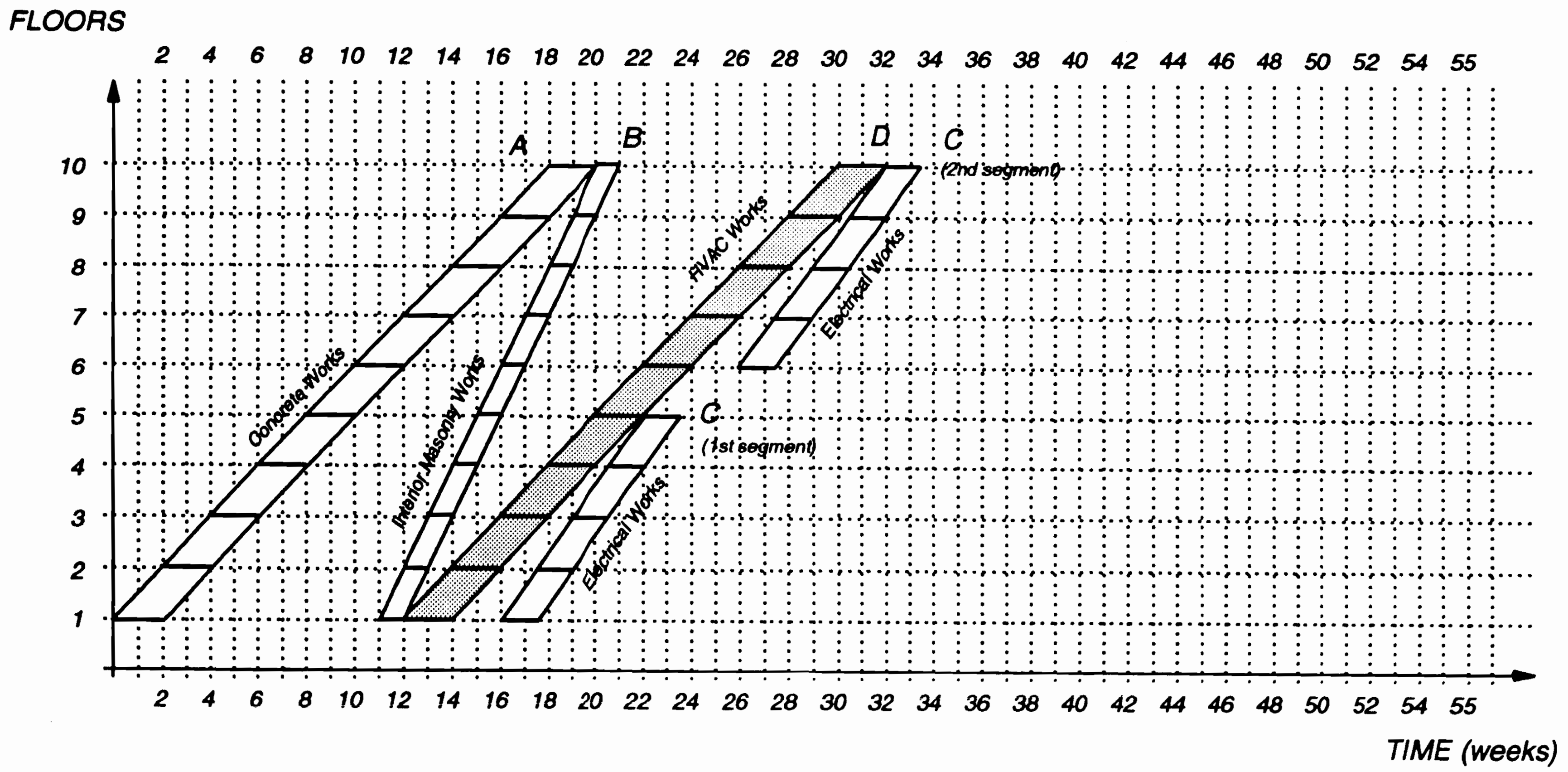


Figure 7.24 - Status of Schedule After C is Scheduled

Time t=122	
EAS	E F G
ES	122 122 122
LS	35 35 35
nor-dur	7 7 7
max-dur	7 7 7
OSS	F G E (Because all activities have equal late start values, normal duration values and maximum duration values, ranking is optional).

Schedule Activity F

• From Figure 7.18, space demand value for activity F is
SD-1 = (3x6) + (5x6) +(2x12) = 72 m3
SD-2 = 20x5 = 100 m3

• From Figure 7.19, activity F is allocated to Block 2 with total space availability of 204 m3.

• Calculate preliminary start and finish dates for activity F. Since the work flow direction of F is upwards, start F at the first floor.

From Floor 1 Start 122 (week 17.5) Finish 129 (week 18.5)
Total Duration = 7(days) x 10(floors) = 70
To Floor 10 Start 185 (week 26.5) Finish 192 (week 27.5)

• From Figure 7.17, activity F has a class B space demand and a continuity class A. F must be scheduled continuously along all floors if available space in Block 2 is sufficient. Based on the preliminary dates, activity F can not be scheduled through the periods 122-192. Because of the production of activity F, the activity will precede activities C and D at upper floors, thus violating the logic defined by the floor network.

• To schedule F continuously while respecting the precedence logic defined by the network in every floor, the activity start must be shifted to time t=171 (week 24.5). This earliest start date is computed by projecting activity F backward from the tenth floor after C is completed at time t=234 (week 33.5).

• Modify early start (ES) value of F to 171, and drop F from OSS.

Schedule Activity G

• From Figure 7.20, space demand value for activity G is
SD-1 = (2x6) + (3x6) = 30 m3
SD-2 = 3x3 = 9 m3

• From Figure 7.21, activity G is allocated to Block 2 with total space availability of 204 m3.

• Calculate preliminary start and finish dates for activity G. Since the work flow direction of G is upwards, start G at the first floor.

From Floor 1 Start 122 (week 17.5) Finish 129 (week 18.5)
Total Duration = 7(days) x 10(floors) = 70
To Floor 10 Start 185 (week 26.5) Finish 192 (week 27.5)

• From Figure 7.17, activity G has a class B space demand and a continuity class A. G must be scheduled continuously along all floors if available space in Block 2 is sufficient. Based on the preliminary dates, activity G can not be scheduled through the periods 122-192. Similar to activity F, because of the production rate of activity G, the activity will precede activities C and D at upper floors, thus violating the logic defined by the floor network.

• To schedule G continuously while respecting the precedence logic defined by the network in every floor, the activity start must be shifted to time t=171 (week 24.5). This earliest start date is computed by projecting activity G backward from the tenth floor after C is completed at time t=234 (week 33.5).

• Modify early start (ES) value of G to 171, and drop G from OSS.

Schedule Activity E

• From Figure 7.18, space demand value for activity E is
SD-1 = 2x6 = 12 m3
SD-2 = 20x3 = 60 m3

• From Figure 7.19, activity E is allocated to Block 2 and 3 with total space availability of 204 and 36 m3 respectively.

• Calculate preliminary start and finish dates for activity E. Since the work flow direction of E is upwards, start E at the first floor.

From Floor 1 Start 122 (week 17.5) Finish 129 (week 18.5)
Total Duration = 7(days) x 10(floors) = 70
To Floor 10 Start 185 (week 26.5) Finish 192 (week 27.5)

• From Figure 7.21, activity E has a class B space demand and a continuity class B. Based on the preliminary dates, activity E can not be scheduled continuously through the periods 122-192. Similar to activity F and G, because of the production rate of activity E, the activity will precede activities C and D at upper floors, thus violating the logic defined by the floor network.

• To schedule E continuously while respecting the precedence logic defined by the network in every floor, the activity start must be shifted to time t=171 (week 24.5). This earliest start date is computed by projecting activity E backward from the tenth floor after C is completed at time t=234 (week 33.5).

• However, as indicated in Figure 7.17, activity E has a continuity class B. To start activity E earlier, the activity can be split along the typical floors. From Figure 7.17, maximum number of splits for activity E is 1. Hence, split the activity into two segments along the typical floors. Floors 1-5 will be executed in one segment and floors 6-10 will be executed in another segment. Because of the production rate of E as compared to C and D, splitting activity E will allow the first segment of the activity along floors 1-5 to start as early as day 136 (week 19.5). Again, this time is calculated by projecting the first segment of activity E backward from the fifth floor after C is completed at time t=164 (week 23.5).

• Modify early start (ES) value of E to 136, and drop E from OSS.

• At this point, the status of the schedule is still as shown in Figuıe 7.24.

Time t=123
EAS E F G
ES 136 171 171
LS 35 35 35
nor-dur 7 7 7
max-dur 7 7 7
OSS none (no activities can be scheduled because ES values of all eligible activities are bigger than t=123. Go to t=136).

Time t=136
EAS E F G
ES 136 171 171
LS · 35 35 35
nor-dur 7 7 7
max-dur 7 7 7
OSS E (Only activity E can be scheduled because ES values of other eligible activities are bigger than t=136).

Schedule Activity E

• Calculated space demand value for activity E is
SD-1 = 12 m3
SD-2 = 60 m3

• From Figure 7.19, activity E is allocated to Block 2 and 3 with total space availability of 204 and 36 m3 respectively.

• Calculate preliminary start and finish dates for first segment of activity E along floors 1-5.

From Floor 1 Start 136 (week 19.5) Finish 143 (week 20.5)
Total Duration = 7(days) x 5(floors) = 35
To Floor 5 Start 164 (week 23.5) Finish 171 (week 24.5)

• Activity E is allocated to Block 2 and 3. Since no other activity is already scheduled in Block 2 and 3 during the same time period along floors 1-5, available space exceeds the demand. Schedule the first segment of activity E to start at floor 1 at time t=136 (week 19.5) and finish at floor five at time t=171 (week 24.5).

• The second segment of E on floor 6 can only start after the completion of the activity on the fifth floor (i.e. at time t=171). Modify the early start (ES) value for activity E to 171.

• Remove E from the OSS.

• The status of the schedule at this point is as shown in Figure 7.25.

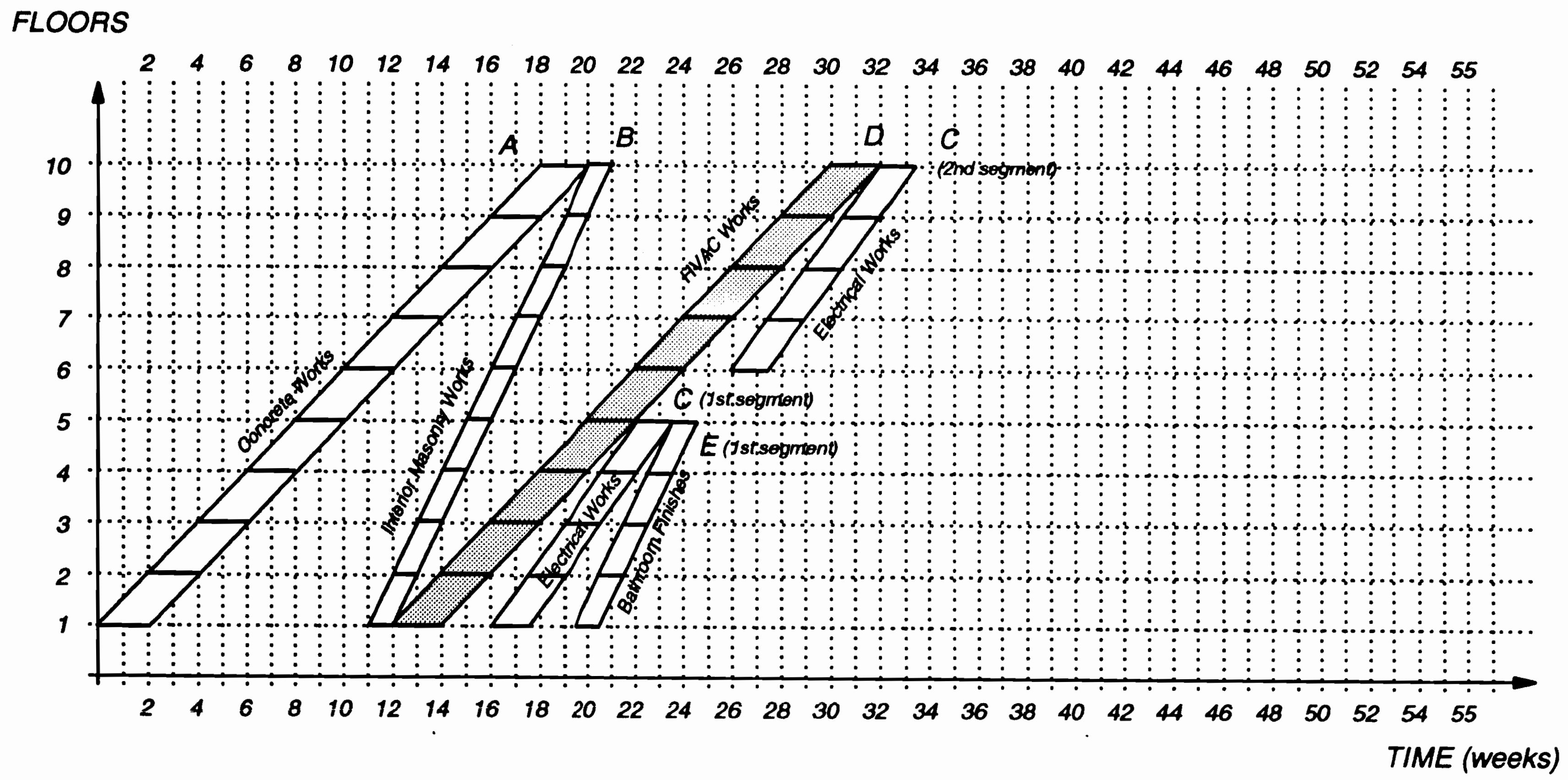


Figure 7.25 - Status of Schedule After 1st Sgment of E is Scheduled

Time t=137
EAS E F G
ES 171 171 171
LS 35 35 35
nor-dur 7 7 7
max-dur 7 7 7
OSS none (no activities can be scheduled because ES values of all eligible activities are bigger than t=137. Go to t=171).

Time t=171
EAS E F G
ES 171 171 171
LS 35 35 35
nor-dur 7 7 7
max-dur 7 7 7
OSS F G E(2nd segment) (Because all activities have equal late start and normal duration values, ranking is based on minimum max-dur).

Schedule Activity F

• Calculated space demand value for activity F is
SD-1 = 72 m3
SD-2 = 100 m3

• From Figure 7.19, activity F is allocated to Block 2 with total space availability of 204 m3.

• Calculate preliminary start and finish dates for activity F. Since the work flow direction of F is upwards, start F at the first floor.

From Floor 1 Start 171 (week 24.5) Finish 178 (week 25.5)
Total Duration = 7(days) x 10(floors) = 70
To Floor 10 Start 234 (week 33.5) Finish 241 (week 34.5)

• From Figure 7.17(a), activity F has a class B space demand and a continuity class A. Activity must be scheduled continuously with no other activity in the same block already scheduled during the same time periods. Since no other activity is scheduled in the block yet, then schedule activity F continuously to start at the first floor at time t=171 (week 24.5) and finish at the tenth floor at time t=241 (week 34.5).

• Remove activity F from the ordered scheduled set (OSS) and the eligible activity set (EAS).

Schedule Activity G

• Calculated space demand value for activity G is

SD-1 = 30 m3
SD-2 = 9 m3

• From Figure 7.19, activity G is allocated to Block 2 with total space availability of 204 m3.

• Calculate preliminary start and finish dates for activity G. Since the work flow direction of G is upwards, start G at the first floor.

From Floor 1 Start 171 (week 24.5) Finish 178 (week 25.5)
Total Duration = 7(days) x 10(floors) = 70
To Floor 10 Start 234 (week 33.5) Finish 241 (week 34.5)

• Preliminary start and finish dates for activity G indicate that G will share Block 2 with the already scheduled activity F along floors 1-10.

Total available space of Block 2 = 204 m3
Current usage (activity F) = 72 + 100 = 172 m3
Current available = 204 - 172 = 32 m3

• Available space is more than the required space by material for activity G but less than the total space demand of the activity. The scheduling procedure will attempt to schedule activity G with a reduced rate of production.

• Calculate SCF of G and attempt to schedule G with a lower production rate.

$$\text{SCF} = \frac{\text{Space Demand}}{\text{Current Space Available}}$$

$$= \frac{\text{Space Demand}}{\text{Total Space Available - Space Usage}}$$

$$= 30 / (204 - 172 - 9) \quad = 1.304$$

With reference to the SCF-Productivity Table-2 shown in Figure 7.17(b), a SCF of 1.3 implies reducing the production by 80%. With a 80% productivity, the required duration = 7 * 10/8 = 10.5 days.

• Schedule activity G to start at first floor at time t=172 (week 24.5) with a 50% productivity loss. Scheduled dates will be:

From Floor 1 Start 171 (week 24.5) Finish (171 + 10.5) = 181.5 (week 26)

Floor 2 Start 181.5 (week 26) Finish (181.5 + 10.5) = 192 (week 27.5)

After the second floor, activity G does not share Block 2 with F during the same time period. Hence, schedule activity G along floors 3-10 with the normal productivity rate (i.e. 100% production).

From Floor 3 Start 192 (week 27.5) Finish (192 + 7) = 199 (week 28.5)
Total Duration = 7(days) x 8(floors) = 65 days
To Floor 10 Start 241 (week 34.5) Finish (241 + 7) = 248 (week 35.5)

• Remove activity G from the ordered scheduled set (OSS) and the eligible activity set (EAS).

Schedule Activity E

• Calculated space demand value for activity E is
SD-1 = 12 m3
SD-2 = 60 m3

• From Figure 7.19, activity E is allocated to Block 2 and 3 with total space availability of 204 and 36 m3 respectively.

• Calculate preliminary start and finish dates for second segment of activity E along floors 6-10.

From Floor 6 Start 171 (week 24.5) Finish 178 (week 25.5)
Total Duration = 7(days) x 5(floors) = 35
To Floor 10 Start 199 (week 28.5) Finish 206 (week 29.5)

• Activity E can not be scheduled through the periods 171-206. Based on the calculated preliminary dates, activity E will precede activities C and D at upper floors. This will violate the logic defined by the floor network.

• To schedule the second segment of activity E while respecting the precedence logic defined by the floor network, the activity must be shifted to start at time t=206 (week 29.5). This earliest start date is computed by projecting the second segment of E backwards from the tenth floor after activity C is completed at time t=234 (week 33.5).

• Modify early start (ES) value of activity E to 206, and remove activity from OSS.

• The status of the schedule at this point is as shown in Figure 7.26.

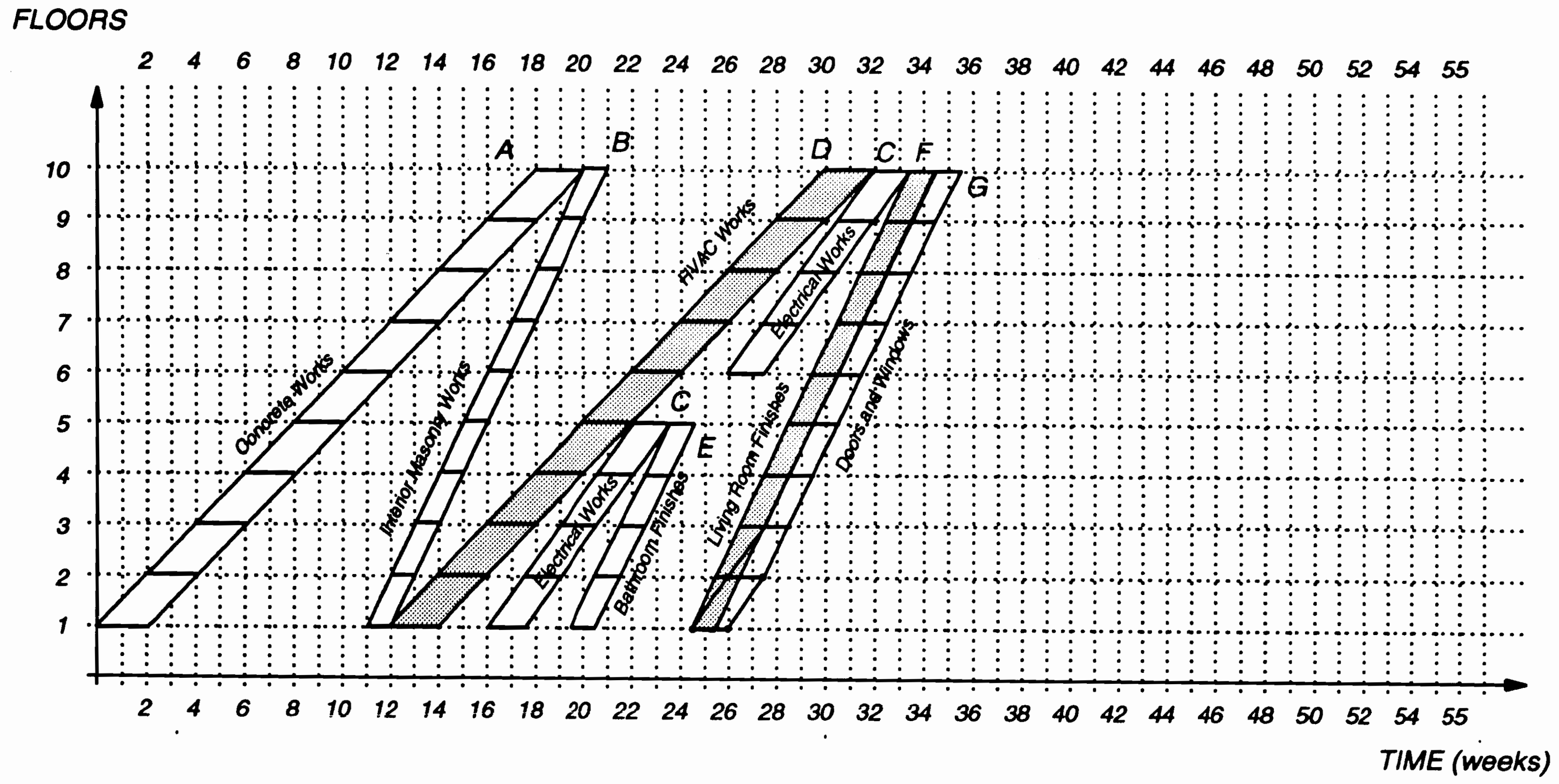


Figure 7.26 - Status of Schedule After F and G are Scheduled

Time	t=173
EAS	E
ES	206
LS	35
nor-dur	7
max-dur	7
OSS	none - activity E can not be scheduled because ES value of activity is bigger than t=173. Go to t=206.

Time	t=206
EAS	E
ES	206
LS	35
nor-dur	7
max-dur	7
OSS	E

Schedule Activity E

• Calculated space demand value for activity E is
SD-1 = 12 m3 and SD-2 = 60 m3

• From Figure 7.19, activity E is allocated to Block 2 and 3 with total space availability of 204 and 36 m3, respectively.

• Calculate preliminary start and finish dates for second segment of activity E along floors 6-10.

From Floor 6 Start 206 (week 29.5) Finish 213 (week 30.5)
Total Duration = 7(days) x 5(floors) = 35
To Floor 10 Start 234 (week 33.5) Finish 241 (week 34.5)

• Preliminary dates indicate that second segment of activity E will share Block 2 with the already scheduled activity F along floors 1-10.

Total available space of Block 2 = 204 m3
Current usage (activity F) = 72 + 100 = 172 m3
Current available = 204 - 172 = 32 m3

• Available space is less than the required space to store material for activity E. The activity can not be scheduled to share the block with activity F. The scheduling procedure will attempt to shift the start date of the second segment of activity E to a later date.

• Since activity E has the same production rate as activity F, move activity E to start after the completion of F at the sixth floor. This will insure that activity E will always start after F along floor 6-10.

• Modify the early start (ES) value of E to 213 (week 30.5) and drop E from the OSS.

• At this point, the status of the schedule is still as shown in Figure 7.26.

Time	t=207
EAS	E
ES	213
LS	35
nor-dur	7
max-dur	7
OSS	none - activity E can not be scheduled because ES value of activity is bigger than t=203. Go to t=213.

Time	t=213
EAS	E
ES	213
LS	35
nor-dur	7
max-dur	7
OSS	E

Schedule Activity E

• Calculated space demand value for activity E is

SD-1 = 12 m3
SD-2 = 60 m3

• From Figure 7.19, activity E is allocated to Block 2 (material) and 3 (manpower/equipment) with total space availability of 204 and 36 m3, respectively.

• Calculate preliminary start and finish dates for second segment of activity E along floors 6-10.

From Floor 6 Start 213 (week 30.5) Finish 220 (week 31.5)
Total Duration = 7(days) x 5(floors) = 35
To Floor 10 Start 241 (week 34.5) Finish 248 (week 35.5)

• Preliminary start and finish dates for activity E indicate that E will share Block 2 with the already scheduled activity G along floors 6-10.

Total available space of Block 2 = 204 m3
Current usage (activity G) = 30 + 9 = 39 m3
Current available = 204 - 39 = 165 m3

• Available space is more than the required space by material for activity E (SD_2 = 60 < 165). Since available space in Block space is sufficient for space demand of manpower and equipment (SD-1 = 12 < 36), then schedule activity E to start at floor 6 at time t=213 (week 30.5) and finish at floor 10 at time t=248 (week 35.5)

• Remove activity E from the ordered scheduled set (OSS) and the eligible activity set (EAS).

• Since activity E is scheduled, and activities F and G are already scheduled, add succeeding activity H to the eligible activity set (EAS). Since activity H has a downward workflow direction, it can only start at the tenth floor after activities F, G, and E are completed. Modify early start (ES) value of H to 248 (week 35.5).

• At this point, the status of the schedule is as shown in Figure 7.27.

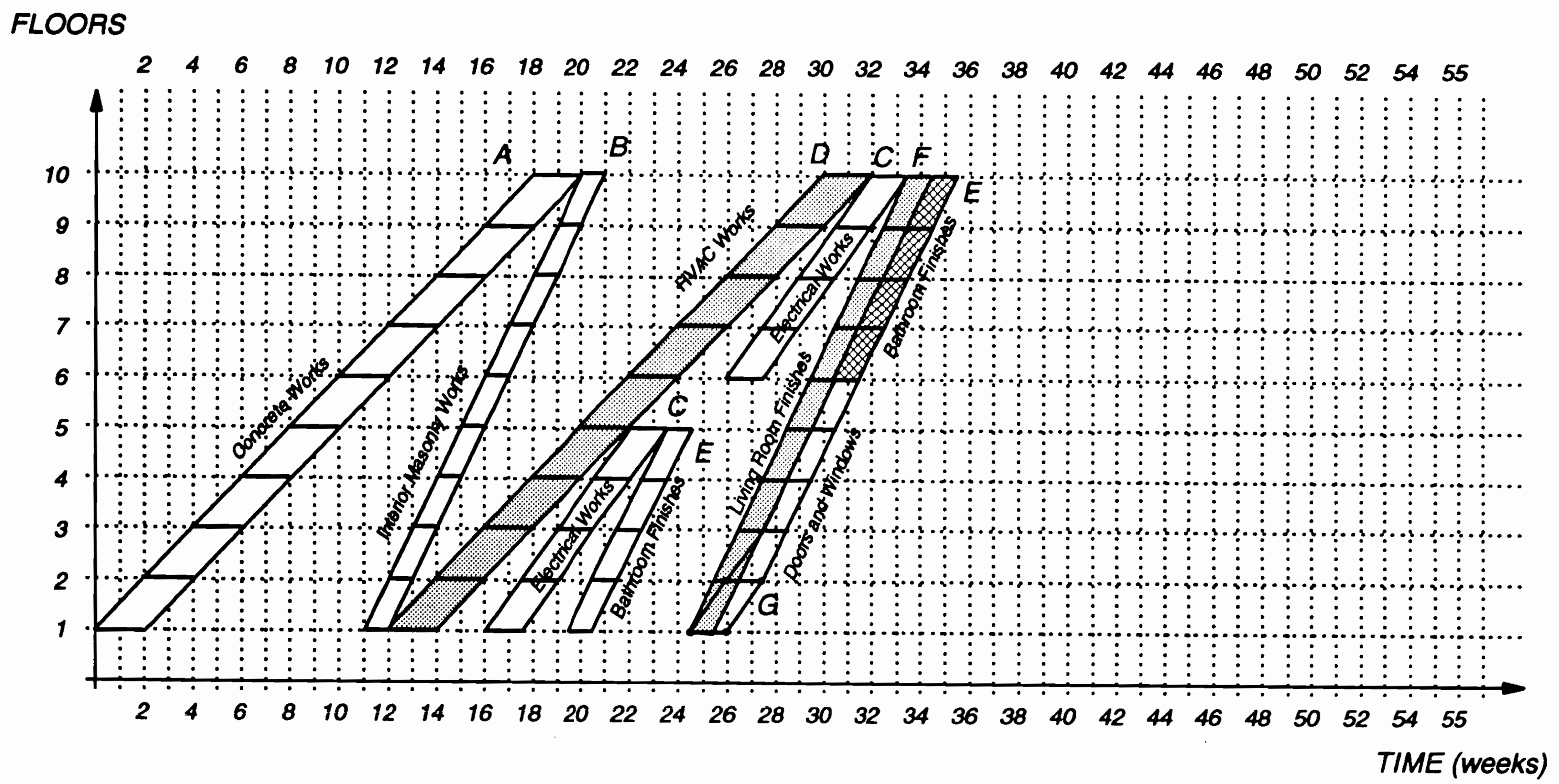


Figure 7.27 - Status of Schedule After E is Scheduled

Time	t=214
EAS	H
ES	248
LS	42
nor-dur	7
max-dur	7
OSS	none - activity H can not be scheduled because ES value of activity is bigger than t=214. Go to t=248.

Time	t=248
EAS	H
ES	248
LS	42
nor-dur	7
max-dur	7
OSS	H

Schedule Activity H

• From Figure 7.18, the space demand value for activity H is

SD-1 = 2x6 = 12 m3
SD-2 = 0

• From Figure 7.19, activity H is allocated to Block 2 and 3 with total space availability of 204 and 36 m3, respectively.

• Calculate preliminary start and finish dates for activity H along floors 1-10. Since workflow direction of activity H is downward, start H at the tenth floor.

From Floor 10 Start 248 (week 35.5) Finish 255 (week 36.5)
Total Duration = 7(days) x 10(floors) = 70
To Floor 10 Start 311 (week 44.5) Finish 318 (week 45.5)

• From Figure 7.17(a), activity H has a class B space demand and a continuity class B. Total available space for activity exceeds the demand. Since there is no need to split the activity, then schedule activity H continuously to start at the tenth floor at time t=248 (week 35.5) and finish at the first floor at time t=318 (week 45.5).

• Remove activity H from the ordered scheduled set (OSS) and the eligible activity set (EAS).

• All activities are scheduled and the schedule generation process is completed. The final status of the schedule is as shown in Figure 7.28.

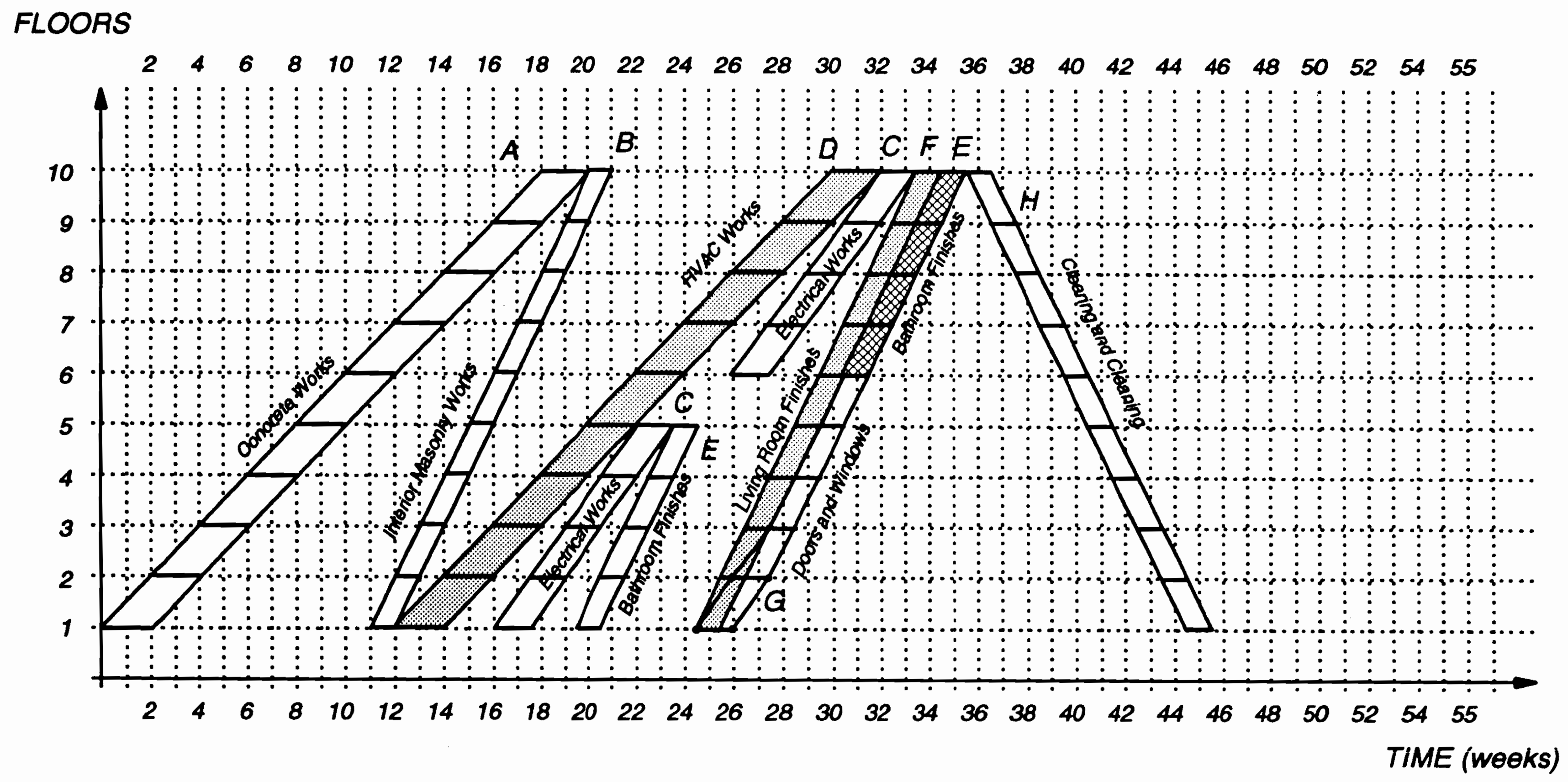


Figure 7.28 - Final Status of Schedule After H is Scheduled

7.6 Chapter Summary

This chapter has introduced the basic concepts and theory of space-based scheduling. A space-based scheduling module is presented. The model constitutes a procedure to quantify work space parameters (i.e. demand and availability) and to use these parameters in making scheduling decisions to schedule each activity.

The proposed scheduling model was presented as follows:

1- Modeling of Work Space Demand

2- Modeling of Work Space Availability

3- Space-Based Scheduling Factors and Decisions

The chapter also provided a worked scheduling example to demonstrate the implementation concepts of the proposed model. The example manually generates a schedule for a 10-floor story building by considering work space constraints only.

The space-based model is integrated with other scheduling procedures in the implementation of the Scar scheduling system. The model and the overall Scar system are discussed in detail in the following chapters.

8.0 The SCaRC SCHEDULING SYSTEM

The **SCaRC** (Space Constrained and Resource Constrained) scheduling system is developed to generate construction schedules for the repetitive floors of multi-story buildings. The system utilizes resource constraints and space constraints for the production of the schedule. The system also considers other issues associated with scheduling repetitive work. Such issues include: horizontal logic constraints, vertical logic constraints, continuity issues, and variable production rates. The developed system is implemented using a knowledge-based approach.

This chapter presents an overall description of the SCaRC system. The first part of the chapter presents the schedule generation approach adopted by the system. The scheduling constraints considered in developing a schedule are first discussed. The different scheduling parameters that influence and control the verification of these constraints are presented. The second part of the chapter describes in general the different components of the SCaRC system and the basic function of each component. Chapters 9 through 12 discuss these components and their respective functions in more detail.

8.1 Scheduling Constraints

The scheduling system generates a schedule by verifying several scheduling constraints and making scheduling decisions regarding each constraint. There are four major scheduling constraints considered by the model to generate a schedule. These constraints are:

1- Horizonal logic constraints

2- Vertical logic constraints

3- Resource constraints

4- Work space constraints

During the verification of these constraints, continuity issues are considered along with redefining activity duration to comply with the dominant activity pace.

Horizontal logic constraints are constraints defined by the logical sequence among activities of the typical floor network. Based on this logic, an activity can not be scheduled unless all its preceding activities of the same floor are scheduled. Also, non of the succeeding activities to the activity can be scheduled unless the activity is scheduled. Because this type of constraints are confined to activities of the same floor, it is termed horizontal logic constraints.

Horizontal logic constraints are verified at each floor due to the problem of the dominant activity production rate. As was discussed in Chapter 5, activities with a big duration value (i.e. slow production rate) tend to control the pace over activities with a faster rate. Verifying horizontal constraints allows to insure that at any floor the scheduled activity complies with the defined sequence of its preceding slower activity activities.

Vertical logic constraints are a result of logical links between activities of different floors. These links may be due to different reasons such as: constructability issues, weather conditions, safety and code regulations, etc. The objective of verifying this type of constraint is to insure that, at any floor, the scheduled activity comply with the vertical logic forced by an activity on a different floor.

Resource constraints are based on demand versus availability values for project resources. If the total resource demand of all activities scheduled during a specific time period exceeds the availability, a conflict occurs. Activities must be rescheduled to insure that the demand does not exceed the availability for any resource.

Similar to resource constraints, *space constraints* are based on demand versus availability values for work space. The total work space demand for any group of activities scheduled concurrently must not exceed the space availability value of the work area during any time period.

8.2 Scheduling Parameters

Verification of the scheduling constraints is dependent on several scheduling parameters associated with each activity. These parameters control the scheduling decisions made during the verification of each constraint. The parameters are either defined by user or computed by the scheduling system.

This section introduces the different scheduling parameters that influences the scheduling of each activity. As illustrated in Figure 8.1, these parameters are grouped into four main categories:

1- General Scheduling Parameters.

2- Special Scheduling Parameters

3- Resource Parameters.

4- Work Space Parameters.

General Scheduling Parameters - These are general activity parameters associated with basic scheduling procedures. The parameters are defined by the user during data

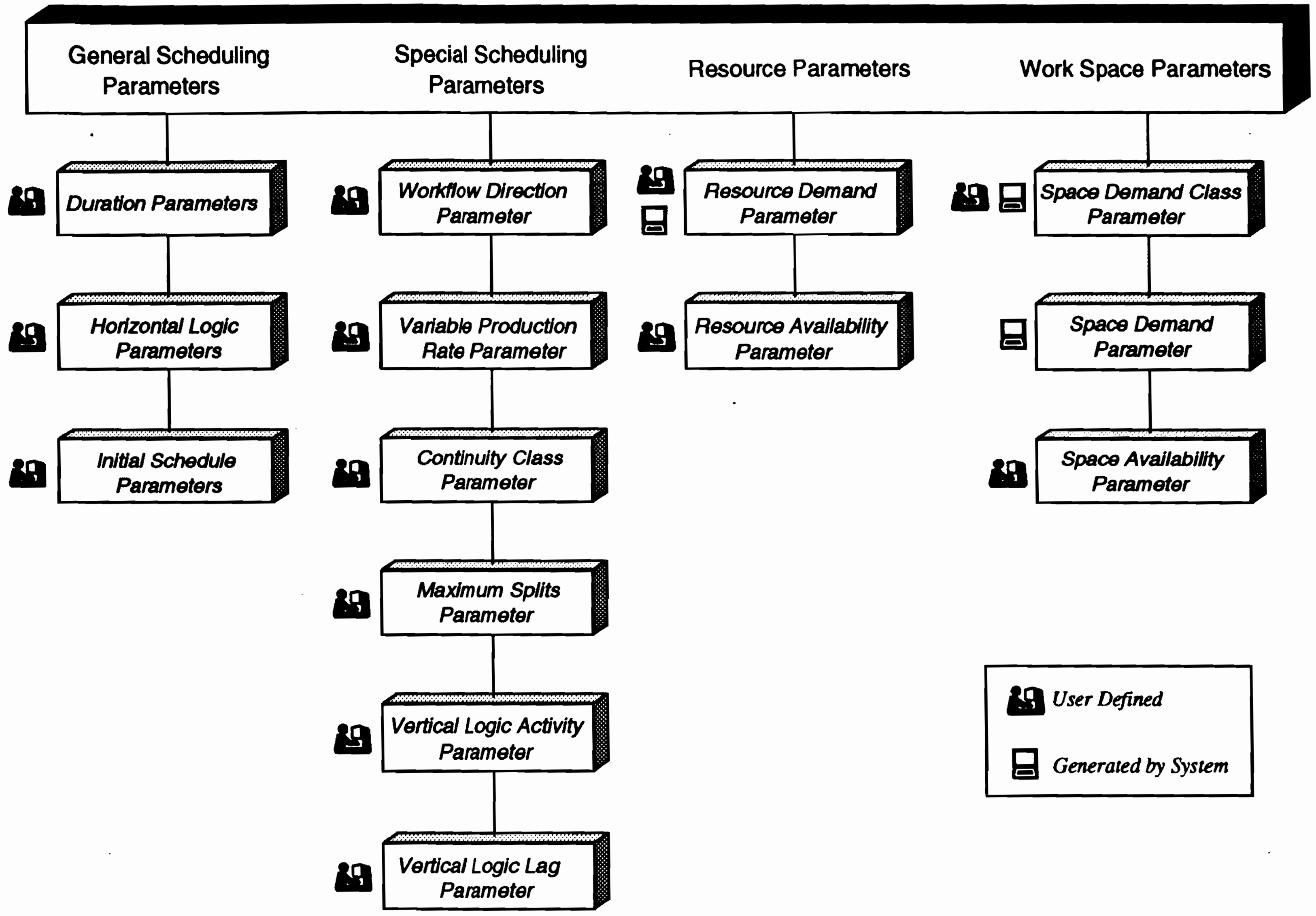

General Scheduling Parameters
Special Scheduling Parameters
Resource Parameters
Work Space Parameters
Duration Parameters
Horizontal Logic Parameters
Initial Schedule Parameters
Workflow Direction Parameter
Variable Production Rate Parameter
Continuity Class Parameter
Maximum Splits Parameter
Vertical Logic Activity Parameter
Vertical Logic Lag Parameter
Resource Demand Parameter
Resource Availability Parameter
Space Demand Class Parameter
Space Demand Parameter
Space Availability Parameter
User Defined
Generated by System

Figure 8.1 - Scheduling Parameters

input. The parameters are based on a logical network and an initial schedule for a typical floor. These parameters are

1- Duration Parameters

Two duration parameters are associated with each activity; a *normal duration* parameter and

a *maximum duration* parameter. The value of the normal duration is associated with normal production rates and reflects the duration required to perform the activity under normal construction conditions.

The value of the maximum duration is associated with the minimum possible production rate of the activity and reflects the maximum feasible time in which the activity can be executed. The maximum duration parameter is important as it influences the scheduling decision process during the generation of the schedule. During activity scheduling, decisions may be made to increase the activity duration to satisfy a specific constraint. These decisions are controlled by the maximum duration limit established by this parameter.

2- Horizontal Logic Parameters

These parameters define the logical sequence among the activities in each typical floor. The parameters are established by the typical floor network. The logical sequence defined in the network is based on hard logic only (i.e. technological constraints) with no consideration to any other constraints.

Two logical parameters are defined; *preceding activity parameter* and *succeeding activity parameter*. The value of the preceding activity parameter defines the set of activities that must be scheduled in the floor before the current activity can be considered for scheduling. The value of the succeeding activity parameter defines the set of activities that become eligible for scheduling once the current activity is scheduled along all typical floors.

3- Initial Schedule Parameters

These parameters are defined by an initial schedule for a typical floor. Two parameters are utilized; an *early start (ES) parameter* and a *late start (LS) parameter*.
The early start (ES) parameter defines the earliest time to start an activity at the first typical floor. In addition to the preceding activity parameter, this parameter defines the activities eligible for scheduling at any time period. At any time t_i, an activity becomes eligible for scheduling if all its preceding activities are scheduled along all floors and its early start (ES) value is less than or equal to t_i.

The late start (LS) parameter is utilized for the heuristic prioritization of activities competing for limited resource or space availabilities. The parameter constitutes one of the criteria used for ranking the competing activities. Heuristic prioritization is discussed in section 8.3.2.

Special Scheduling Parameters - These are special activity parameters associated with scheduling repetitive work in multi-story buildings. Similar to the general scheduling parameters, these special parameters are defined by the user during data input. There are five special parameters:

1- Workflow Direction Parameter

This parameter specifies the direction of work flow for the activity. The parameter controls the start floor number to schedule each activity. Activities with an upward workflow direction are scheduled to start at the first typical floor and proceed upwards. Activities with a downward workflow direction are scheduled to start at the last typical floor and proceed downwards.

2- Variable Production Rate Parameter

This parameter specifies whether the activity production rate suffers a loss as a result of the increased travel time effect and learning curve phenomena associated with repetitive work in multi-story projects.

3- Continuity Class Parameter

This parameter controls the continuity status of the activity during scheduling. This parameter defines two categories or classes of continuity for each activity:

Class A: Continuous activity - splitting not allowed; and
Class B: Intermittent activity - splitting allowed.

Class A activities must be scheduled continuously along all the typical floors with no interruptions or stoppage. Activities of this class may include activities of the major trades such as: construction of the skeleton activities, exterior cladding or masonry, sheetrock installation, etc.

Class B activities constitute those activities that can be split along the typical floors. Splitting may be required for the following reasons:

1- Comply with the pace of the dominant activity. As discussed in chapter 5, if the activity has a production rate faster than the preceding activities, the activity may be split several times along the floors.

2- Satisfy limited resource availability. If resources can not be made available for the activity along all the floors, the activity can be split into several segments along the entire floors. Each segment can be scheduled during time periods when the required resources can be made available.

3- Satisfy limited space availability. Similar to the limited resource availability condition, if space can not be made available for the activity along all the floors, the activity may be split into several segments. Each segment is scheduled during time periods when space is found available.

The number of times an activity of class B can be split along the typical floors (i.e. number of possible activity segments) is controlled by a maximum value. This value is defined by another parameter (maximum splits) that is associated with this class of activities.

4- Maximum Splits Parameter

This parameter defines the maximum number of times an activity can be split along the typical floors. The parameter is defined for activities with a B continuity class. For example, if the maximum number of splits is equal to 3, then the activity can only be split into a maximum of three segments along the entire typical floors. To schedule the activity floor by floor, the value of this parameter could be set equal to the total number of typical floors. Activities with continuity class A will have a zero value for this parameter.

5- Vertical Logic Activity Parameter

This parameter defines the activity number forcing the vertical logic on the current activity. The parameter allows to define one vertical logic activity for each activity.

6- Vertical Logic Lag Parameter

This parameter defines the vertical buffer (i.e. no of floor) between the vertical logic activity forcing the constraint and the current activity.

Activity Resource Parameters - Resource parameters define the resource demand and availability values for each resource type. These parameters control the scheduling decisions when resource constraints are verified. Two parameters are defined: resource demand parameter and resource availability parameter.

1- <u>Resource Demand Parameter</u>

For each resource type required for the execution of each activity,, a demand value will be defined. The demand value for each resource is initially defined by the user during data entry. The user defined value is based on activity normal duration (i.e. normal production rate). During the scheduling process, if the activity duration is modified by the scheduling procedure to satisfy a specific constraint, demand values for all allocated resources are adjusted, if possible, to reflect the new duration.

2- <u>Resource Availability Parameter</u>

For each resource type required in the project, an availability value is defined. This value determines the maximum daily quantity of the resource that can be made available for the project. Resource availability values are defined by the user during data input.

Work Space Parameter - Space parameters control the scheduling decisions during the verification of space constraints. Three parameters are defined:

1- A Space Demand Class Parameter

2- A Space Demand Parameter

3- A Space Availability Parameter

The space demand class parameter defines the demand class for each activity. The parameter is defined by the user during data input. As discussed in chapter 7, three categories or classes are defined: A, B or C.

The other two parameters define the space demand and availability values. The space demand parameter defines the work space value required by any activity of the project. The space availability parameter defines the total available work space of each work block.

8.3 Schedule Generation Approach

The production of the schedule is based on verifying the specified scheduling constraints and is controlled by the defined parameters for each activity. This process is executed in two major steps. The first step is a *Pre-Scheduling Step* and consists of different procedures to compute and define all required scheduling data. All the procedures for data preparation defined in this first step are performed within a database environment.

The second step of the schedule generation process is a *Scheduling Step* and consists of the actual production of the schedule. A knowledge-based scheduling system is responsible for this step and generates a schedule in two consecutive scheduling stages. In these two stages, horizontal and vertical constraints, resource constraints, and space constraints are verified and checked for each activity. This is accomplished through a series of iterations between these two stages until all constraints are satisfied. The process is repeated for each activity until all activities are scheduled and the overall schedule is established.

Figure 8.2 depicts the schedule generation steps. These steps are discussed in more detail in the following sections.

8.2.1 STEP-1: Pre-scheduling Preparation Step

The objective of this step is to identify the general characteristics of the project and specify all the construction parameters and data necessary to produce the schedule. The following is accomplished in this step:

a) Define Work Space Availability

1- Define the zone and layer block structure of the typical floor. As discussed in chapter 7, the construction floor work area is divided into a number of zone and layer blocks.

2- Using a CAD model of the floor, design elements and their geometric data are used to compute the available work space of each block. This is accomplished using a geometric algorithm.

b) Define Work Space Demand

1- Space demand values for generic construction resources are stored in a global space demand data file in the database.

2- Based on the resource requirements for the project, the space demand values

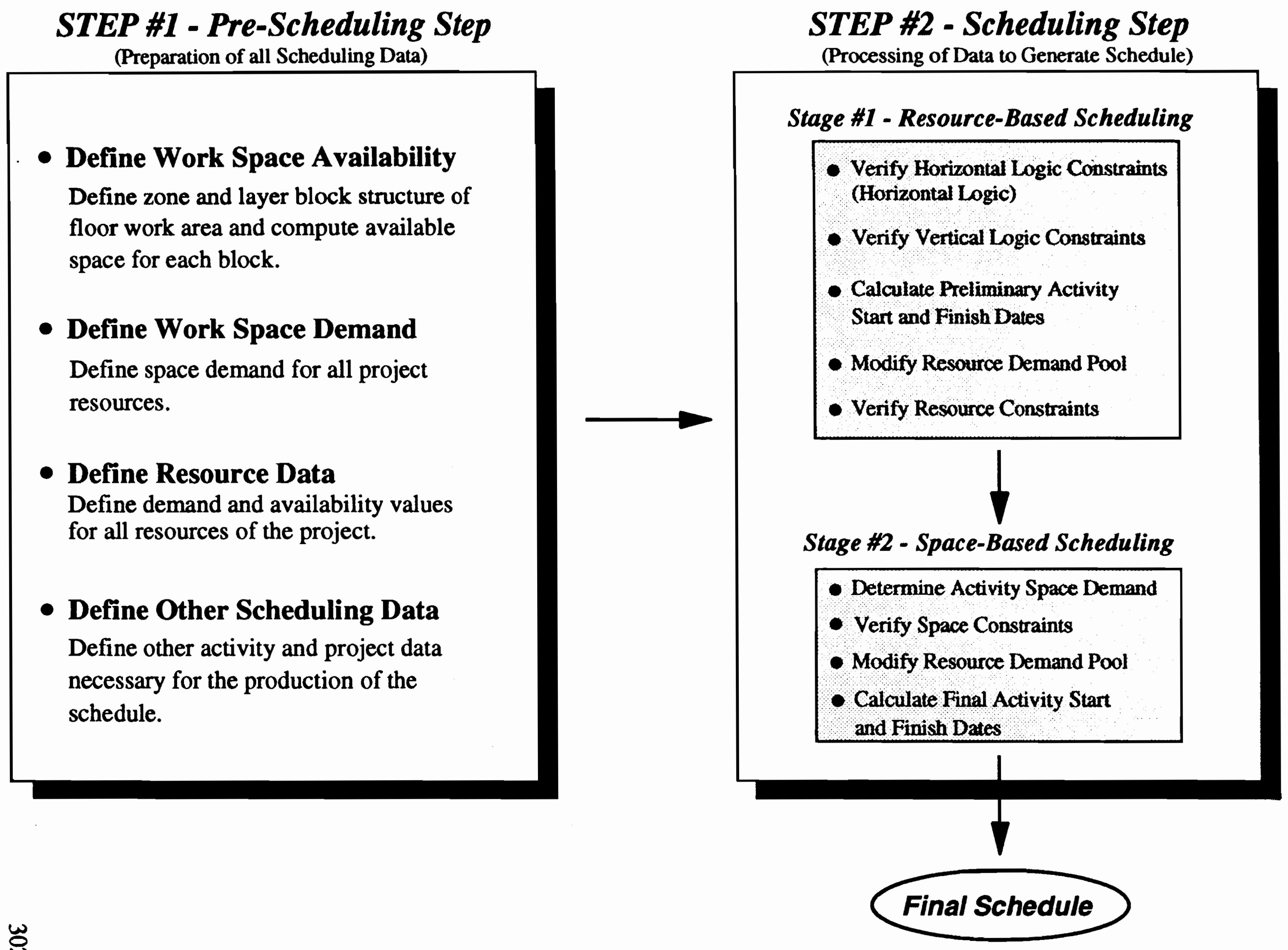

STEP #1 - Pre-Scheduling Step
(Preparation of all Scheduling Data)
• Define Work Space Availability
Define zone and layer block structure of floor work area and compute available space for each block.
• Define Work Space Demand
Define space demand for all project resources.
• Define Resource Data
Define demand and availability values for all resources of the project.
• Define Other Scheduling Data
Define other activity and project data necessary for the production of the schedule.
STEP #2 - Scheduling Step
(Processing of Data to Generate Schedule)
Stage #1 - Resource-Based Scheduling
• Verify Horizontal Logic Constraints (Horizontal Logic)
• Verify Vertical Logic Constraints
• Calculate Preliminary Activity Start and Finish Dates
• Modify Resource Demand Pool
• Verify Resource Constraints
Stage #2 - Space-Based Scheduling
• Determine Activity Space Demand
• Verify Space Constraints
• Modify Resource Demand Pool
• Calculate Final Activity Start and Finish Dates
Final Schedule

Figure 8.2 - Overall Schedule Generation Steps

for each resource are extracted from the global file and stored in a project specific data file. This is accomplished by another algorithm that interacts with the different database files.

3- Using the extracted global demand values, the user estimates space demand for each resource.

c) Define Resource Data

1- Define resource demand pools for each activity.

2- Define resource availability values for each resource used in the project.

d) Define Other Scheduling Data

1- Define all floor activities and the logical relationship network for these activities based on technological dependencies (i.e. hard logic). The network may be manually prepared, or can be the output of any knowledge-based planning system. Also, determine an initial schedule for a typical floor based on the activity hard logic dependencies defined in the logical network. Similarly, the initial schedule may be determined manually or may be the outcome of any scheduling program such as PRIMEVERA.

2- Define the special activity data parameters associated with scheduling multi -story buildings characterized with repetitive work.

3- Define other project data: first typical floor number, and last typical floor number.

Figure 8.3 depicts an overall picture of this first step.

8.2.2 Step-2: Scheduling Step (Production of the Schedule)

This second step consists of the actual production of the schedule. The schedule is generated by a knowledge-based system using a parallel scheduling approach. The approach involves moving through time period by period. At any time period, all activities eligible for scheduling are considered together, i.e. in parallel. These activities are ranked as a group according to a specified criteria. Each activity is then examined in the order of its priority against certain scheduling constraints. If all constraints are satisfied, the activity is scheduled during this period. If an activity can not be scheduled it is delayed until the next possible time period. This process is continued until all activities are scheduled.

During each cycle time t_i, a set of activities are defined whose predecessors are all scheduled and their early start (ES) values are less than or equal to t_i. This set is called the *Eligible Activity Set* (EAS). The activities of the EAS are selected for scheduling in an order determined based on a ranking or prioritizing criteria specified. The prioritizing process of the eligible activities uses three basic rules or ranking criteria defined in a specific order. The ranking criteria and the order in which it is applied are as follows:

1- **Late start value (LS)** - Activity with least late start (LS) value is scheduled first.

2- **Normal activity duration (nor-dur)** - Activities with the same late start value are ranked according to the shortest normal duration. The activity with the least normal duration (nor-dur) value is scheduled first.

3- **Maximum activity duration (max-dur)** - Activities with the same late start and normal duration values are ranked according to their shortest maximum duration. The activity with the least maximum duration (max-dur) value is scheduled first.

At every time period (t_i), each activity of the EAS is prioritized based on the ranking criteria. Once an activity is ranked and selected for scheduling, it is placed in the *Ordered Scheduled Set* (OSS). Each activity in the OSS is checked for the different scheduling constraints in the two scheduling stages. Once all constraints are verified to be satisfactory the activity is scheduled and dropped from the EAS and the OSS. If other activities become eligible for scheduling as a result of scheduling the current activity, these activities are added to the EAS. If any of the constraints are found unsatisfactory, the activity is dropped from the OSS but remain in the EAS to be re-considered for scheduling in the next possible time period. The scheduling procedure moves to consider the next activity in order and place it in the OSS.

Once all activities of the EAS are examined, the scheduling cycle is completed and the scheduling procedure moves to the next time period. At each successive time period, a new rank ordering of all eligible activities in the EAS is made and the process is continued until all activities are scheduled.

Scheduling Stages

As each activity placed in the OSS is considered for scheduling at any time period (t_i), activity constraints are checked and verified. Based on the continuity class and maximum splits parameters of the activity, the total number of floors to be scheduled is first calculated. For continuity class A activities, splitting is not allowed and the activity must be scheduled continuously. The activity is scheduled in one segment from first typical floor to the last floor. The activity is checked for all the specified constraints along all the floors. If any of the constraints are not satisfactory, the activity is not scheduled at the current time period (t_i) and is delayed to a following time period.

For continuity class B activities, each activity is divided into several segments along the typical floor. Each segment includes a specific number of floors. The total number of segments determined is equal to the value of the "maximum splits" parameter. The total number of floors for each segment (except last segment) is computed as follows:

$$\text{Total floors of segment } \textit{(except last segment)} = \frac{\text{Total number of typical floors}}{\text{Max number of splits}}$$

For the last activity segment, the total number of floors is computed as follows:

$$\text{Total floors } \textit{(last floor)} = \text{Total number of typical floors} - \sum \text{floors of previous segments}$$

Activity segments are scheduled sequentially in order of flocrs. Each segment is checked for the specified constraints individually. If the activity segment satisfies all the constraints it is scheduled and the next segment is considered. This process is continued until all segments are scheduled. If any constraint are found unsatisfactory, the activity segment is not scheduled during the current time period. The activity segment, along with other succeeding segments, are delayed to a following time period. If two or more segments are scheduled continuously, back to back, the scheduling procedure group these segments into one segment.

As illustrated in Figure 8.4, the verification process of the scheduling constraints is executed in two consecutive stages; *a Resource-Based Scheduling Stage*, and *a Space-Based Scheduling Stage.*

Scheduling Stage 1: Resource-Based Scheduling

In this stage horizontal logic constraints, vertical logic constraints and resource constraints are verified. The activity segment is checked for horizontal logic first. Using the normal duration, start and finish dates for the first and last floors of the segment is computed and compared to corresponding dates of all preceding scheduled activities. If there is a conflict, activity duration is modified to correspond to the dominant activity pace. Modifying the activity duration is limited by the maximum duration value defined. If the conflict still persists, the activity is delayed to a following time period.

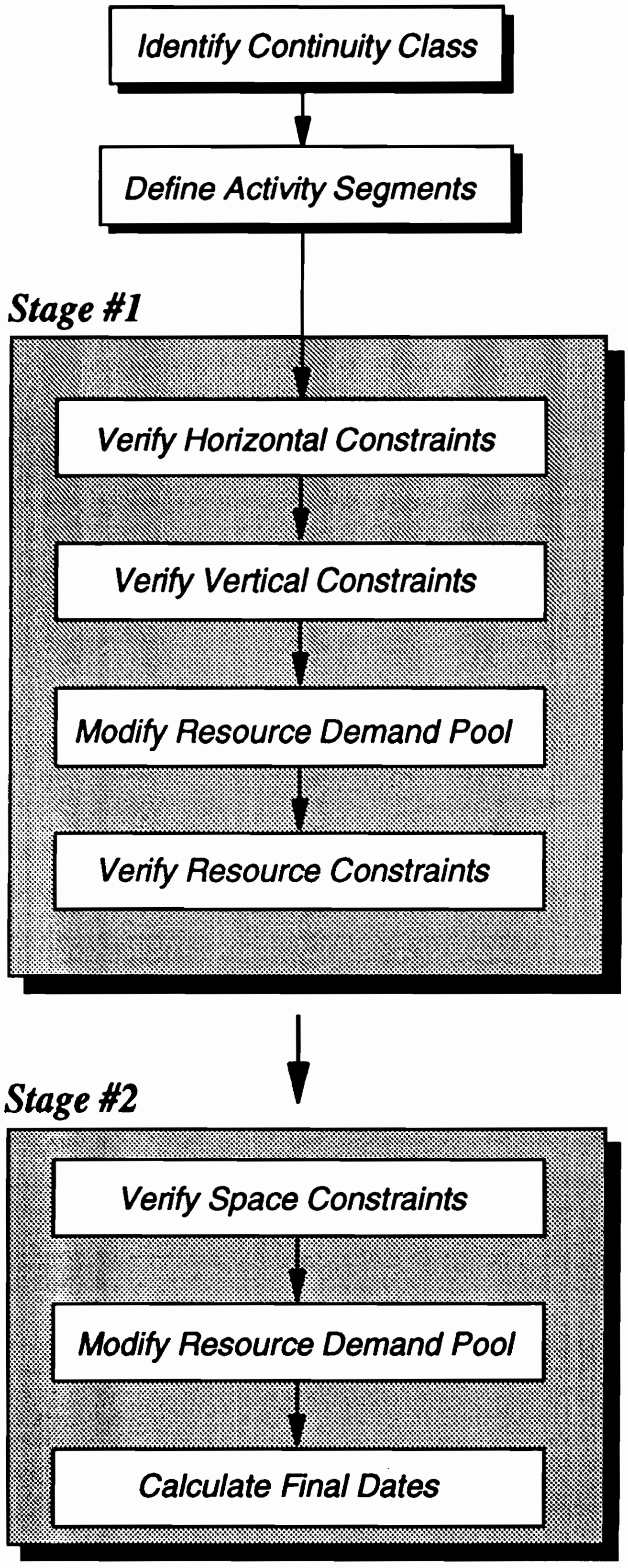

Identify Continuity Class
Define Activity Segments
Stage #1
Verify Horizontal Constraints
Verify Vertical Constraints
Modify Resource Demand Pool
Verify Resource Constraints
Stage #2
Verify Space Constraints
Modify Resource Demand Pool
Calculate Final Dates

Figure 8.4 - Scheduling Stages

Once the horizontal logic constraints are satisfied, the scheduling procedure verifies vertical logic constraints. Based on the vertical logic activity and lag, the activity is checked with the preceding activity imposing the constraint. Floor dates for the activity segment are checked with the corresponding dates of the vertical logic activity. If these constraints are not satisfied, the activity duration is modified or the activity may be delayed to a following time period.

At this point, if the activity duration has been modified beyond the initial normal value defined by the user, the scheduling procedure attempts to modify the resource demand pool (manpower and equipment only) of the activity. A simple function is utilized to compute the new resource demand pool based on the modified duration value.

Once the horizontal and vertical logic constraints are satisfied and the activity pool is adjusted, the scheduling procedure verifies resource constraints. The resource demand pool of activity is compared to the current resource availability during the preliminary period in which the activity is to be scheduled. The current availability value for each resource is determined by subtracting the usage of the resource by already scheduled activities during the preliminary period from the total availability pool of the resource. If available quantities of all resources required by activity are satisfactory, the scheduling procedure moves to the second stage.

Scheduling Stage 2: Space-Based Scheduling

The second stage in the production of the schedule is a space-based scheduling stage. Activity space demand values SD-1 (manpower and equipment) and SD-2 (material) are first determined. The value of SD-1 is determined based on the modified values of manpower and equipment. Values of SD-1 and SD-2 are computed using equations 5(a) and 5(b) defined in Chapter 7,

$$\text{Space Demand}_{\textit{Manpower and Equipment}}\ (\text{SD-1}) = \Sigma\ \text{Quantity x } (S_{\text{physical}} + S_{\text{surrounding}}) \quad \text{----(5a)}$$

$$\text{Space Demand}_{\textit{Equipment}}\ (\text{SD-2}) = [\ \Sigma\ (\text{Quantity x } S_{\text{physical}})\ ] + S_{\text{surrounding}} \quad \text{----(5b)}$$

Based on the values of SD-1 and SD-2, available space for each activity segment is compared to the current available space in corresponding blocks for every floor during preliminary start and finish times computed. If the space available is satisfactory, the activity segment is scheduled. If required work space is not available at one or more blocks, space-based scheduling decisions are made to schedule or delay the activity. If any of these decisions require increasing activity duration to satisfy space constraints,, the resource pool is re-modified before the activity is finally scheduled. Once all constraints are satisfactory, final start/finish dates are computed for the activity and the activity is dropped from the scheduling cycles.

The scheduling procedure in each stage is iterated for each activity segment until all constraints are satisfied. This process is repeated until all activities are scheduled. The scheduling stages are discussed in more detail in Chapter 12.

8.4 The SCaRC Scheduling System Components

The SCaRC scheduling system developed consists of two main components:

1- A Database System

2- A Knowledge-Based System

Both components are responsible for the overall schedule generation process. The database system provides for the defining all necessary input data (Pre-scheduling Step #1). The database also allows the user to view the output of the scheduling process (i.e. generated schedule). The second component of the SCaRC system, the knowledge-based system, is responsible for the actual production of the schedule (Step #2). Both components interact with each other through a set of ASCII interface files.

8.4.1 The Database System

The database system is the first component of the SCaRC scheduling system. All data needed by the knowledge-based system (2nd component) is defined via the database system. The majority of the required data is input by the user. Other data, such as geometric data of design elements, is extracted directly from external ASCII type files. All data defined in the database system is transferred to the knowledge-based system, through a set of ASCII files, for processing and generation of the schedule.

The database system also provides for retrieval of the generated schedule results from the knowledge-based system. The objective is to allow the user a greater flexibility to view the results of the generated schedule in a variety of output formats including graphical output. The results of the generated schedule is transferred by the knowledge-based system to an output ASCII file. The file is extracted by the user from the database system and stored in a database DBF type file.

In addition to data input and output, the database system allows to define global space demand values for resources associated with scheduling multi-story buildings. Values to be defined should be based on space demand for unit resources. This allows to build some sort of a library of space demand data for generic resource types. This space demand data for different resources may be based on empirical equations, site observations, or historical data from previous similar projects. The data may be retrieved during user input to assist him/her in making feasible estimates while defining space demand values of resources for the project under consideration.

The database system is designed as an interactive menu-driven system. As depicted in Figure 8.5, the system comprises mainly of a database file structure that consists of database files linked to a menu/screen system. Data stored in the database files is manipulated via a set of screens also linked to the menu system.

The database system contains six main categories of data:

1- Global resource space demand system data

2- Activity input data

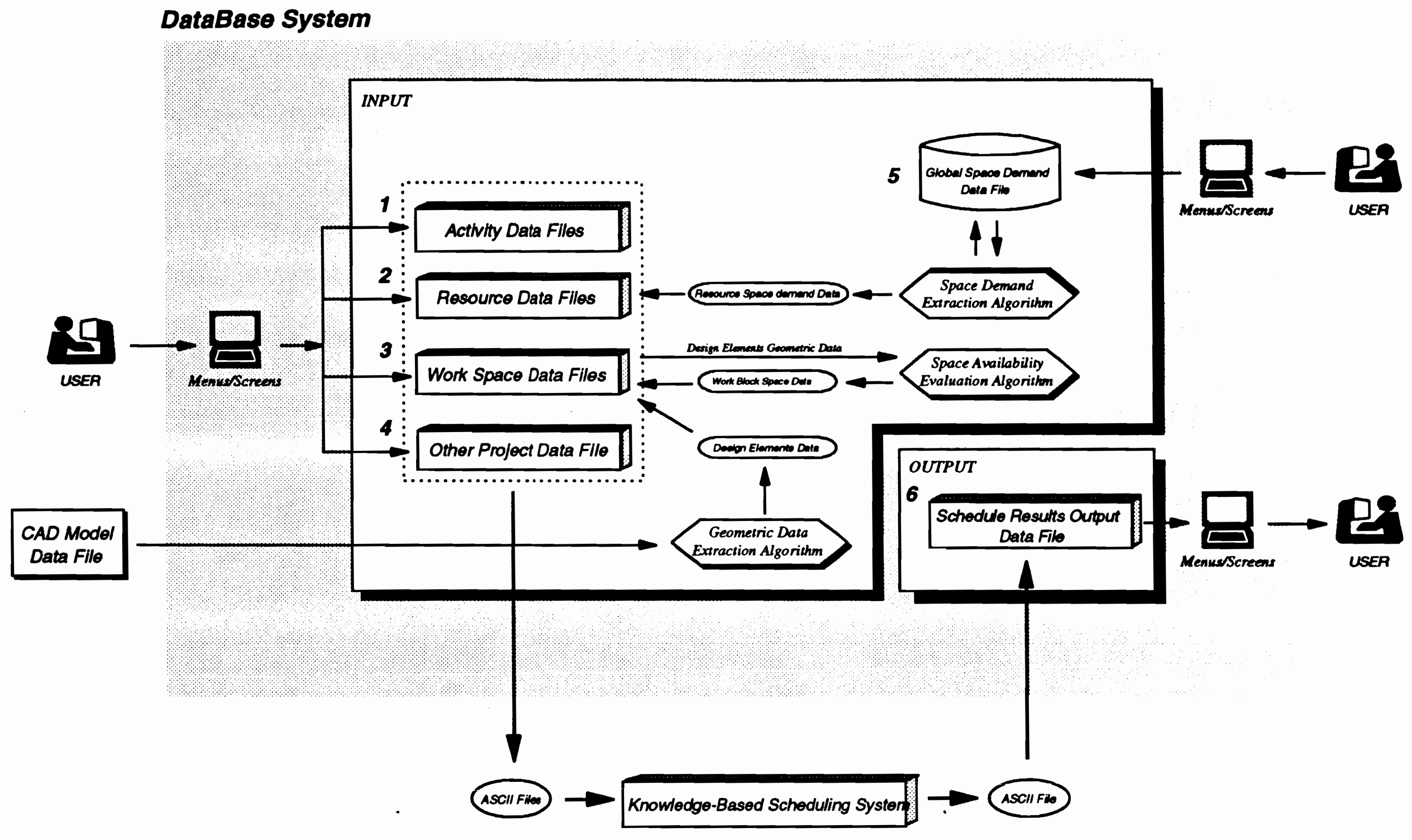

DataBase System
INPUT
USER
Menus/Screens
1
Activity Data Files
2
Resource Data Files
3
Work Space Data Files
4
Other Project Data File
CAD Model Data File
5
Global Space Demand Data File
Space Demand Extraction Algorithm
Resource Space demand Data
Design Elements Geometric Data
Space Availability Evaluation Algorithm
Work Block Space Data
Design Elements Data
Geometric Data Extraction Algorithm
Menus/Screens
USER
OUTPUT
6
Schedule Results Output Data File
Menus/Screens
USER
ASCII Files
Knowledge-Based Scheduling System
ASCII File

Figure 8.5 - Overall Database System

3- Resource input data

4- Work space input data

5- Other project input data

6- Schedule results output data

The *Global Resource Space Demand System Data* constitutes global space demand values for various resources associated with multi-story building construction. Values is defined based on unit resources.

The *Activity Input Data* constitutes all data associated with the project activities. Four sub-categories of activity input data are present. These sub-categories are:

1- General Activity Data: This consists of general activity scheduling data associated with general scheduling procedures. Data in this sub-category is based on the user defined typical floor logical network and initial schedule.

2- Special Activity Data: This sub-category of activity input data consists of the special scheduling parameters associated with linear scheduling of multi-story buildings.

3- Activity Resource Demand Data: This sub-category of activity input data defines resource number, type and quantity required for each activity.

4- Activity Work Space Data: This forth sub-category of activity input data defines work space related data necessary for the space-based scheduling decisions during the schedule generation of the schedule.

The *Resource Input Data* constitutes data specific to each resource of the project. Resource data consists of availability data and required space demand data associated with each resource.

The *Work Space Input Data* defines all work space data associated with the work block structure of the typical floor. Two types or sub-categories of work space data for the project are present in the database:

1- Zone and Layer Work Block Data: This first sub-category constitutes space availability data for the different zone and layer work blocks.

2- Design Elements Data: This second sub-category of data defines geometric data of each design element of the typical floor. Design elements and their geometric data are extracted from a CAD model of the typical floor. The geometric data for each element is defined using different geometric parameters of the element (e.g. length, width, depth, surface area, thickness, etc.). These geometric parameters is extracted form the CAD model of the typical floor.

The *Other Project Input Data* defines other specific project data. This includes: first typical floor number, and last typical floor number.

The *Schedule Results Output Data* constitutes the output data generated by the scheduling system. Output data is extracted from the knowledge-based system component and stored in the database to allow the user to review the results.

Within the database file structure environment, several evaluation algorithms are developed to interact with the database files. There are three evaluation algorithms developed. These are:

1- Space Demand Extraction Algorithm

2- Geometric Data Extraction Algorithm

3- Space Availability Evaluation Algorithm

The *Space Demand Extraction Algorithm* is responsible for extracting global work space demand values for all resources of the project from the global data file and store it in the resource data file. The purpose of this process is to provide the user with standard space demand values of unit resources. These values can assist the user to make a better estimate of space demand for each resource taking into account the conditions of the project under consideration.

The *Geometric Data Extraction Algorithm* is responsible for extracting the geometric data of design elements from a CAD model database file and store it into a design element file in the database. The extracted geometric data is utilized to compute the space availability of each work block. This computation is accomplished by the Space Availability Evaluation Algorithm (third algorithm).

The *Space Availability Evaluation Algorithm* is responsible for computing the total work space availability for each work block. The algorithm computes the space of a work block based on the geometric data of all design elements allocated to that block.

The database system is developed using dBASE IV programming language. The database file structure, menu/screen system, and evaluation algorithms are discussed in more detail in Chapter 9.

8.4.2 The Knowledge-Based System

The second component of SCaRC is the knowledge-based scheduling system. The system architecture consists of *schemata representational structure* and *scheduling modules*.

The schemata representational structures are used to define the declarative knowledge. Extracted input data along with other knowledge and data are stored in the system using mainly schemata structures defined using the ART-IM schema format.

The scheduling modules define the procedural knowledge responsible for the different procedures to verify constraints and generate a schedule. The modules are defined using the ART-IM rule and user function formats. External data interface procedures along with schedule generation procedures are defined in the scheduling modules.

The Schemata Representational Structures

Input data defined in the database system is transferred and stored in the knowledge-based system using a hierarchical representational structures that are composed of objects or schemas. Each object or schema has a unique name and contains one or more slots. Each slot contains single or multiple values that defines its attribute. Slot attributes are either data attached to the schema or define a method that is activated when a message is sent to the appropriate slot.

There are three data schemata structures created: an *Activity Schemata Structure*, a *Resource Schemata Structure*, and a *Work-Block Schemata Structure*. Each data schemata structure stores specific scheduling data. The different schemas of the data schemata structures are created at the beginning of the scheduling process during extracting data from the database. These schemas are stored in a global store of the knowledge-based system called the *Context*.

Schemas of each data schemata structure are linked with a parent schema using an ART-IM "*is-a*" relationship. There are three parent schemas: an *Activity Parent Schema*, a *Resource Parent Schema*, and a *Work-Block Parent Schema*. The "is-a" relationship allows the different data schemas to inherit additional slots from the parent schema. This reduces redundancy and memory usage during the creation of each data schema. Also, this allows to perform iterative operations during scheduling on data schemas that belong to a specific parent. As opposed to the data schemata, the parent schemata are permanently stored in the knowledge base of the system.

The overall picture of the schemata representational structures is depicted in Figure 8.6.

1- The Activity Schemata Structure. Each construction activity is represented by a single activity schema. Each schema stores all data associated with that activity. Activity related data include: general activity scheduling data, special activity parameters data, activity resource demand data and activity space demand data. Figure 8.7 depicts an example of

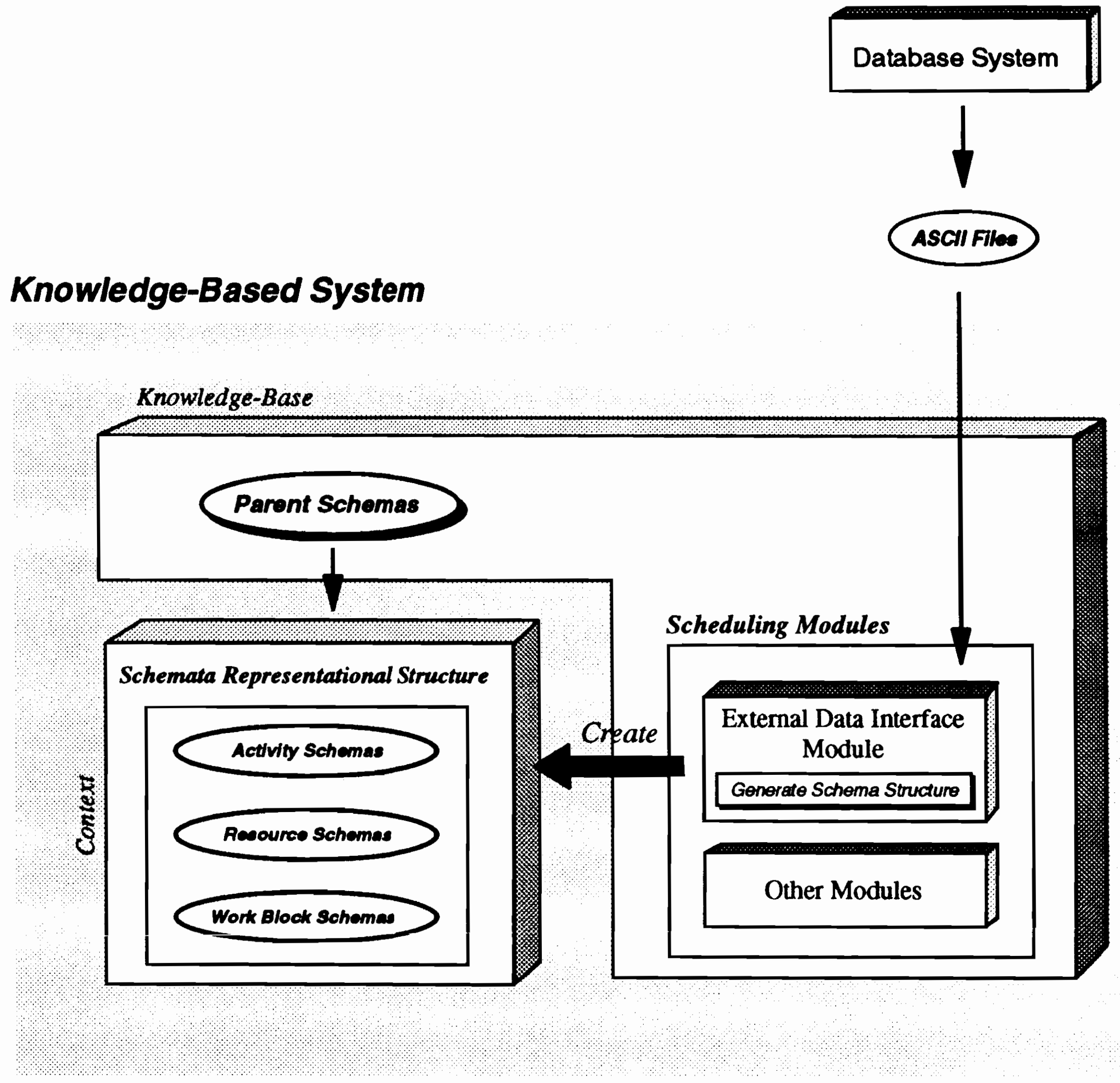
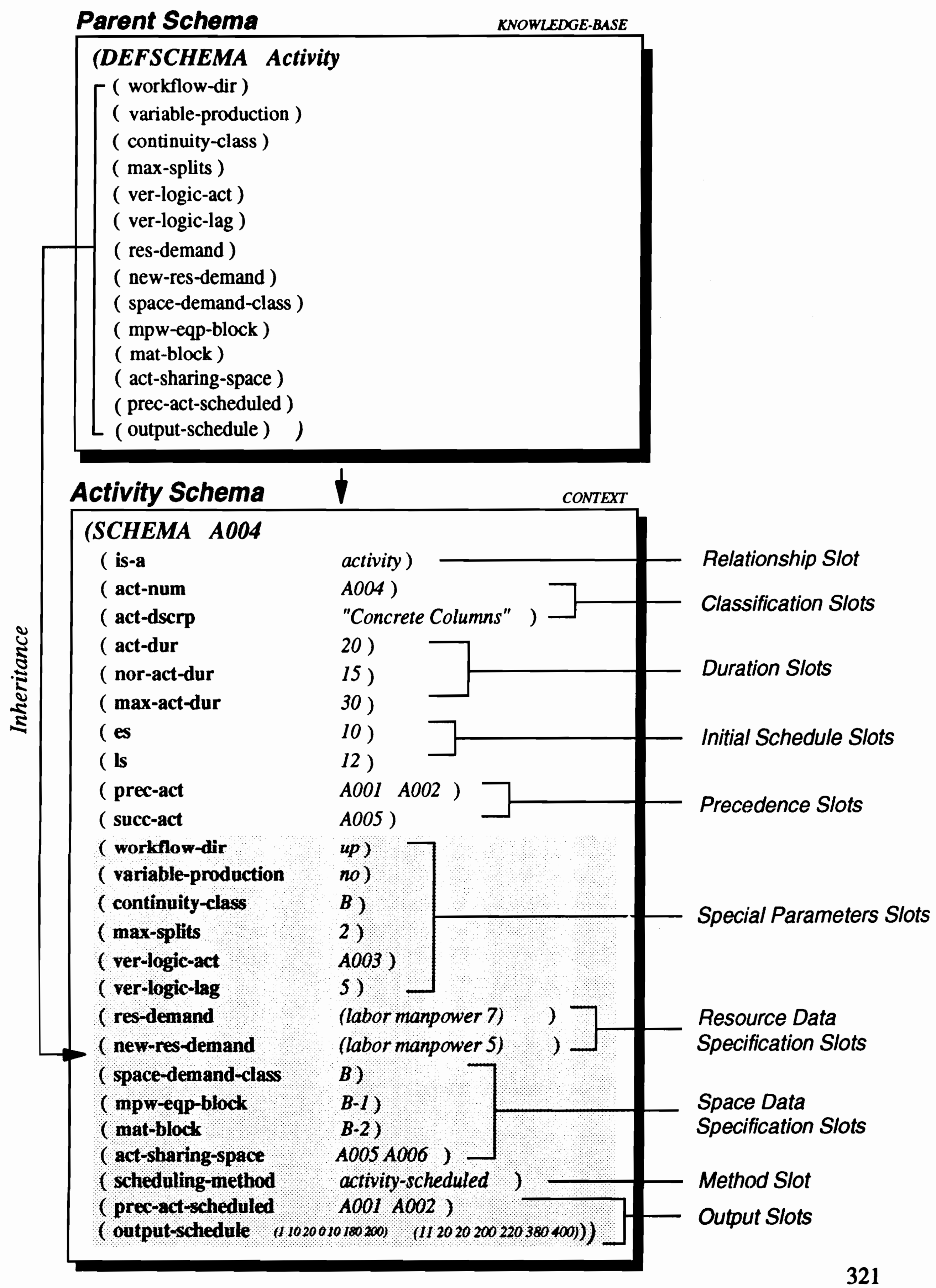


Figure 8.6 - Schemata Representational Structure



```
(DEFSCHEMA  Activity
  ( workflow-dir )
  ( variable-production )
  ( continuity-class )
  ( max-splits )
  ( ver-logic-act )
  ( ver-logic-lag )
  ( res-demand )
  ( new-res-demand )
  ( space-demand-class )
  ( mpw-eqp-block )
  ( mat-block )
  ( act-sharing-space )
  ( prec-act-scheduled )
  ( output-schedule )    )
```

```
(SCHEMA  A004
  ( is-a                  activity )
  ( act-num               A004 )
  ( act-dscrp             "Concrete Columns"  )
  ( act-dur               20 )
  ( nor-act-dur           15 )
  ( max-act-dur           30 )
  ( es                    10 )
  ( ls                    12 )
  ( prec-act              A001  A002  )
  ( succ-act              A005 )
  ( workflow-dir          up )
  ( variable-production   no )
  ( continuity-class      B )
  ( max-splits            2 )
  ( ver-logic-act         A003 )
  ( ver-logic-lag         5 )
  ( res-demand            (labor manpower 7)      )
  ( new-res-demand        (labor manpower 5)      )
  ( space-demand-class    B )
  ( mpw-eqp-block         B-1 )
  ( mat-block             B-2 )
  ( act-sharing-space     A005 A006  )
  ( scheduling-method     activity-scheduled   )
  ( prec-act-scheduled    A001  A002  )
  ( output-schedule   (1 10 20 0 10 180 200)   (11 20 20 200 220 380 400)))
```

Figure 8.7 - Example Activity Schema

an activity schema and its slots. Some slots are inherited from the parent schema, others are created during the generation of the schema. The activity schema contains the following slots:

- A ***Relationship Slot*** represents links between the activity schema and the parent schema. One relationship slot **is-a** identifies the schema as an activity schema. This slot is generated during scheduling.

- ***Classification slots*** identify the number and description of the activity. Two slots are used for this purpose: (1) the **act-num** slot stores the activity number; and (2) the **act -dscrp** slot stores the description of the activity. These slots are generated during scheduling.

- ***Duration slots*** store information describing the duration of the activity. Three slots are available for this purpose: (1) the **act-dur** slot stores the duration computed by the scheduling procedure during the schedule generation process. The value stored in this slot represents the duration used to calculate the start and finish dates of the activity segment; (2) the **nor-act-dur** slot stores the normal activity duration defined by the user; and (3) the **max-act-dur** slot stores the maximum activity duration also defined by user.

The act-dur slot is inherited, while the other two duration slots are generated during scheduling.

- ***Initial Schedule slots*** describe the initial scheduling data for the activity based on the initial schedule of a typical floor. There are two slots: (1) the **es** slot defines the early start time of the activity; and (2) the **ls** slot defines the late start time of the activity. Both slots are generated during scheduling.

- ***Precedence slots*** store the numbers of the preceding and succeeding activities to the activity. Two precedence slots are available: (1) the **prec-act** slot stores the numbers of all activities preceding to the activity; and (2) the **succ-act** slot stores the numbers of all activities succeeding to the activity. These slots represents the hard logic of the typical floor network as defined by the user. Both slots are generated during scheduling.

- ***Special Parameters slots*** define the special activity parameters that are associated with scheduling of repetitive work in multi-story buildings. The **workflow-dir** slot stores the direction of work for the construction crew along the typical floors. The **variable -production** slot stores a flag to indicate whether the activity production rate varies as the crew moves upward or downward along the typical floors as a result of the combined effect of travel time and learning curve. The **continuity-class** slot specifies the continuity class of the activity. The **max-splits** slot specifies the maximum number an activity can be split along the floors. The **ver-logic-act** slot stores the activity number forcing the vertical constraint. The **ver-logic-lag** slot stores the total number of floors defining the vertical constraint. All the special parameters slots are inherited.

- ***Resource data slots*** defines the resource types and quantities required by the activity. There are two slots: (1) the **res-demand** slot defines the resource code, type and quantity for each resource required to perform the activity; and (2) the **new-res-demand** defines the resource code, type and new quantities as computed by the scheduling procedure based on the modified activity duration. If the activity duration is not modified during the scheduling of the activity, values of resource quantities in this slot will be identical to the values defined in the res-demand slot. These slots are inherited.

- ***Space data slots*** specify the different work space data of the activity. A **space -demand-class** slot stores the space demand class of the activity. A **mpw-eqp-block** slot defines the work block number to which manpower and equipment are allocated. A **mat-block** specifies the work block number to which activity material is allocated. An **act-sharing-space** slot stores the number of other activities sharing the same work block with the activity. These activities are determined during the scheduling process. All the space data slots are inherited.

- ***Method slot*** defines a method that is activated to execute a specific scheduling procedure. The method is executed by sending messages to this slot using an object

-oriented approach. One slot :**scheduling-method** is used. The value of the slot attribute defines the method to be activated. Each method is associated with a scheduling procedure responsible for verifying a specific constraint. Once a method is activated and completed, the attribute of the slot is modified to define the next method. Once an activity is scheduled, the attribute of the slot is modified to "activity -completed'. This slot is created during scheduling.

• ***Output slots*** define the schedule results for the activity. There are two slots: (1) the **prec-act-scheduled** slot specifies the number of the preceding activities that are scheduled. When the attribute of this slot matches that of the **prec-act** slot, the current activity becomes eligible for scheduling; (2) the **output-schedule** slot defines the scheduling results of the activity. The attribute of this slot specifies the start and finish floors for the activity segment, the computed duration of segment, and the start/finish dates of the 1st and last typical floors of the segment. An example output slot is shown in Figure 8.8.

2- The Resource Schemata Structure. Each project resource is represented by a single resource schema. Each schema stores all data associated with that resource. Resource related data consists of: resource code, resource type, availability value, and space demand data. Figure 8.9 depicts an example of a resource schema and its slots. Some slots are inherited from the parent schema, others are created during the generation of the schema. The resource schema contains the following slots:

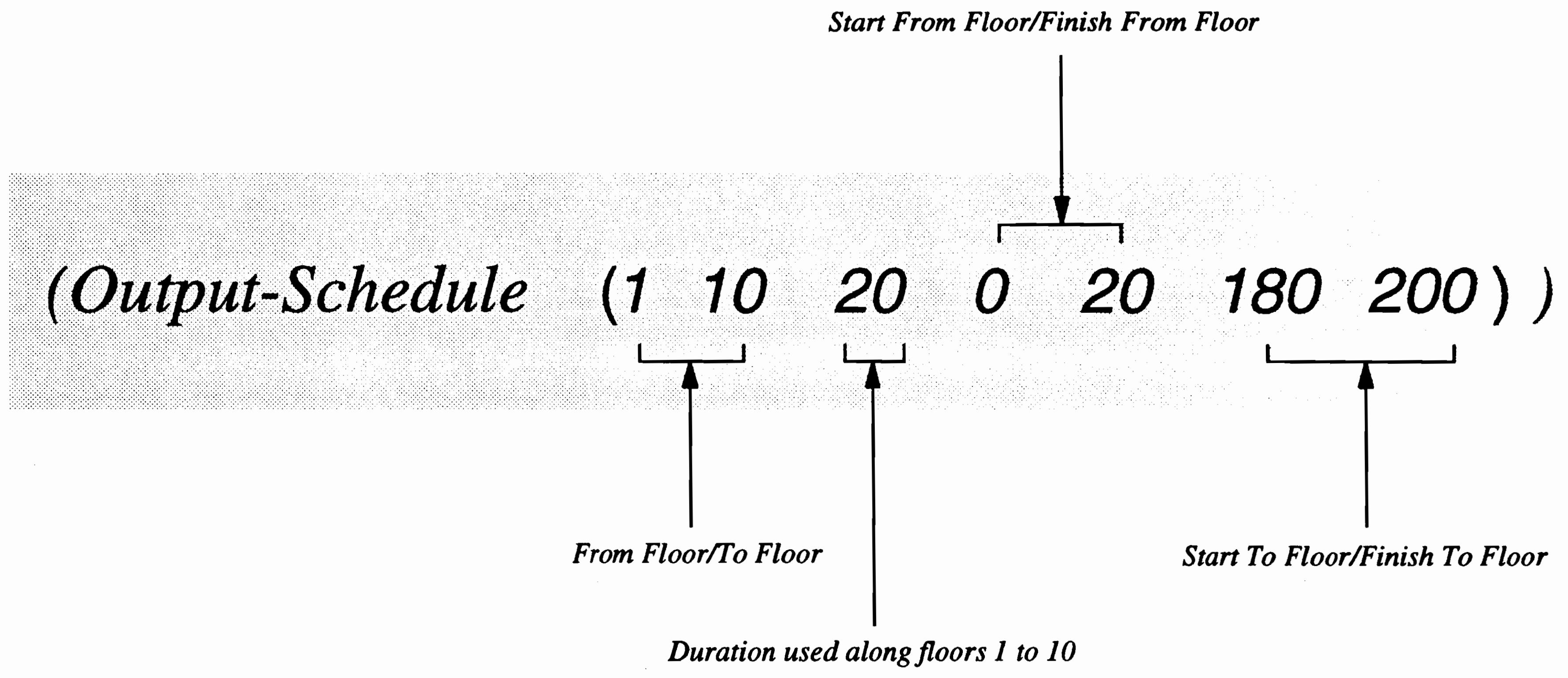

Start From Floor/Finish From Floor
(Output-Schedule (1 10 20 0 20 180 200))
From Floor/To Floor
Start To Floor/Finish To Floor
Duration used along floors 1 to 10

Figure 8.8 - Example Attribute Value of an "Output-Schedule" Slot

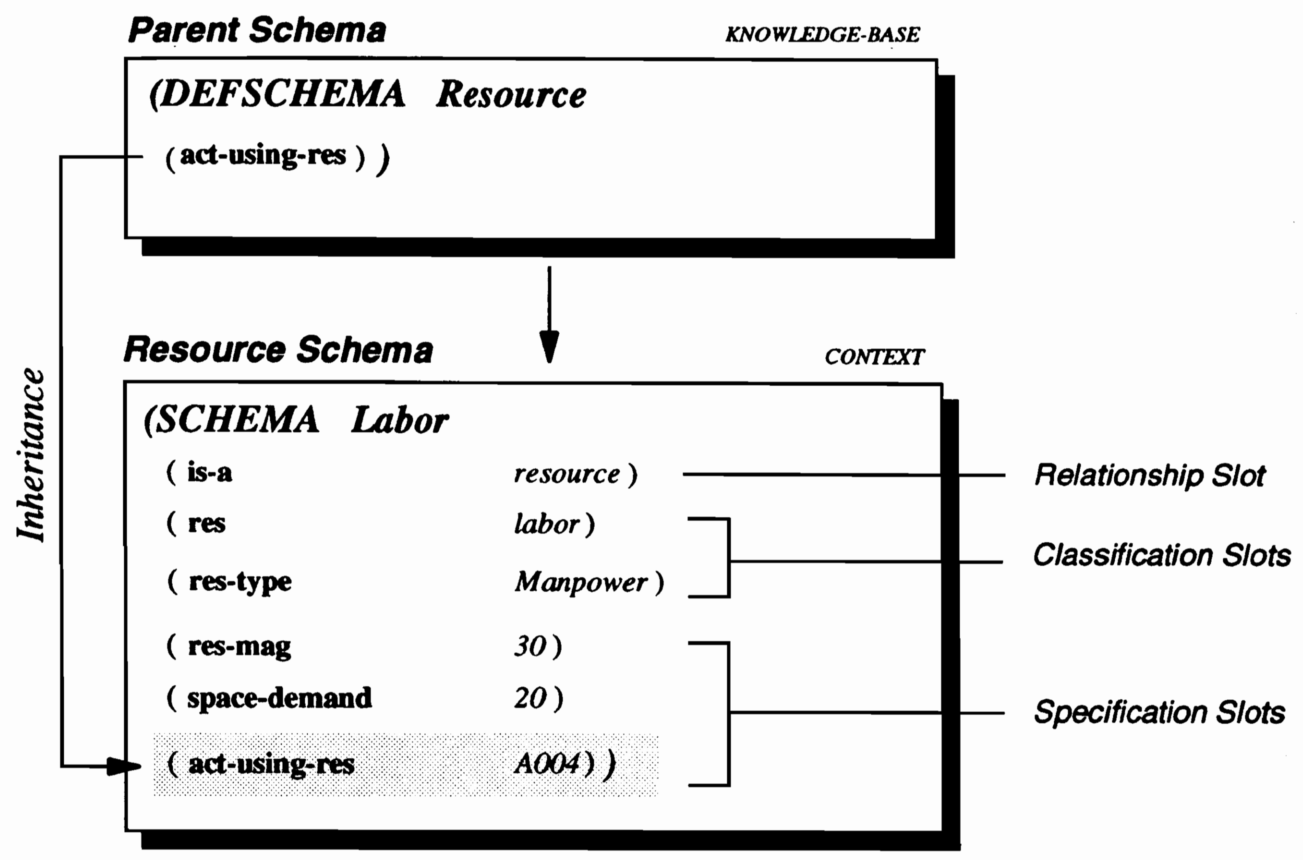


Figure 8.9 - Example Resource Schema

- A ***Relationship Slot*** represents links between the resource schema and the parent schema. One relationship slot **is-a** identifies the schema as a resource schema. This slot is created during scheduling.

- ***Classification slots*** identify the number and type of the resource. There are two classification slots; (1) a **res** slot stores the number of the resource, and (2) a **res-type** slot defines the type of each resource. These slots are created during scheduling.

- ***Specification slots*** contain specifications data of the resource. There are three slots: (1) a **res-mag** slot specifies the availability value of the resource, as defined by the user; (2) a **space-demand** slot specifies the total work space demand required for the resource as computed by the system; and (3) an **act-using-res** slot stores the number of the activity using the resource. During the scheduling process, as an activity is scheduled, its number is stored in this slot for each resource required for the activity. This allows the scheduling procedure to identify those activities utilizing the resource during verification of resource constraints. The first two slots are generated during scheduling. The last slot is inherited.

3- The Work-Block Schemata Structure. Work-block schemas store all data associated with zone and layer work blocks. This data consists of: work block number and total space availability value of the block. Figure 8.10 depicts an example of a work-block schema and its slots. All slots are created during the generation of the schema. The work-

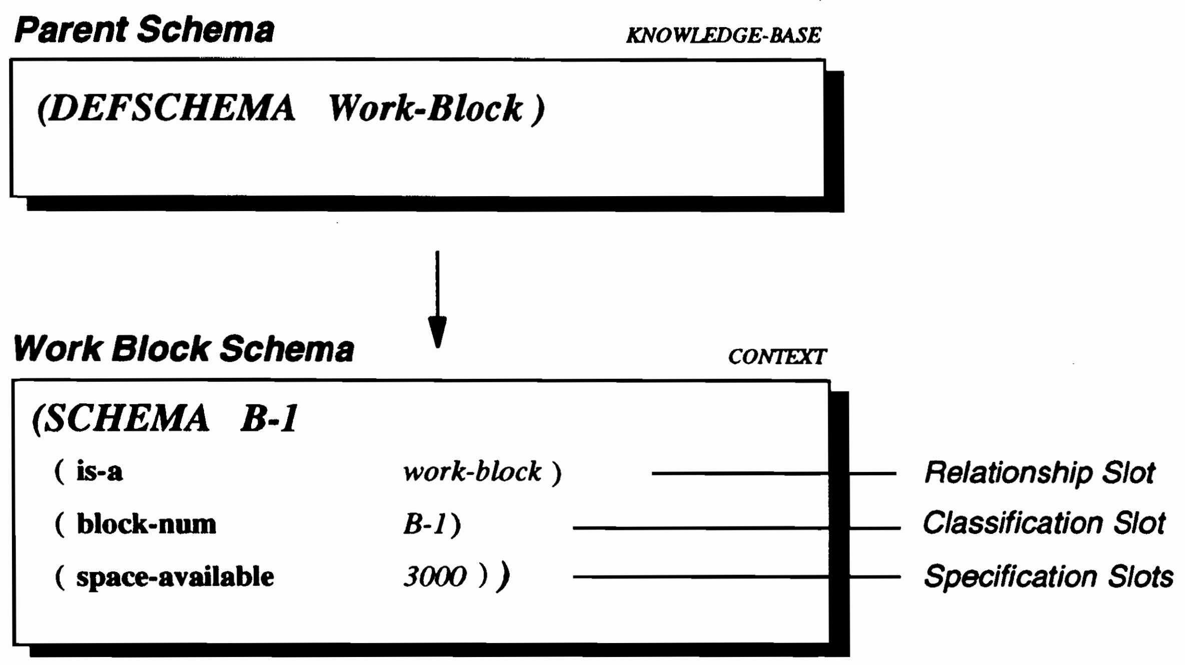


Figure 8.10 - Example Work Block Schema

block schema contains the following slots:

- ***A Relationship Slot*** represents links between the work block schema and the parent schema. One relationship slot **is-a** identifies the schema as a work-block schema.

- ***A classification slot*** identifies the number of the work block. The **block-num** slot defines the number that identifies each block.

- ***A Specification slot*** contains information about the available space in the block. The **space-available** slot defines the total work space availability of the block as computed by the geometric algorithm.

The Scheduling Modules

The scheduling modules constitute the major part of the knowledge-based system component. The These modules define the procedural scheduling knowledge responsible for the reasoning process and the generation of the schedule. The modules utilize the data and knowledge stored in the schemata representational structures to generate the schedule. Each module comprises of several
processors. Each processor accomplishes a specific task of the module.

Figure 8.11 depicts the scheduling modules and the processors of each module. There are

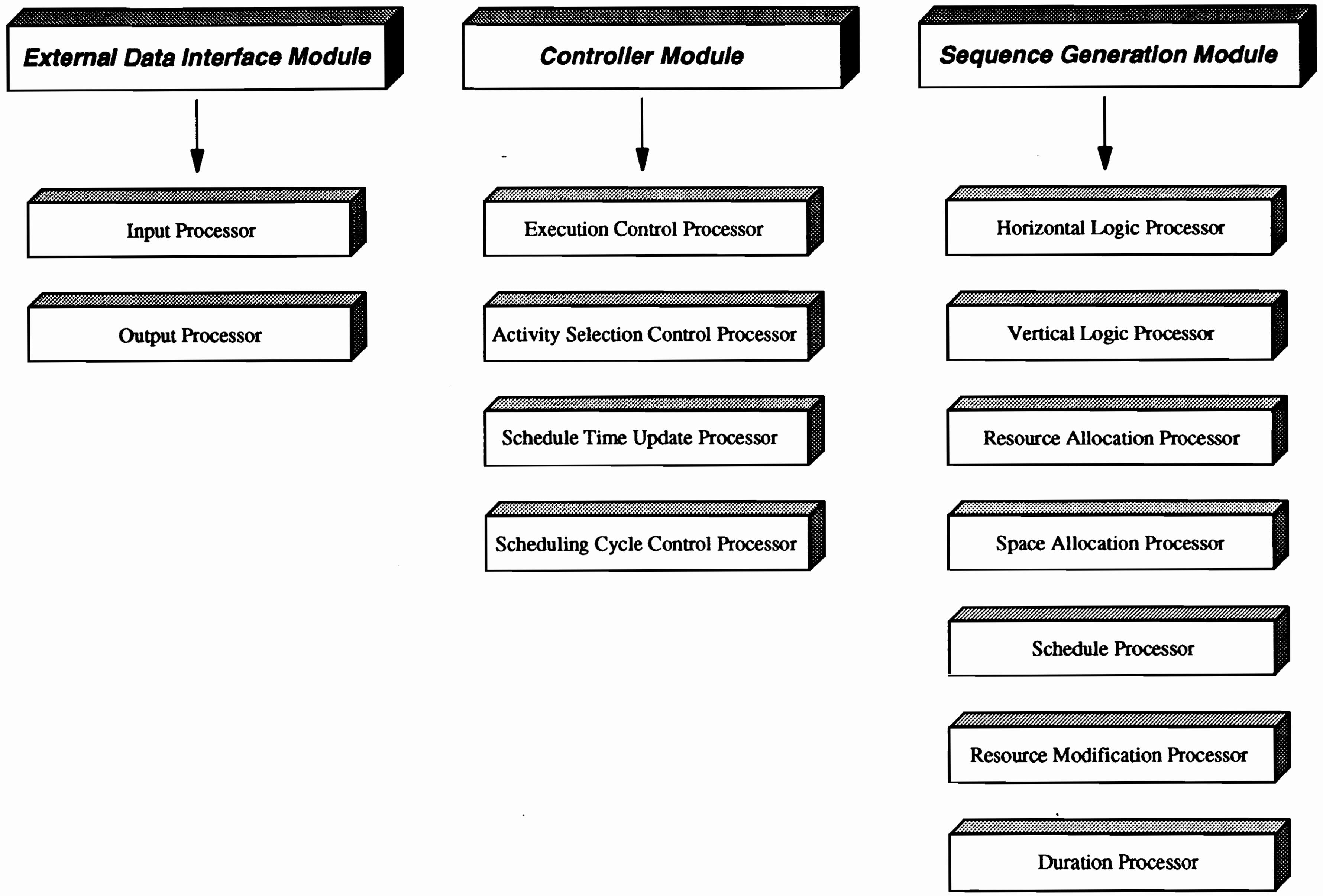

External Data Interface Module
Input Processor
Output Processor
Controller Module
Execution Control Processor
Activity Selection Control Processor
Schedule Time Update Processor
Scheduling Cycle Control Processor
Sequence Generation Module
Horizontal Logic Processor
Vertical Logic Processor
Resource Allocation Processor
Space Allocation Processor
Schedule Processor
Resource Modification Processor
Duration Processor

Figure 8.11 - Scheduling Modules and Processors

three modules:

1- An External Data Interface Module.

2- A Controller Module.

3- A Sequence Generation Module.

The ***External Data Interface module*** performs two main functions. The module is responsible for extracting the input data from the database and storing it in the context of the knowledge-based system using the schemata representational structures. The module is also responsible for transferring the outcome of the scheduling process (i.e. generated schedule) back to the database system to allow the user to review the results. Input/Output data is interchanged between the module and database through a set of ASCII files.

The functions of the external data interface module are performed by two processors:

(1) Data Input Processor; and

(2) Data Output Processor

The *Data Input Processor* is responsible for extracting all input data defined in the database and storing it in the context using the schemata structures. The data input processor consists of

several ART-IM rules that extracts the input data. There are 7 rules classified as follows:

Rule-1: *Activity schemata creation and activity general data extraction rule*

Rule-2: *Activity special data extraction rule*

Rule-3: *Activity resource demand data extraction rule*

Rule-4: *Activity space data extraction rule*

Rule-5: *Resource schemata creation and data extraction rule*

Rule-6: *Work block schemata creation and data extraction rule*

Rule-7: *Other project data extraction rule*

The data input processor and its extraction rules are illustrated in Figure 8.12.

The *Data Output Processor* is responsible for transferring the results of the generated schedule back to the database. The purpose of this process is to allow the user to view the results of the schedule. The processor consists of one rule: **Rule**: *Output-Data.* The rule extracts the schedule data stored in each activity schema and sends it back to the database. The output processor is illustrated in Figure 8.12.

The external data interface module is discussed in detail in Appendix A.

The ***Controller module*** constitutes the second module of the knowledge-based system. The module controls the overall scheduling process. The module has two basic function:

1- Control the execution of the external data interface module.

2- Processing of the scheduling cycles.

In the first function, the controller module is responsible for the control of execution of

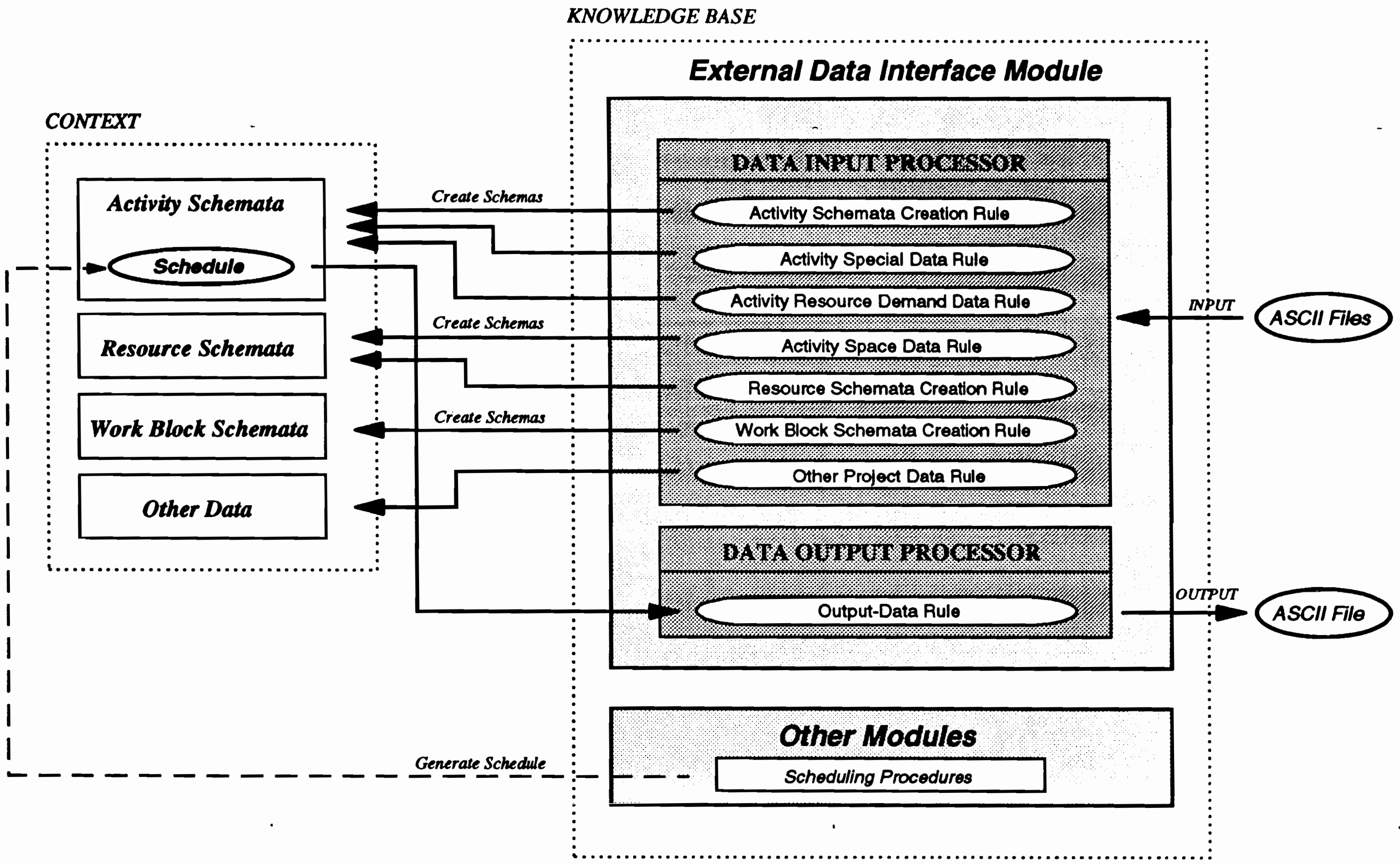


Figure 8.12 - The External Data Interface Module

the data input and data output processors of the external data interface module. The execution of these processors is dependent on specific ART-IM facts that should be present in the context. The controller module is responsible for inserting these facts to initiate the start of execution of these processors.

The second function of the controller module is the actual processing or execution of the scheduling cycles. During each cycle, the module executes the sequence generation module (third module) to verify the scheduling constraints and schedule the activities. The processing of the scheduling cycles is accomplished by performing a set of tasks:

1- Determine the list of activities that are eligible for scheduling in every cycle. The eligible activities define the eligible activity set (EAS).

2- Define the scheduling time for each scheduling cycle.

3- Prioritize the activities of the EAS during every scheduling cycle. The objective is to determine the order in which each activity competing for resources and work space is considered for scheduling. The prioritizing process is accomplished using the different defined ranking criteria. During each scheduling cycle, the activity with the highest priority is placed in the Ordered Scheduled Set (OSS) to be considered for scheduling.

4- Execute the sequence generation module to verify the scheduling constraints for the activity placed in the OSS. Using an object oriented approach, instructions are sent to the different processors of the sequence generation module to verify the scheduling constraints and schedule each activity.

The functions of the controller module are performed by several processors that constitute the module. There are four processors:

1- An Execution Control Processor

2- An Activity Selection Control Processor

3- A Cycle Time Control Processor

4- A Scheduling Cycle Control Processor

The *Execution Control Processor* performs the first function of the controller module. The processor controls the execution of the data input and data output processors of the external data interface module.

The other three processors perform the second function of the controller module. The *Activity Selection Control Processor* determines the list of activities that are eligible for scheduling at the beginning of each scheduling cycle (EAS).

The *Cycle Time Control Processor* determines the start time for each scheduling cycle. The processor computes the next possible time to start each cycle based on the list of activities (EAS) that is considered for scheduling in that cycle.

The *Scheduling Cycle Control Processor* is the main processor of the controller module. This processor is responsible for processing each scheduling cycle based on the cycle time defined and eligible activities selected. During each cycle, the processor is responsible for prioritizing activities of the EAS. The processor uses heuristic criteria to define the order in which these activities are considered for scheduling.

The processor is also responsible for sending instructions to the sequence generation module to verify the different scheduling constraints for each prioritized activity placed in the ordered scheduled set (OSS). Instructions are sent to the sequence generation module in the form of messages using an object-oriented programming approach. Each message executes a method or procedure responsible for verifying a specific constraint. The sequence generation module consists of several procedures (i.e several processors). As each procedure is executed, a new message is sent to execute another procedure. The procedures are executed in a specific order. The process is continued until all procedures of the sequence generation module are executed and all constraints are verified.

Figure 8.13 depicts the overall control process of the controller module. The module is discussed in detail in Appendix B.

The ***Sequence Generation Module*** is the center piece of the knowledge-based system. The module is responsible for verifying the scheduling constraints and sequencing of each activity. The module contains several processors that generate the schedule in the two scheduling stages. There are seven processors:

1- A Horizontal Logic processor

2- A Vertical Logic Processor

3- A Resource Allocation Processor

4- A Space Allocation Processor

5- A Schedule Processor

6- A Resource Modification Processor

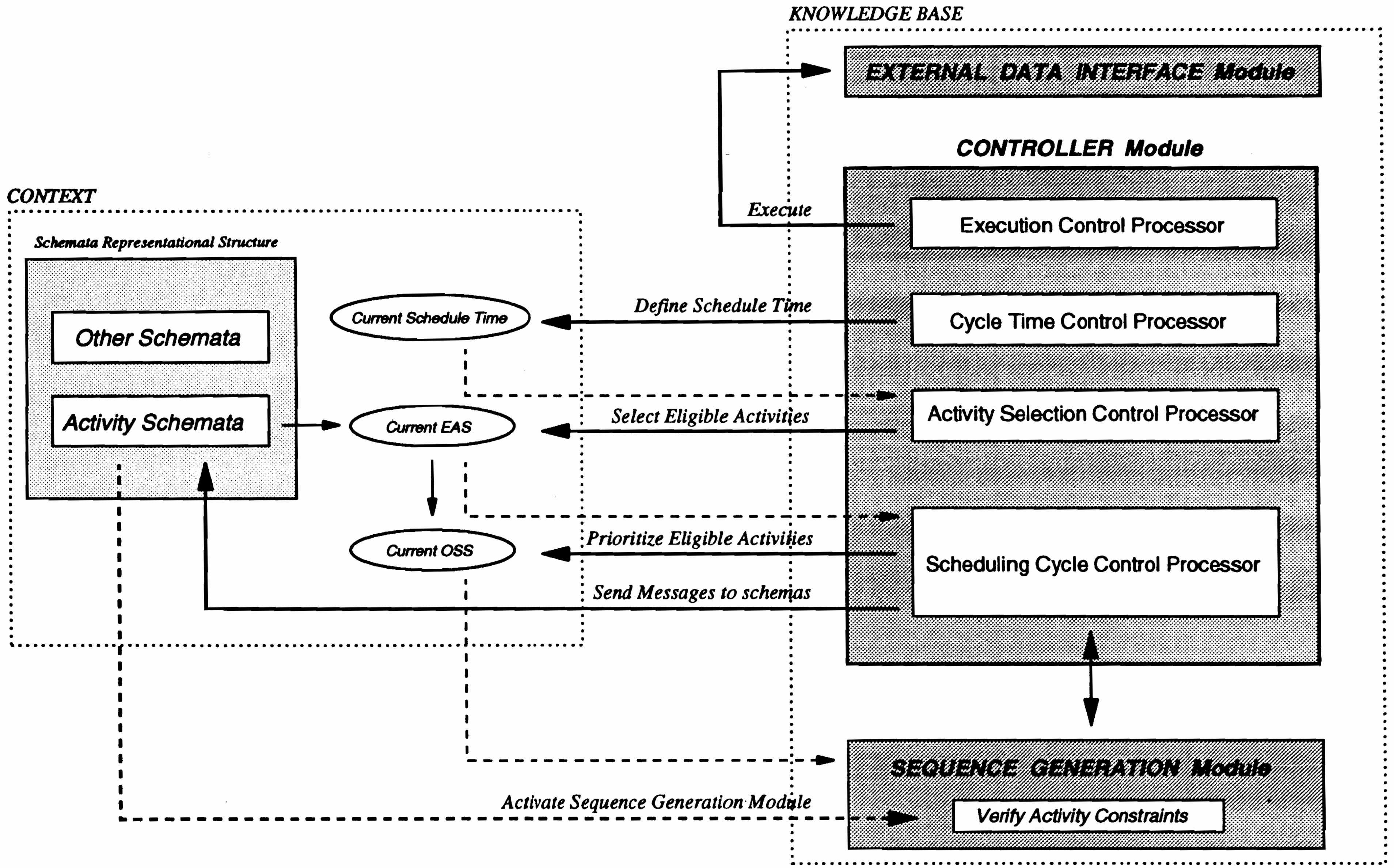

KNOWLEDGE BASE
EXTERNAL DATA INTERFACE Module
CONTROLLER Module
Execution Control Processor
Cycle Time Control Processor
Activity Selection Control Processor
Scheduling Cycle Control Processor
SEQUENCE GENERATION Module
Verify Activity Constraints
CONTEXT
Schemata Representational Structure
Other Schemata
Activity Schemata
Current Schedule Time
Current EAS
Current OSS
Execute
Define Schedule Time
Select Eligible Activities
Prioritize Eligible Activities
Send Messages to schemas
Activate Sequence Generation Module

Figure 8.13 - The Controller Module

7- A Duration Processor

The sequence generation processors are defined as ART-IM functions in the knowledge-base. Each processor consists of one or more functions. Except for the resource modification processor and the duration processors, the execution of the other processors is controlled by the controller module using an object oriented approach. As explained earlier, this is accomplished by sending execution messages to a specific slot of each activity to execute the method defined in that slot. The method defined in each slot will be the name of the function of the particular processor. The resource modification processor and the duration processor are executed from within the other five processors by calling the function name defining these processors.

The *Horizontal Logic Processor* is responsible for verifying the horizontal logic constraints defined by the floor network. Based on the continuity class of activity, the processor verifies that all start/finish dates of each activity segment does not contradict with the corresponding start and end dates of preceding activities along all floors. New activity duration may be selected to
confirm with the dominant activity pace.

The *Vertical Logic Processor* verifies the vertical logic constraints among the activities. Based on the vertical logic activity defined, this processor insures that vertical constraints are satisfied between the activity and vertical activity forcing the constraint. Activity duration may be adjusted to confirm with the vertical constraints.

The *Resource Allocation Processor* is responsible for verifying resource availability to schedule the activity. Preliminary start and finish dates are first computed. Resource demand are compared to current availability during the computed preliminary dates. If available resources are satisfactory control is transferred to the space allocation processor. If available resources are not satisfactory for the activity during the computed preliminary dates the activity can not be scheduled. The processor determines the next possible time period when these resources become available.

As control is transferred to the *Space Allocation Processor*, the scheduling procedure moves into the second stage. The main function of the processor during this stage is to verify space availability. This is accomplished by comparing space demand of the activity with the available space in the work blocks allocated to the activity.

The processor first determines total space demand for the activity (SD-1 and SD-2). Based on the resource demand pool and space demand of each resource, the processor computes SD-1 and SD-2 for the activity. The processor then compares space demand values of the activity to space availability values in the corresponding blocks and makes the space-based scheduling decision to schedule the activity or delay it to the next period.

The *Schedule Processor* is responsible for scheduling the activities once all constraints are verified and found satisfactory. The processor computes final start and finish dates for the activity and place all schedule results in the attribute of the 'output-schedule' slot of activity.

The *Resource Modification Processor* is responsible for modifying the activity resource demand pool if the selected activity duration is not the initial normal duration defined by the user.

The *Duration Processor* is responsible for computing the duration between any two floors based on the 'variable production' parameter defined for activity. If the activity is not influenced by the combined effect of travel time and learning curve, the processor computes the duration based on multiplying the floor duration by the number of floors. However, if the activity is affected by the travel time and learning curve, the processor use a step function to model the decrease in duration as a result of such effect.

Figure 8.14 depicts the processors of the sequence generation module and the order of their execution. The module is discussed in detail in Appendix C.

8.5 Chapter Summary

This chapter describes the overall methodology of the SCaRC prototype scheduling system developed under this research.

The chapter first presents the schedule generation approach adopted by the SCaRC

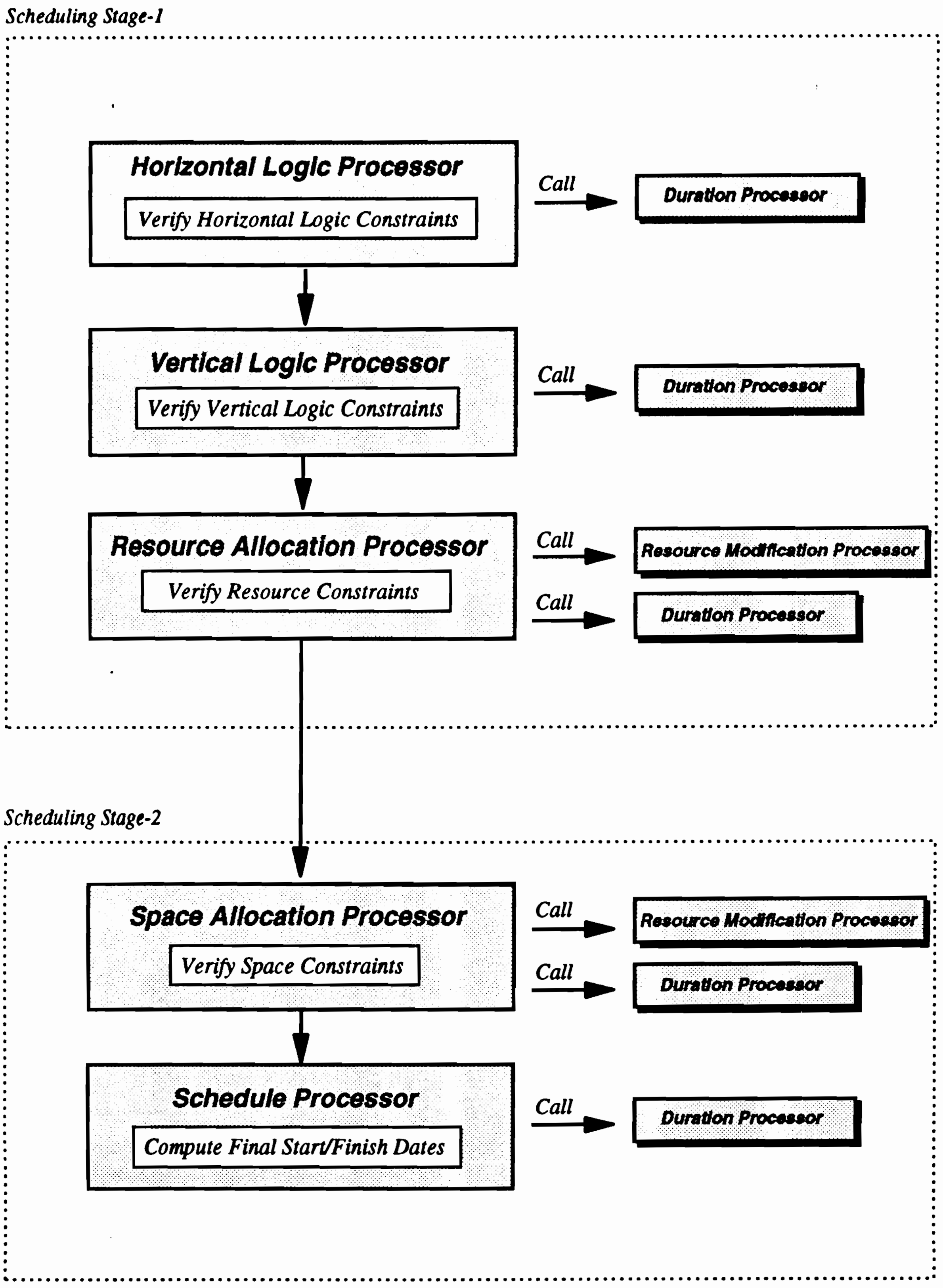

Scheduling Stage-1
Horizontal Logic Processor
Verify Horizontal Logic Constraints
Call
Duration Processor
Vertical Logic Processor
Verify Vertical Logic Constraints
Call
Duration Processor
Resource Allocation Processor
Verify Resource Constraints
Call
Resource Modification Processor
Call
Duration Processor
Scheduling Stage-2
Space Allocation Processor
Verify Space Constraints
Call
Resource Modification Processor
Call
Duration Processor
Schedule Processor
Compute Final Start/Finish Dates
Call
Duration Processor

Figure 8.14 - The Sequence Generation Module

system. Scheduling constraints considered in the production of the schedule are discussed. The different scheduling parameters that influence and control the production of the schedule are defined.

The chapter also presents an overall description of the structure of the system, its components, and their functions. The SCaRC system comprises of two main components:

1- A Database System, and

2- A Knowledge-Based System

The database system provides for defining all input data and for reviewing of output schedule results extracted from the knowledge-based system. The database system consists mainly of 10 data files linked to a user interactive menu system. Three algorithms are also developed as part of the database to manipulate data in the data files. The database system is discussed in detail in Chapter 9.

The second component of SCaRC is the knowledge-based system. The system is responsible for the actual production of the schedule. The system consists of:

1- Schemata Representational Structures, and

2- Scheduling Modules

The schemata representational structures allows to store input data and other data in the system. The scheduling modules are responsible for the reasoning process and the generation of the schedule. There are three modules: an *External Data Interface Module*, a *Controller Module*, and a *Sequence Generation Module*.

The knowledge-based system and the scheduling modules are discussed in detail in Appendices A, B, and C.

9.0 THE DATABASE SYSTEM

The database system is the first component of the SCaRC scheduling system. All data needed by the knowledge-based system (2nd component) is defined via the database system. The majority of the required data is input by the user. Other data, such as geometric data of design elements, is extracted directly from external ASCII type files. All data defined in the database system is transferred to the knowledge-based system for processing and generation of the schedule.

The database system also provides for retrieval of the generated schedule results from the knowledge-based system. The objective is to allow the user a greater flexibility to view the results of the generated schedule in a variety of output formats including a graphical format. The results of the generated schedule is transferred by the knowledge-based system to an output ASCII file. The file is extracted by the user from the database system and stored in a database DBF type file.

In addition to data input and output, the database system allows to define global space demand values for resources associated with scheduling multi-story buildings. Values to be defined should be based on space demand for unit resources. This allows to build some sort of a library of space demand data for generic resource types. This space demand data for different resources may be based on empirical equations, site observations, or historical data from previous similar projects. The data may be retrieved during user input. The objective is to assist the user in making feasible estimates while defining space demand values of resources for the project under consideration.

For the purpose of this research global space demand values for several resources are estimated as examples. Estimated values are approximate. Future work is required to define accurate values for all pertinent resources.

The database system is designed as an interactive menu-driven system. The system comprises mainly of a database file structure that consists of 10 database files linked to the menu system. Data stored in the database files is manipulated via a set of screens also linked to the menu system.

The database system also has several algorithms developed within the database environment that interact with the database files. There are three database algorithms. The objective of these algorithms is to manipulate data among the database files and between the database files and external files.

The file structure, system screens, menu system, and evaluation algorithms are developed using dBASE IV programming language. A complete listing of the program source code is provided in Appendix F.

The following sections describe the database system in more detail. The first section discusses the different categories of data within the database system. The remaining sections provide a detailed description of the database system file structure and algorithms.

9.1 Categories of Data

The database system contains six categories of data. As illustrated in Figure 9.1, these categories are as follows:

1- Global resource space demand system data

2- Activity input data

3- Resource input data

4- Work space input data

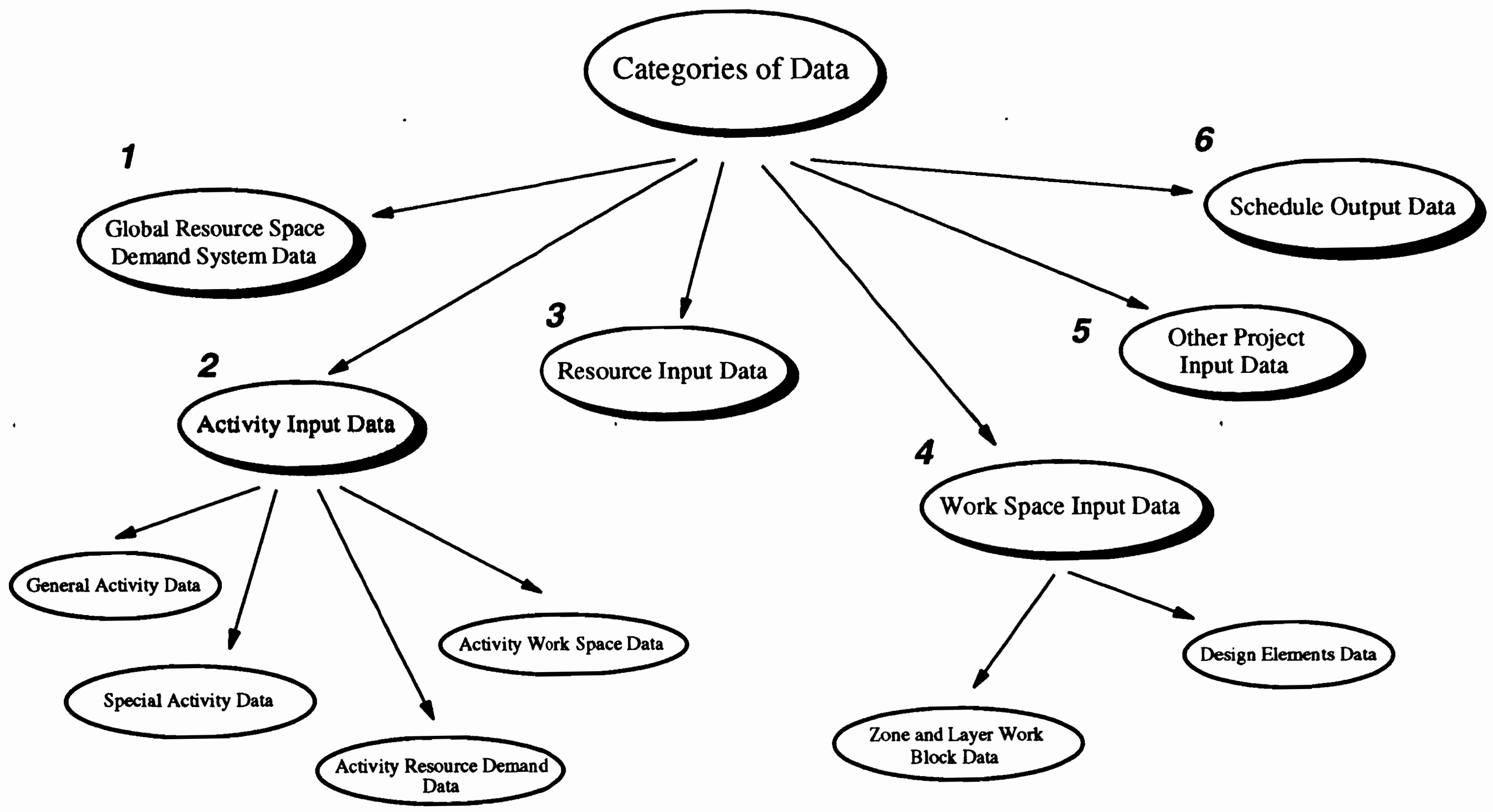


Figure 9.1 - Categories of Data in the Database System

5- Other project input data

6- Schedule results output data

The first category of data defines system data. Categories 2 through 5 comprises input data defined by the user or extracted directly by the system from external ASCII files. The last category of data defines output data of the generated schedule. The following subsections describe these categories in more detail.

9.1.1 Global Resource Space Demand System Data

This category of data is a system data and constitutes global space demand values for various resources associated with multi-story building construction. Values is defined based on unit resources. As indicated earlier, the objective of the global data is to assist the user to estimate space demand values for resources of the particular project under consideration.

The global resource space demand data are defined using two attributes:

- $S_{physical}$: *Global physical space demand value (all resource types)*
- $S_{surrounding}$: *Global surrounding space demand value (manpower/equipment only)*

9.1.2 Activity Input Data

This second category of data constitutes all data associated with the project activities. Four sub-categories of activity input data are present. As illustrated in Figure 9.1, these sub-categories are:

i) General Activity Data:

This consists of general activity scheduling data associated with general scheduling procedures. Data in this sub-category is based on the user defined typical floor logical network and initial schedule. General activity data is defined using seven attributes:

- *Activity number*
- *Activity description*
- *Normal activity duration*
- *Maximum activity duration*
- *Preceding activities*
- *Succeeding activities*
- *Early start value (initial schedule value)*
- *Late start value (initial schedule value)*

ii) Special Activity Data:

This sub-category of activity input data consists of special scheduling parameters associated with linear scheduling of multi-story buildings. These parameters are defined using six attributes:

- *Workflow direction*

- *Variable production*
- *Continuity class*
- *Maximum number of splits*
- *Vertical logic activity*
- *Vertical logic lag*

iii) Activity Resource Demand Data:

This sub-category of activity input data defines resource number, type and quantity required for each activity. Activity resource demand data are defined using three attribute:

- *Resource code*
- *Resource type*
- *Resource demand value*

iv) Activity Work Space Data:

This forth sub-category of activity input data defines work space related data necessary for the space-based scheduling decisions during the schedule generation of the schedule. Activity work space data is defined using three attributes:

- *Activity space demand class*
- *Manpower and equipment block number*
- *Material block number*

9.1.3 Resource Input Data

The third main category of data constitutes data specific to each resource of the project. Resource data consists of availability data and work space demand data associated with each resource. Resource data is defined using three attributes:

- *Available quantity*
- *$S_{physical}$: User defined physical space demand of resource*
- *$S_{surrounding}$: User defined surrounding space demand of resource*
- Total work space demand ($S_{physical}$ + $S_{surrounding}$)

Values of Total Space, $S_{physical}$, and $S_{surrounding}$ for manpower and equipment are based on unit resources. While same values for material are defined based on overall quantities.

9.1.4 Work Space Input Data

This forth category of data defines all the work space data associated with the project under consideration. Two types or sub-categories of work space data for the project are present:

i) <u>Zone and Layer Work Block Data</u>:

This first sub-category constitutes space availability data for different zone and layer work blocks of a typical floor. Two attributes are used to define work block data:

- *Zone and layer work block number*

- *Total work space available*

ii) Design Elements Data:

This second sub-category of data defines geometric data of each design element. Design elements and their geometric data are extracted from a CAD model of the typical floor. Four attributes are used to define this type of data:

- *Design element number*
- *Design element type*
- *Geometric data of design element*
- *Designated blocks*

The geometric data for each element is defined using different geometric parameters of the element (e.g. length, width, depth, surface area, thickness, etc.). These geometric parameters is extracted form the CAD model of the typical floor.

9.1.5 Other Project Input Data

This category of data defines other specific project data. Two attributes are used to define this data:

- *First typical floor number*
- *Last typical floor number*

9.1.6 Schedule Results Output Data

This last category of data constitutes the output data generated by the scheduling system. Output data is extracted from the knowledge-based system and stored in the database to allow the user to review the results.

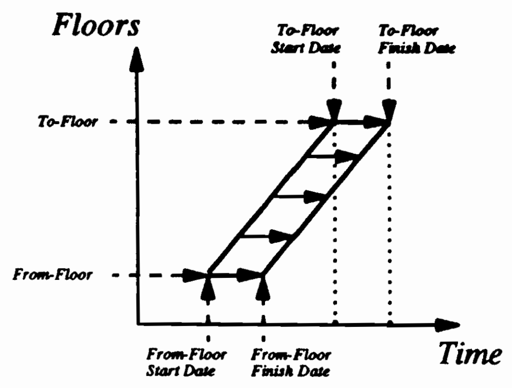
Eight attributes are used to define the schedule output data:

- *Activity number*
- *From-floor number*
- *To-floor number*
- *Duration*
- *From-floor start date*
- *From-floor finish date*
- *To-floor start date*
- *To-floor finish date*

If the activity is scheduled continuously, the *From-Floor/To-Floor* attributes will define the first and last typical floors in the building. For example, for a 20 story building and assuming that the work flow direction is upwards, then the from-floor number will be floor 1 and the to-floor number will be floor 20. This is illustrated in Figure 9.2.

If the activity is scheduled by splitting the activity one or more times along the typical floors, the From-floor/To-floor numbers attributes will define the start and end floors for

Total Floors = 20
First Typical Floor = 1
Last Typical Floor = 20



Activity Number		From-Floor	To-Floor	Duration *(Days)*	From-Floor Dates		To-Floor Dates	
					Start	*Finish*	*Start*	*Finish*
A01	(continuous)	1	20	10	0	10	190	200
A02	(split)	1	10	5	0	5	45	50
		11	20	6	50	56	104	110
A03	(split)	1	7	10	0	10	60	70
		8	14	12	70	82	142	154
		15	20	14	154	168	224	238

Figure 9.2 - Example Output Data

each scheduled segment of the activity. For example, if the same activity is split twice along the typical floors, then for the first section, the from-floor number will be floor 1 and the to-floor number will be floor 10. For the second section of the activity, the from-floor number will be floor 11 and the to-floor number will be floor 20. This is illustrated in Figure 9.2.

9.2 The Database File Structure

The database file structure consists of ten database type files (.dbf). Data stored in these files is manipulated by the user through a user interactive menu-driven system. The database files are linked to the menu system via a set of screens. There are 10 input/output screens. The screens are designed to allow the user to define input data (input screens) or view results of the generated schedule (output screens). The file structure consists of the following files:

1- RES-GLB.dbf

2- ACT-GEN-dbf

3- ACT-PAR.dbf

4- ACT-RES.dbf

5- ACT-SPC.dbf

6- RES.dbf

7- BLOCK.dbf

8- ELEMENT.dbf

9- OTHER.dbf

The *RES-GLB.dbf* data file contains the global space demand data for generic resource types. Values for several resources are defined as examples. As indicated earlier, future research is required to acquire such data. The file is accessed by the user from the menu system through input screen S-1. The *ACT-GEN.dbf* contains the general scheduling data for each activity. The file is accessed from the menu system through input screen S-2. The *ACT-PAR.dbf* data file will contain the special activity scheduling data associated with scheduling repetitive floors of multi-story buildings. The file is accessed from the main menu via input screen S-3. The *ACT-RES.dbf* data base file contains resource demand data for each activity. The file is accessed from the main menu through input screen S-4. The *ACT-SPC.dbf* data file contains work space data for each activity. The file is accessed from the menu system via input screen S-5.

Data file *RES.dbf* contains availability values and space demand data for all defined project resources. The file is accessed from the menu system via input screen S-6. Data file *BLOCK.dbf* contains work space availability data for all defined zone and layer work blocks. The *ELEMENT.dbf* data file contains geometric data for the design elements. Data in this file is automatically extracted from a CAD model data file. This file is accessed via input screen S-8 (view mode only). The database file *OTHER.dbf* contains other project data. The file is accessed via input screen S-9.

Results of the generated schedule is stored in the *OUTPUT.dbf* database file. Schedule results data stored in this file is extracted from an ASCII file generated by the knowledge-based scheduling system. Schedule results stored in this file is viewed by the user via output screen S-10.

The database files and input/output screens are manipulated via a user interactive menu system. Figure 9.3 depicts the overall menu structure of the database system. The menu system comprises of seven menus:

M-1	Main Menu
M-2	System Data Menu
M-3	Project Data Input Menu
M-4	Activity Data Input Menu
M-5	Resource Data Input Menu
M-6	Work Block Data Input Menu
M-7	Schedule Data Output Menu

9.2.1 The Main Menu (M-1)

The main menu of the system controls the overall menu structure of the database system. As illustrated in Figure 9.4, the main menu contains five options:

Option 1: Define System Data

Option 2: Clear All Input/Output Records

Option 3: Define Input Data

Option 4: View Output Data

Option 5: Exit to DOS

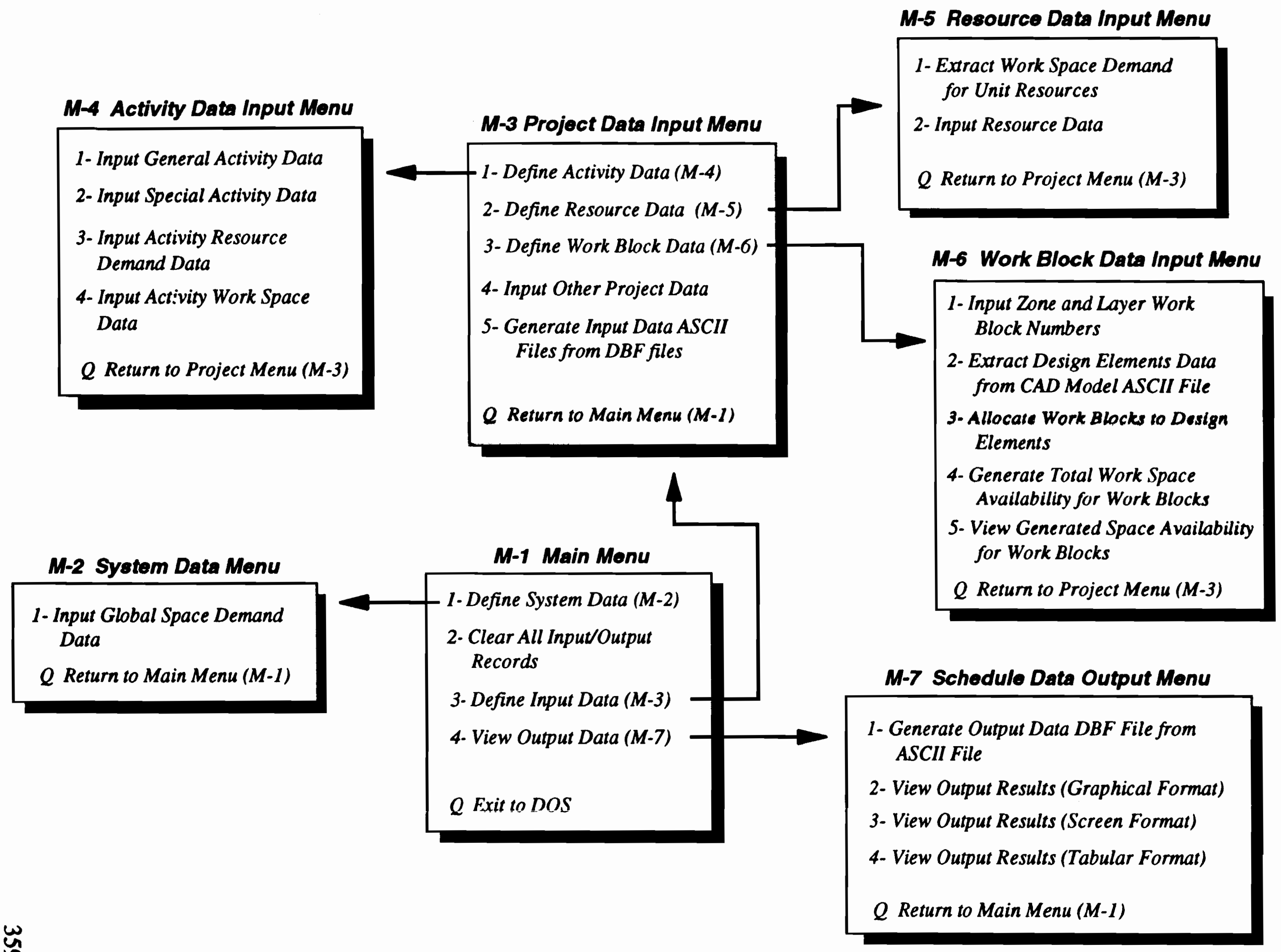

M-4 Activity Data Input Menu
1- Input General Activity Data
2- Input Special Activity Data
3- Input Activity Resource Demand Data
4- Input Activity Work Space Data
Q Return to Project Menu (M-3)
M-3 Project Data Input Menu
1- Define Activity Data (M-4)
2- Define Resource Data (M-5)
3- Define Work Block Data (M-6)
4- Input Other Project Data
5- Generate Input Data ASCII Files from DBF files
Q Return to Main Menu (M-1)
M-5 Resource Data Input Menu
1- Extract Work Space Demand for Unit Resources
2- Input Resource Data
Q Return to Project Menu (M-3)
M-6 Work Block Data Input Menu
1- Input Zone and Layer Work Block Numbers
2- Extract Design Elements Data from CAD Model ASCII File
3- Allocate Work Blocks to Design Elements
4- Generate Total Work Space Availability for Work Blocks
5- View Generated Space Availability for Work Blocks
Q Return to Project Menu (M-3)
M-2 System Data Menu
1- Input Global Space Demand Data
Q Return to Main Menu (M-1)
M-1 Main Menu
1- Define System Data (M-2)
2- Clear All Input/Output Records
3- Define Input Data (M-3)
4- View Output Data (M-7)
Q Exit to DOS
M-7 Schedule Data Output Menu
1- Generate Output Data DBF File from ASCII File
2- View Output Results (Graphical Format)
3- View Output Results (Screen Format)
4- View Output Results (Tabular Format)
Q Return to Main Menu (M-1)

Figure 9.3 - The Database System Menu Structure

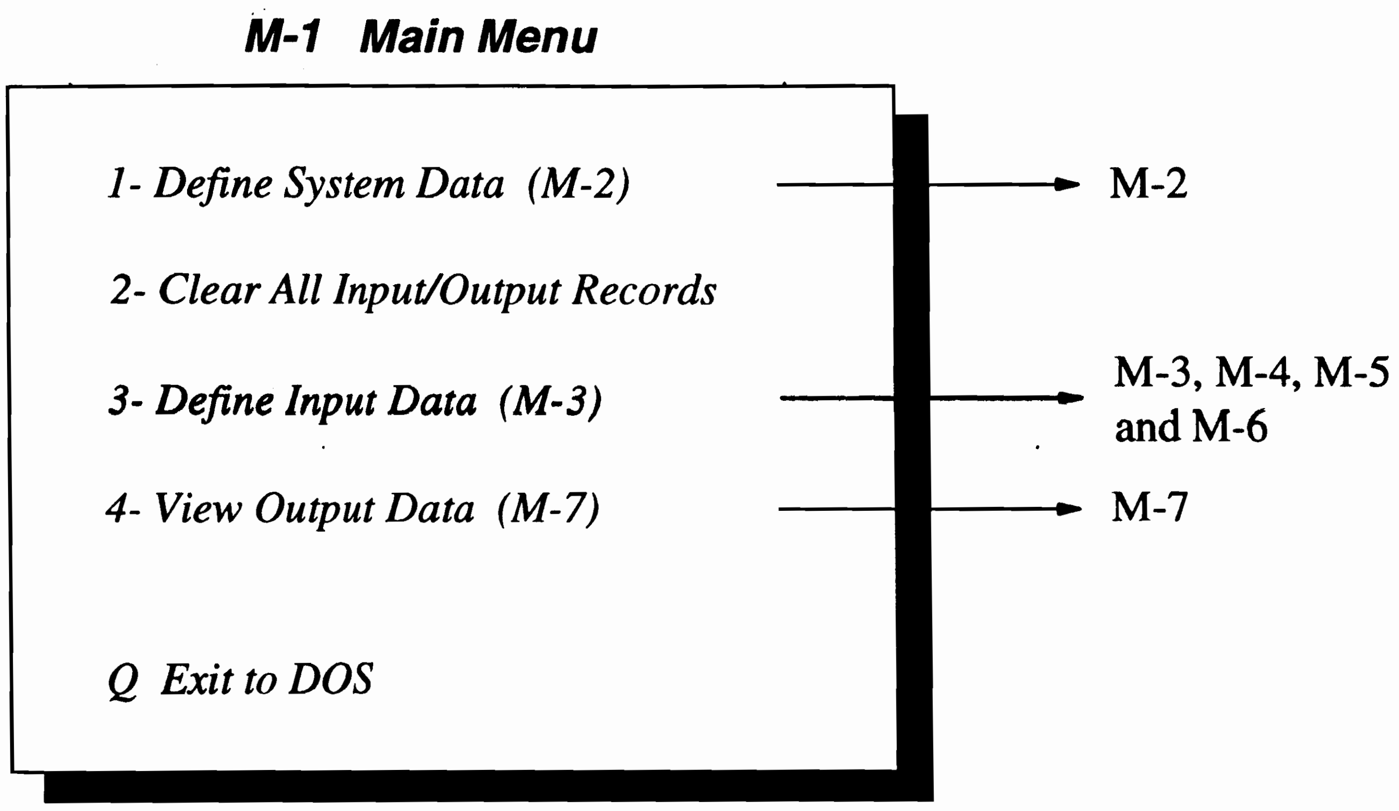

M-1 Main Menu
1- Define System Data (M-2)
2- Clear All Input/Output Records
3- Define Input Data (M-3)
4- View Output Data (M-7)
Q Exit to DOS
M-2
M-3, M-4, M-5 and M-6
M-7

Figure 9.4 - The Main Menu

• Selecting option 1 *Define System Data* from the main menu transfers control to the system data menu (M-2). The system menu is used to define global space demand values for unit resources. As illustrated in Figure 9.3, the menu contains two options. The first option allows the user to define and modify global space demand values for different generic construction resources. The second option allows the user to return to the main menu.

• Selecting option 2 *Clear All Input/Output Data Records* from the main menu deletes input data from all database files (except for RES-GLB.dbf file) from existing data. This process is necessary before the user proceeds to input data for a new project.

• Selecting option 3 *Define Input Data* from the main menu transfers control to the project data input menu (M-3). As illustrated in Figure 9.3, the project data menu includes several options to define and generate all necessary input data required by the scheduling system for the generation of the schedule.

• Selecting the forth option *View Output Data* from the main menu transfers control to the schedule data output menu (M-7). The output menu includes several options to allow the user to view the results of the schedule.

• The last option in the main menu *Exit to DOS* exits the user from the database system to the DOS environment.

9.2.2 The System Data Menu (M-2)

The system data menu allows the user to define work space demand data for different generic construction resources. The menu also allow to modify or update existing space demand data. The system data menu is activated by selecting option 1 of the main menu. As illustrated in Figure 9.5, the system data menu contains 2 options:

Option 1: Input/Modify Global Space Demand Data

Option 2: Return to Main Menu (M-1)

• Selecting the first option activates data input screen (S-1) shown in Figure 9.6. The input screen allows the user to define space demand data for new resources or modify space data for existing resources. As shown in Figure 9.6, the input screen contains three fields:

1- Resource Code

2- $S_{physical}$ space demand value

3- $S_{surrounding}$ space demand value (manpower/equipment only)

The *resource code* field allows to input an alphanumeric number for each resource. The *$S_{physical}$, and $S_{surrounding}$* fields define the two space demand values for the unit resource. Values of $S_{surrounding}$ are defined for manpower and equipment only.

Records of global space demand data for different resources are stored in the **RES-GLB.dbf** database file.

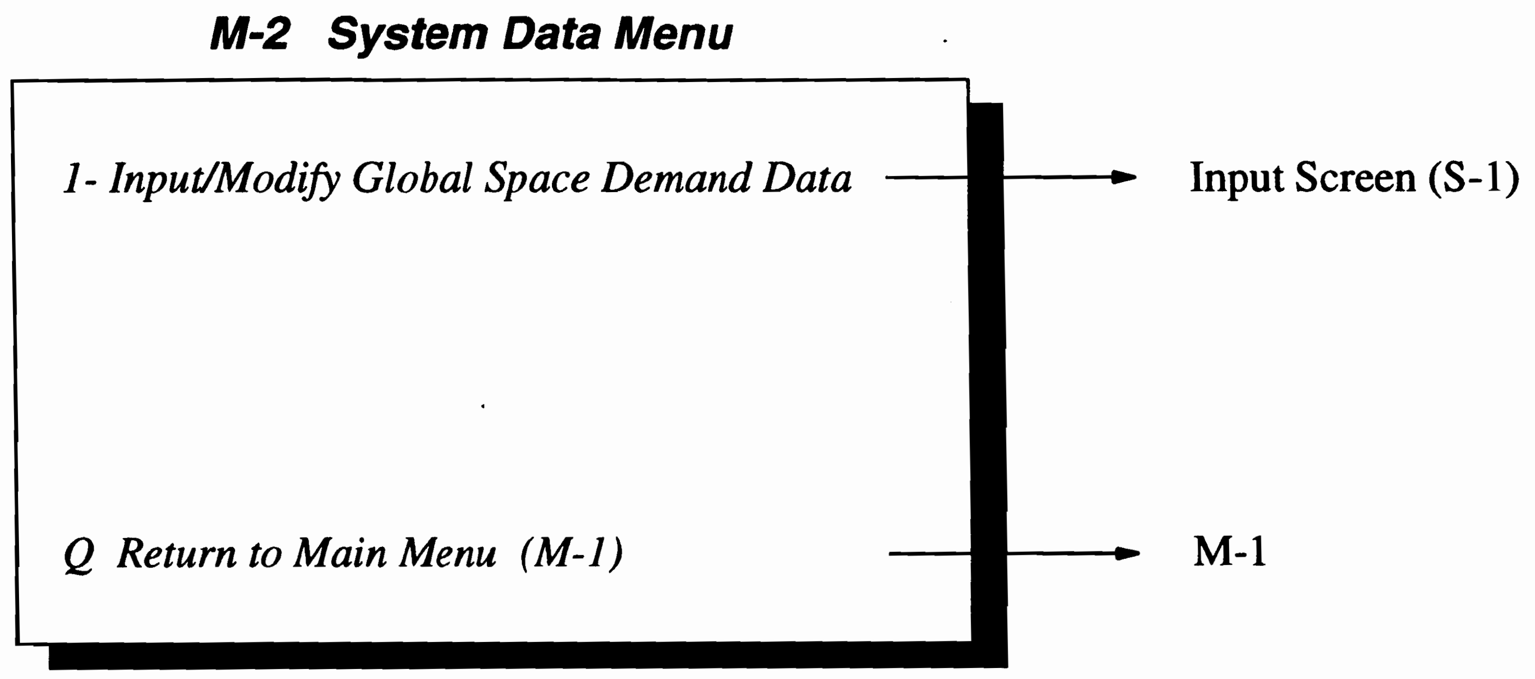

M-2 System Data Menu
1- Input/Modify Global Space Demand Data
Input Screen (S-1)
Q Return to Main Menu (M-1)
M-1

Figure 9.5 - The System Data Menu

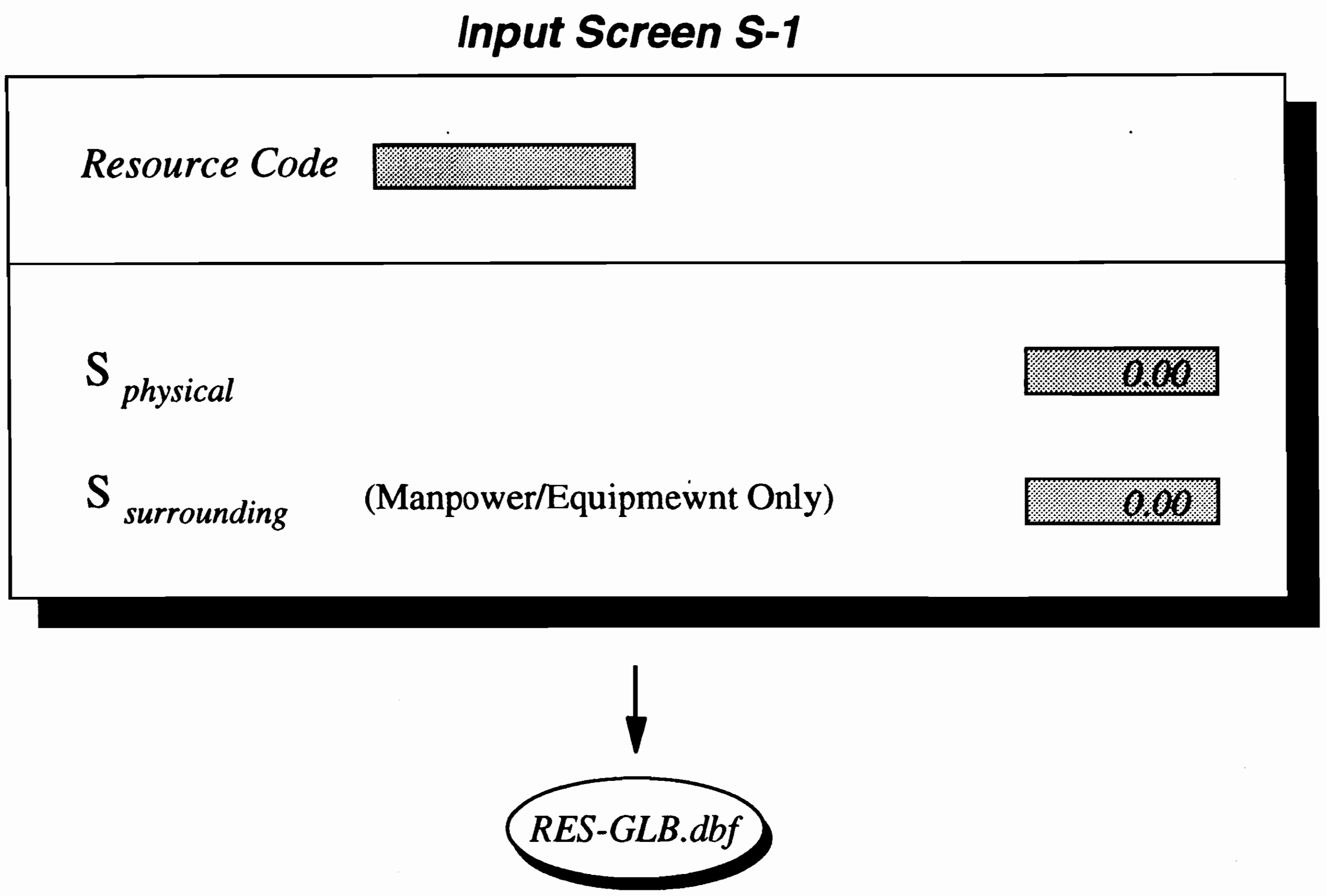


Figure 9.6 - Input Screen for Global Resource Space Demand Data (S-1)

• Selecting the third option of the system data menu transfers control back to the main menu (M-1).

9.2.3 The Project Data Input Menu (M-3)

The project data input menu allows the user to define and generate all necessary project data required for the knowledge-based system to generate a schedule. The project data menu is activated by selecting option 3 from the main menu. As illustrated in Figure 9.7, the project input data menu contains 6 options:

Option 1: Define Activity Data	(M-4)
Option 2: Define Resource Data	(M-5)
Option 3: Define Work Block Data	(M-6)
Option 4: Input Other Project Data	
Option 5: Generate Input Data ASCII Files from DBF Files	
Option 6: Return to Main Menu	(M-1)

• Selecting option 1 *Define Activity Data* from the project menu transfers control to the activity data input menu (M-4). This menu includes several options to define and generate all activity related data via four input screens (S-2, S-3, S-4, and S-5). The activity data input menu and its input screens are discussed in section 9.2.4.

Selecting option 2 *Define Resource Data* from the project menu transfers control to the resource data input menu (M-5). This menu includes several options to define resource

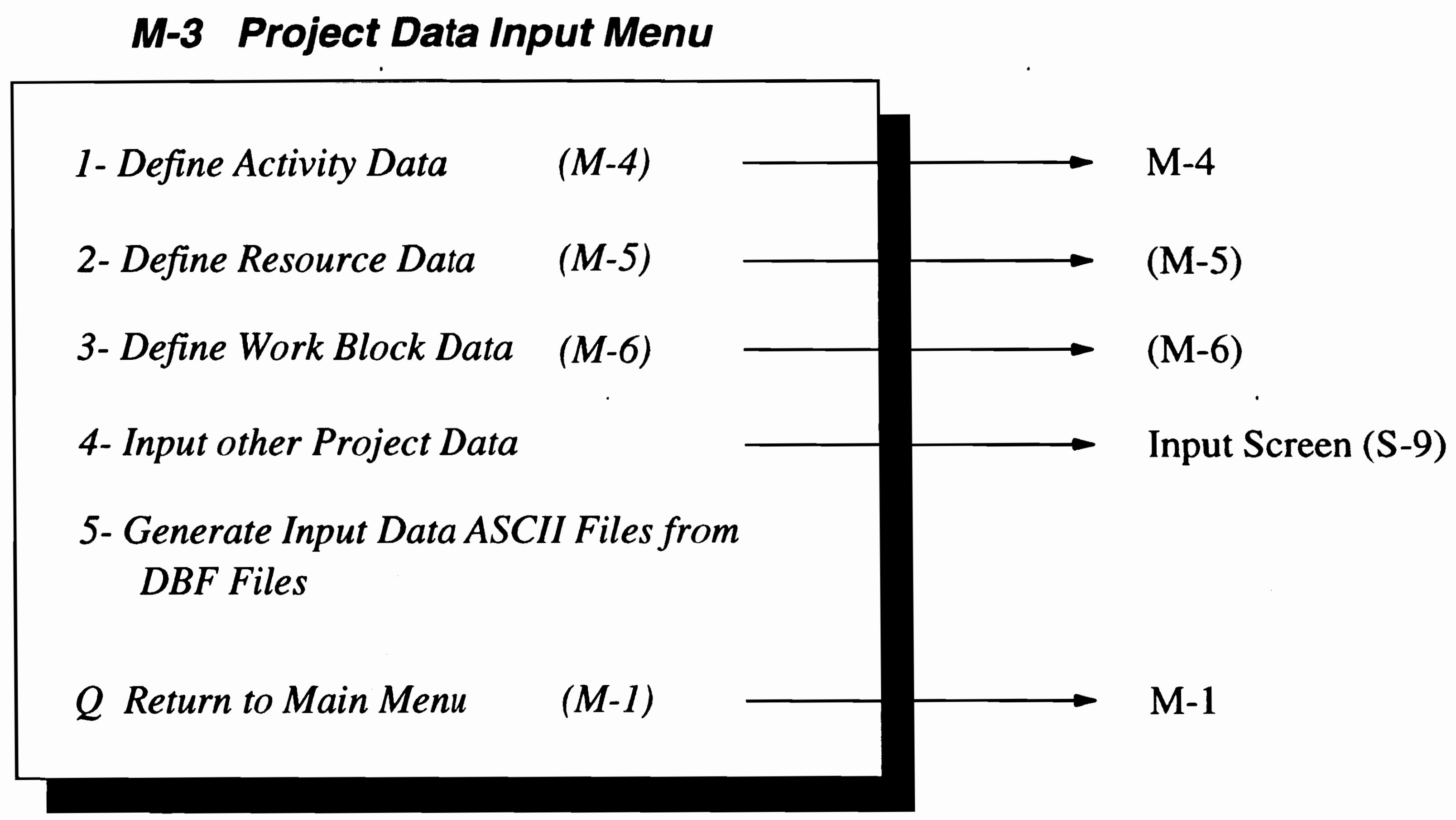

M-3 Project Data Input Menu
1- Define Activity Data (M-4)
2- Define Resource Data (M-5)
3- Define Work Block Data (M-6)
4- Input other Project Data
5- Generate Input Data ASCII Files from DBF Files
Q Return to Main Menu (M-1)
M-4
(M-5)
(M-6)
Input Screen (S-9)
M-1

Figure 9.7 - The Project Data Input Menu

availability values and work space demand values via input screen S-6. The resource data input menu and input screen are discussed in section 9.2.5.

• Selecting option 3 *Define Work Block Data* from the project input data menu transfers controlto the work block data input menu (M-6). This menu includes several options to define and generate work space availability data via input screens S-7 and S-8. The work block input data menu and its input screens are discussed in more detail in section 9.2.6.

• Selecting option 4 *Input Other Project Data* from the project menu activates input screen S-9 shown in Figure 9.8. The screen allows the user to input first and last typical floor numbers of the project.

Records containing other project data are stored in the **OTHER.dbf** database file.

• Selecting option 5 *Generate Input Data ASCII Files from DBF Files* from the project menu generates ASCII type files from some of the database DBF files. Data in the generated ASCII files is retrieved by the knowledge-based system for processing and generation of the schedule. This option is discussed in section 9.4.

• Selecting the last option of the project input data menu transfers control back to the main menu (M-1).

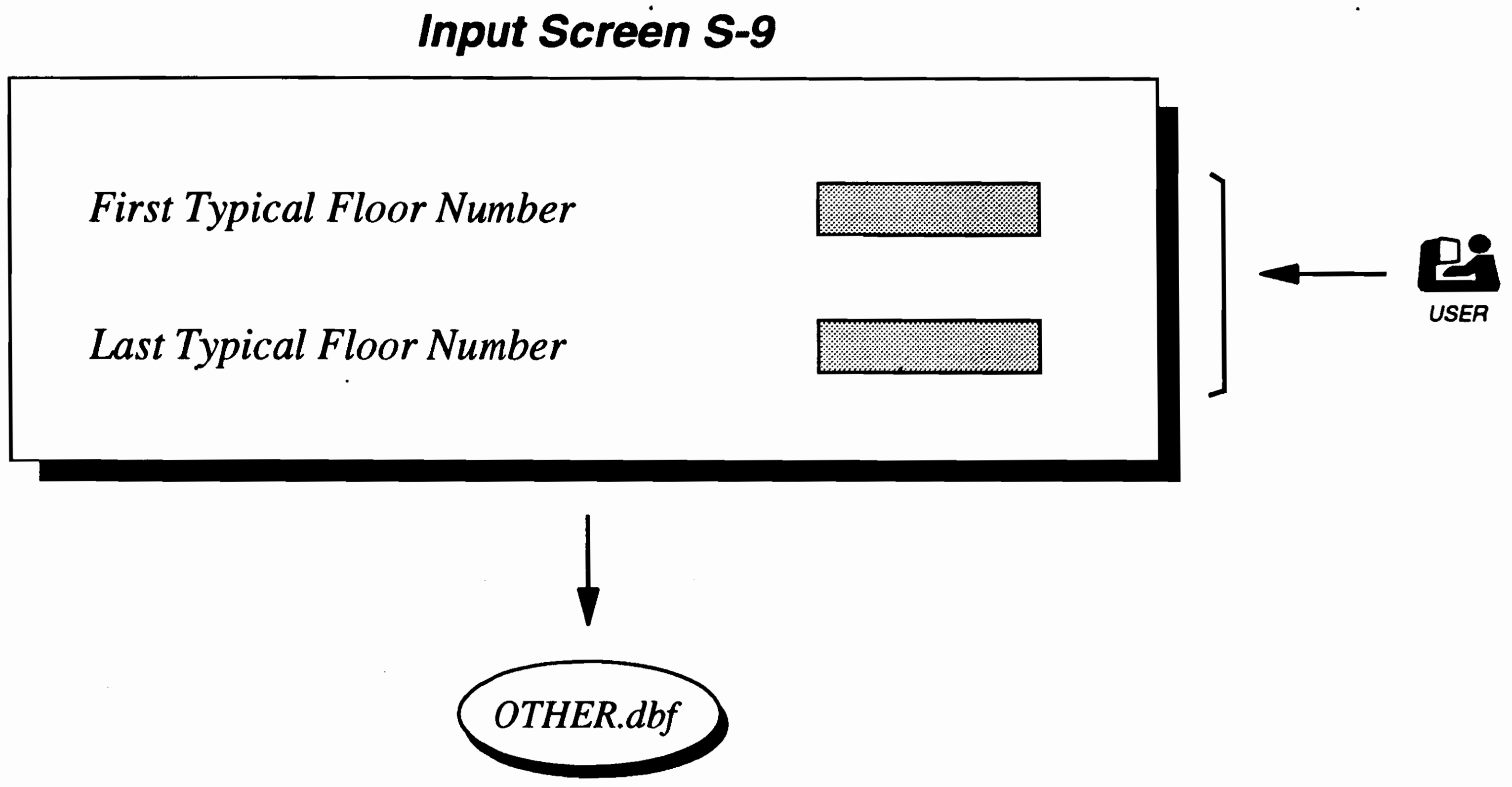


Figure 9.8 - Input Screen for Other Project Data (S-9)

9.2.4 The Activity Data Input Menu (M-4)

The activity data input menu allows the user to define and generate all data related to each activity of the project. The menu is activated by selecting option 1 of the project data input menu (M-3). As illustrated in Figure 9.9, the activity data input menu contains five options:

Option 1: Input General Activity Data

Option 2: Input Special Activity Data

Option 3: Input Activity Resource Demand Data

Option 4: Input Activity Work Space Data

Option 5: Return to Project Menu (M-3)

• Selecting option 1 *Input General Activity Data* from the activity menu activates input screen S-2 shown in Figure 9.10. Input screen S-2 allows the user to define general activity scheduling data. The screen contains eight fields:

1- Activity Number

2- Activity Description

3- Normal Activity Duration

4- Maximum Activity Duration

5- Preceding Activities

6- Succeeding Activities

7- Early Start Value (ES)

8- Late Start Value (LS)

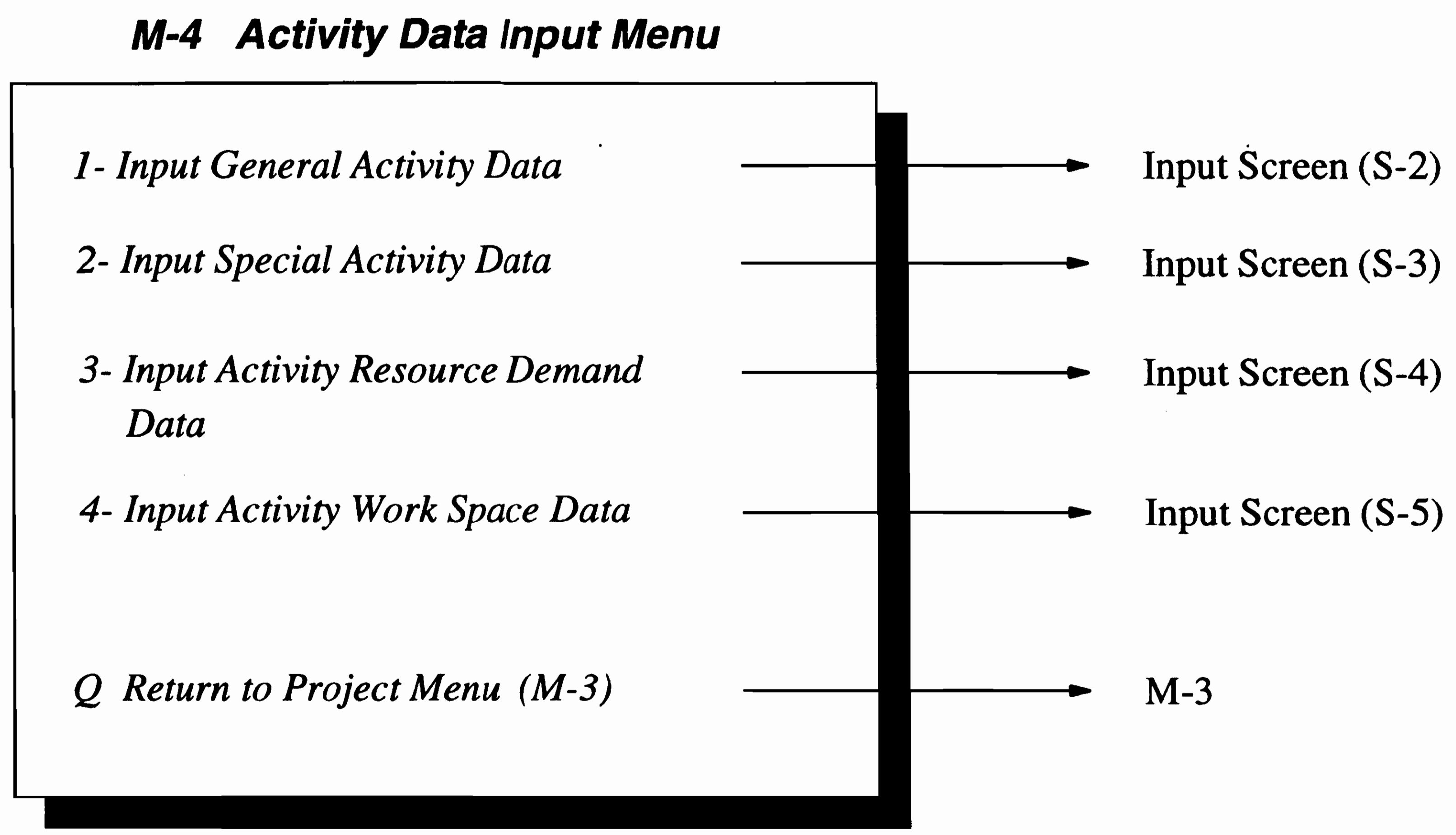

M-4 Activity Data Input Menu
1- Input General Activity Data
2- Input Special Activity Data
3- Input Activity Resource Demand Data
4- Input Activity Work Space Data
Q Return to Project Menu (M-3)
Input Screen (S-2)
Input Screen (S-3)
Input Screen (S-4)
Input Screen (S-5)
M-3

Figure 9.9 - The Activity Data Input Menu

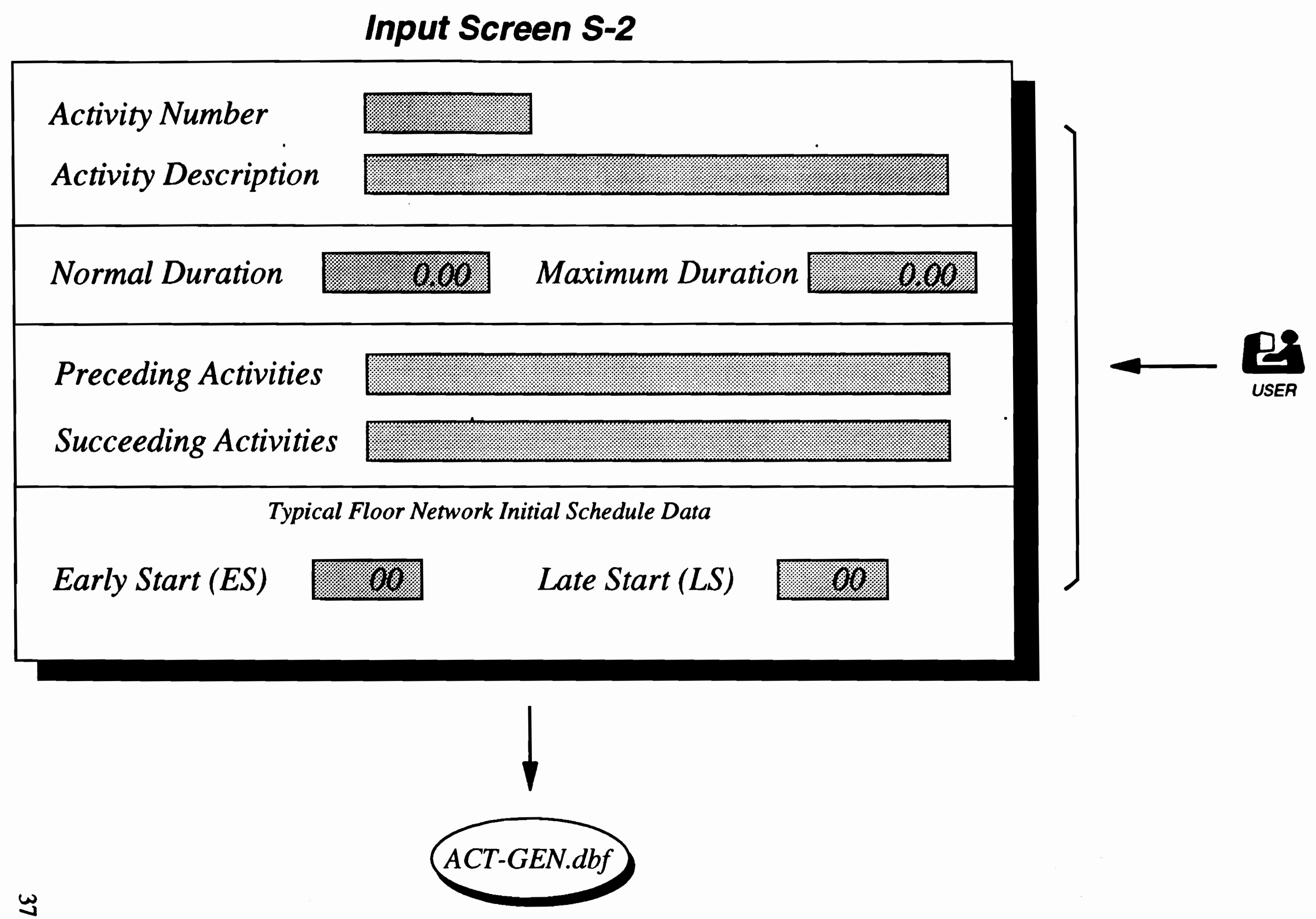


Figure 9.10 - Input Screen for General Activity Data (S-2)

The *activity number* field contains an alphanumeric number for each activity. The *activity description* field contains a description for each activity. The *normal and maximum duration* fields contains the normal and maximum duration values for each activity. The *preceding and succeeding activity* fields contains the numbers for all preceding and succeeding activities to the current activity based on the logical network of a typical floor. If an activity has no preceding or succeeding activities, the value 'none' should be entered. The *early start and late start* fields contains the ES and LS values for each activity based on an initial schedule for a typical floor. All data values in the eight fields are input by the user.

General activity data records are stored in the **ACT-GEN.dbf** database file.

• Selecting option 2 *Input Special Activity Data* from the activity menu activates input screen S-3 shown in Figure 9.11. This screen allows the user to define activity linear scheduling parameters data associated with scheduling this type of repetitive work. As illustrated in Figure 9.11, input screen S-3 contains eight fields:

1- Activity Number

2- Activity Description

3- Workflow Direction

4- Variable Production

5- Continuity Class

6- Maximum Number of Splits

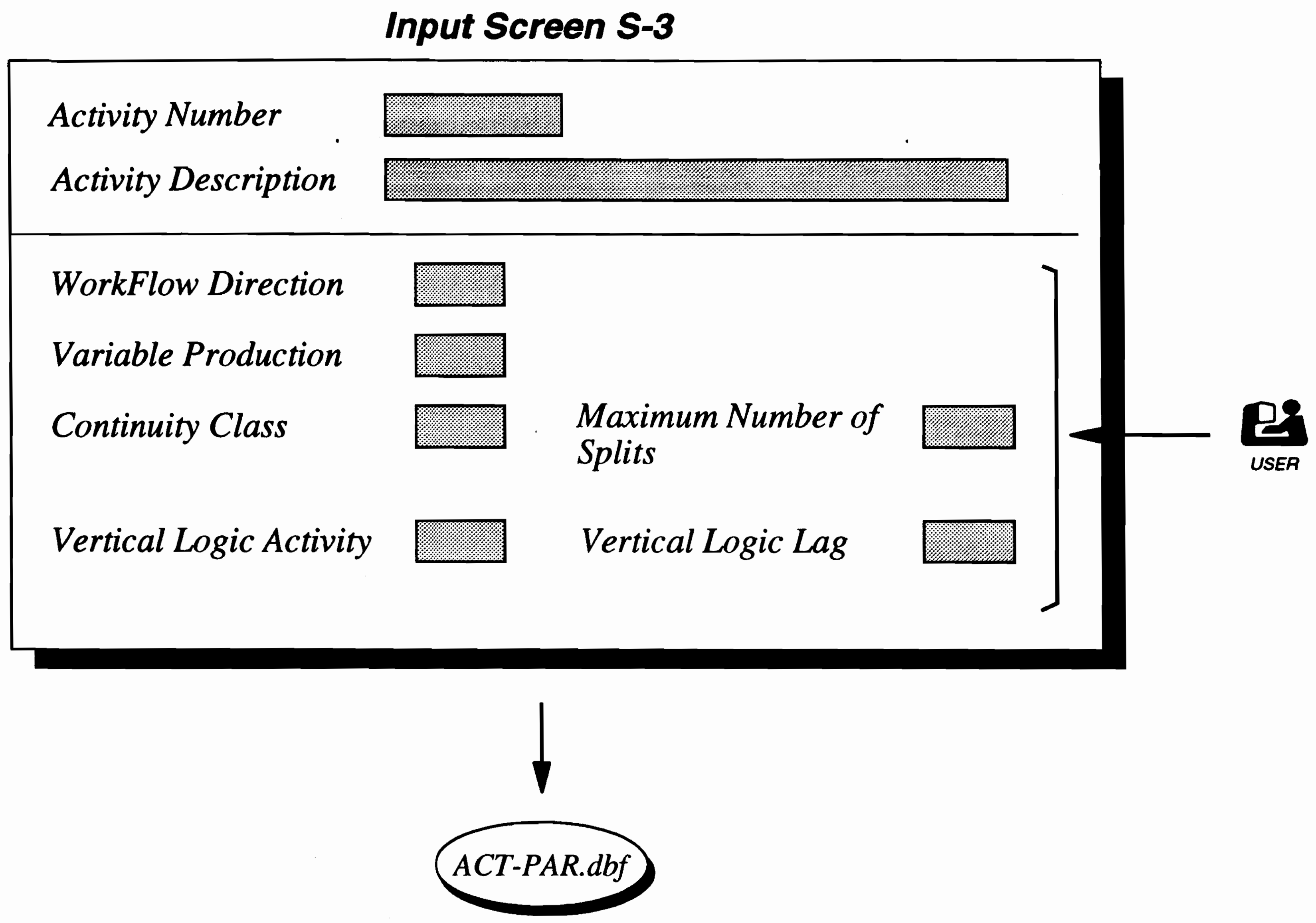


Figure 9.11 - Input Screen for Special Activity Data (S-3)

7- Vertical Logic Activity

8- Vertical Logic Lag

The *activity number* and *activity description* fields contain a number and description for each activity. The *work flow direction* field contains the work flow direction parameter value for the activity. Two values are used: up or down. The *variable production* field contains the variable production parameter value for the activity. This parameter indicates whether the activity duration will vary as the work progresses upwards or downwards through the building as a result of learning curve and increased travel time. Two values are used: yes or no. The *continuity class* field contains the continuity class parameter value for the activity. Two values are used: A or B. The *maximum number of splits* field contains a value for the maximum number of times an activity can be split along the typical floors. A value should be entered in this field only for activities with continuity class B. The *vertical logic activity* field contains the number of the activity forcing the vertical logic constraint. The *vertical logic lag* field contains the floor lag defining the vertical constraint.

The activity number and activity description fields are automatically generated by the system based on activity records defined by the user when selecting option 1 of the activity menu. The other six fields are input by the user for each activity record generated. Activity records containing the special activity data are stored in the **ACT-PAR.dbf** database file.

• Selecting option 3 *Input Activity Resource Demand Data* from the activity menu will activate input screen S-4 shown in Figure 9.12. This screen allows the user to define all resource demand data for each activity. The screen contains fourteen fields:

1- Activity Number

2- Activity Description

3- Resource Code (4 fields)

4- Resource Type (4 fields)

5- Resource Quantity (4 fields)

The first two fields contain the code number and description for the activity. The *resource code* fields contain alphanumeric numbers for the resources. The *resource type* fields contain the type of each resource (i.e. manpower, equipment or material). The *resource quantity* fields contain the required quantity values of the resources.

The activity number and activity description fields are automatically generated by the system based on activity records defined by the user when selecting option 1 of the activity menu. The other twelve fields are input by the user for each activity record generated.

Activity records containing activity resource demand data are stored in the **ACT-RES.dbf** database file.

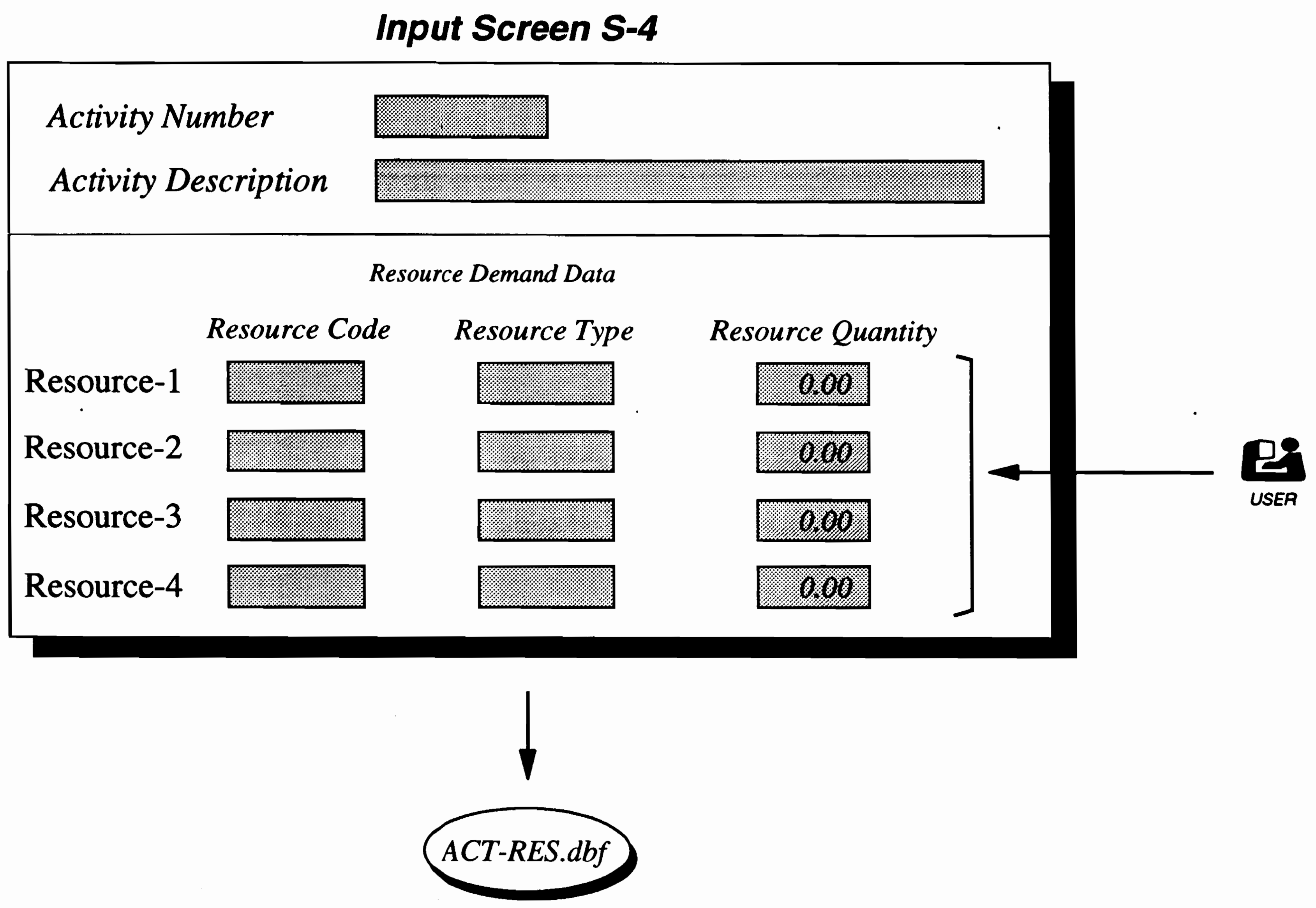


Figure 9.12 - Input Screen for Activity Resource Demand Data (S-4)

• Selecting option 4 *Input Activity Work Space Data* from the activity menu activates input screen S-5 shown in Figure 9.13. This screen allows the user to define specific work space data for each activity. As illustrated in Figure 9.13, the screen contains five fields:

1- Activity Number

2- Activity Description

3- Activity Space Demand Class

4- Manpower and Equipment Work Block Number

5- Material Work Block Number

The first two fields contain the number and description for the activity. The *activity space demand class* field contains the class of space demand for each activity. Three values for this field are used: A, B, or C. The *manpower and equipment work block number* field contains the number of the work block designated for the activity's manpower and equipment. Similarly, the *material block number* field will contain the code number of the work block designated for the activity's material if stored in the floor area. Because the material may be stored on the floor in a different work area from that where the actual activity is executed, two fields are used for allocating the activity to a work block. If the material is stored in the same work area where the activity is executed, both fields will contain the same block number. If the material is stored outside the floor area, no value should be entered in the material block number field.

The activity number and activity description fields are automatically generated by the

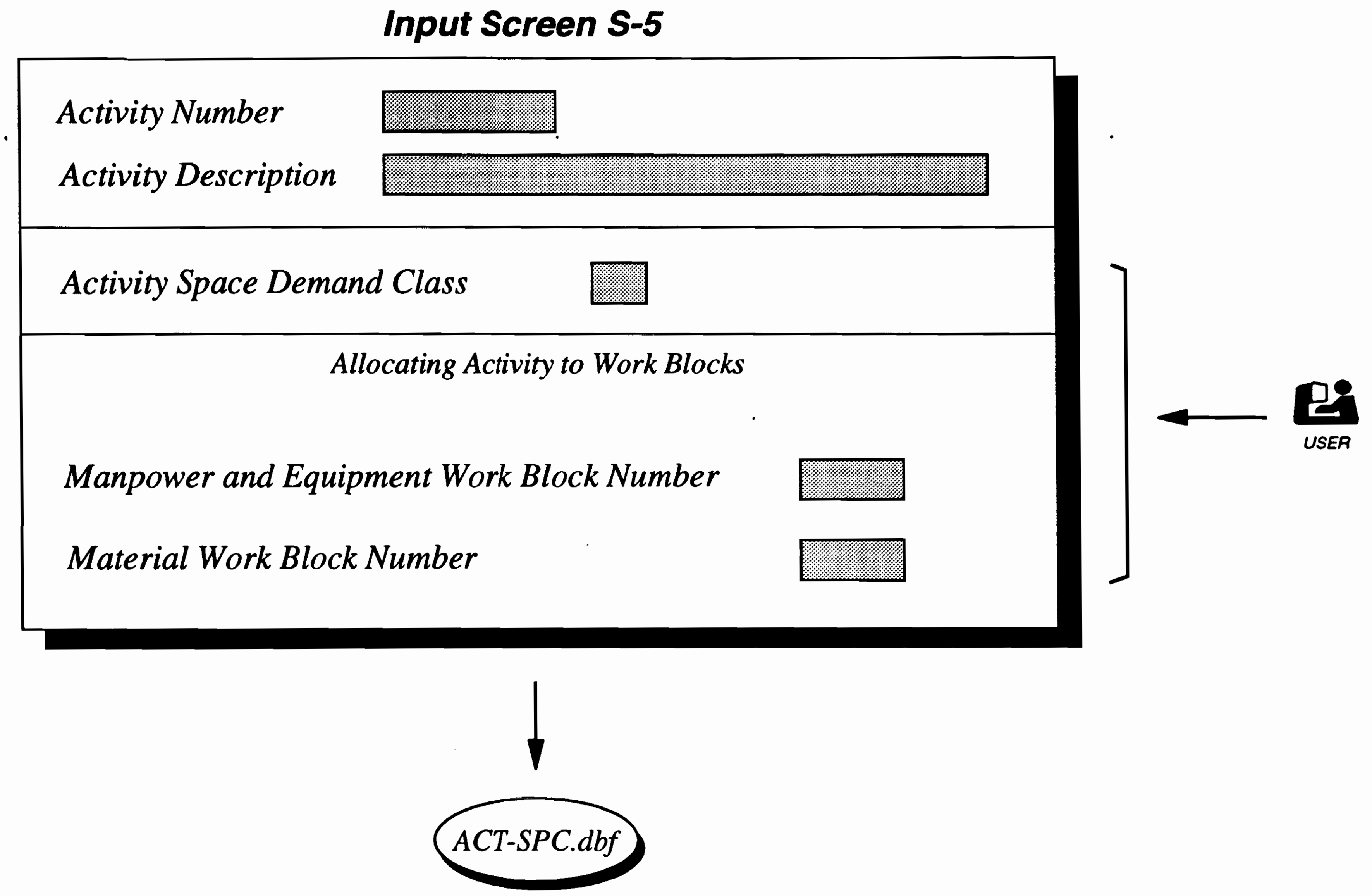

Input Screen S-5
Activity Number
Activity Description
Activity Space Demand Class
Allocating Activity to Work Blocks
Manpower and Equipment Work Block Number
Material Work Block Number
USER
ACT-SPC.dbf

Figure 9.13 - Input Screen for Activity Work Space Data (S-5)

system based on activity records defined by the user when selecting option 1 of the activity menu. The other three fields are input by the user for each activity record generated.

Activity records containing activity work space data are stored in the **ACT-SPC.dbf** database file.

• Selecting the last option of the activity input menu transfers control back to the project data menu (M-3).

9.2.5 The Resource Data Input Menu (M-5)

The objective of the resource data menu is to allow the user to define and generate all data related to each resource of the project. The menu is activated by selecting option 2 of the project data input menu (M-3). As illustrated in Figure 9.14, the resource data menu contains 3 options:

Option 1: Extract Work Space Demand for Unit Resources

Option 2: Input Resource Demand Data

Option 3: Return to Project Menu (M-3)

• Selecting option 1 *Extract Work Space Demand for Unit Resources* from the resource menu activates the data extraction algorithm-1. For each resource code defined, the algorithm extracts the global space demand values of the unit resource from the global

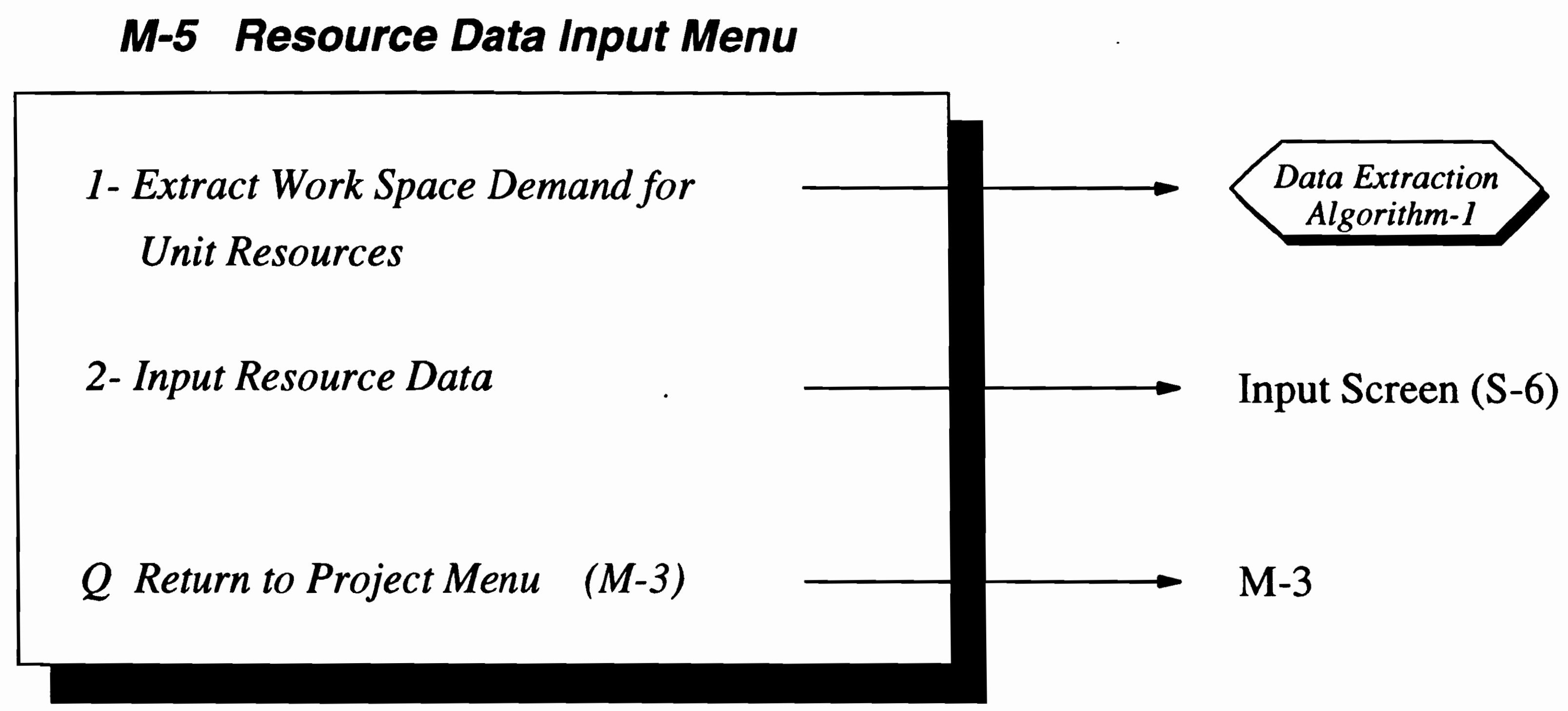

M-5 Resource Data Input Menu
1- Extract Work Space Demand for Unit Resources
2- Input Resource Data
Q Return to Project Menu (M-3)
Data Extraction Algorithm-1
Input Screen (S-6)
M-3

Figure 9.14 - The Resource Data Input Menu (M-5)

RES-GLB.dbf database file and store it in the appropriate fields in the RES.dbf file. The algorithm is discussed in section 9.3.

• Selecting option 2 *Input Resource Data* from the resource menu allows the user to define available quantities and space demand values for each resource. Selecting this option activates input screen S-6 shown in Figure 9.15. Screen S-6 contains eight fields:

1- Resource Code

2- Resource Type

3- Available Quantity

4- Global $S_{physical}$ space demand value for unit resource

5- Global $S_{surrounding}$ space demand value for unit resource (manpower/equipment only)

6- User defined $S_{physical}$ space demand value for resource

7- User defined $S_{surrounding}$ space demand value for resource (user defined)

8- Total space demand value for resource

The *resource code and type* fields contain the number and type of the resource. The *available quantity* field contains the maximum daily quantity value that can be made available to the project. The *global space demand values* fields define the values of $S_{physical}$ and $S_{surrounding}$ based on the unit resource. These values are extracted automatically from the global file RES-GLB.dbf and is stored in these two fields when selecting option 1 of this menu. This is accomplished by a data extraction algorithm-1. The algorithm is activated by selecting option 1 of the resource menu.

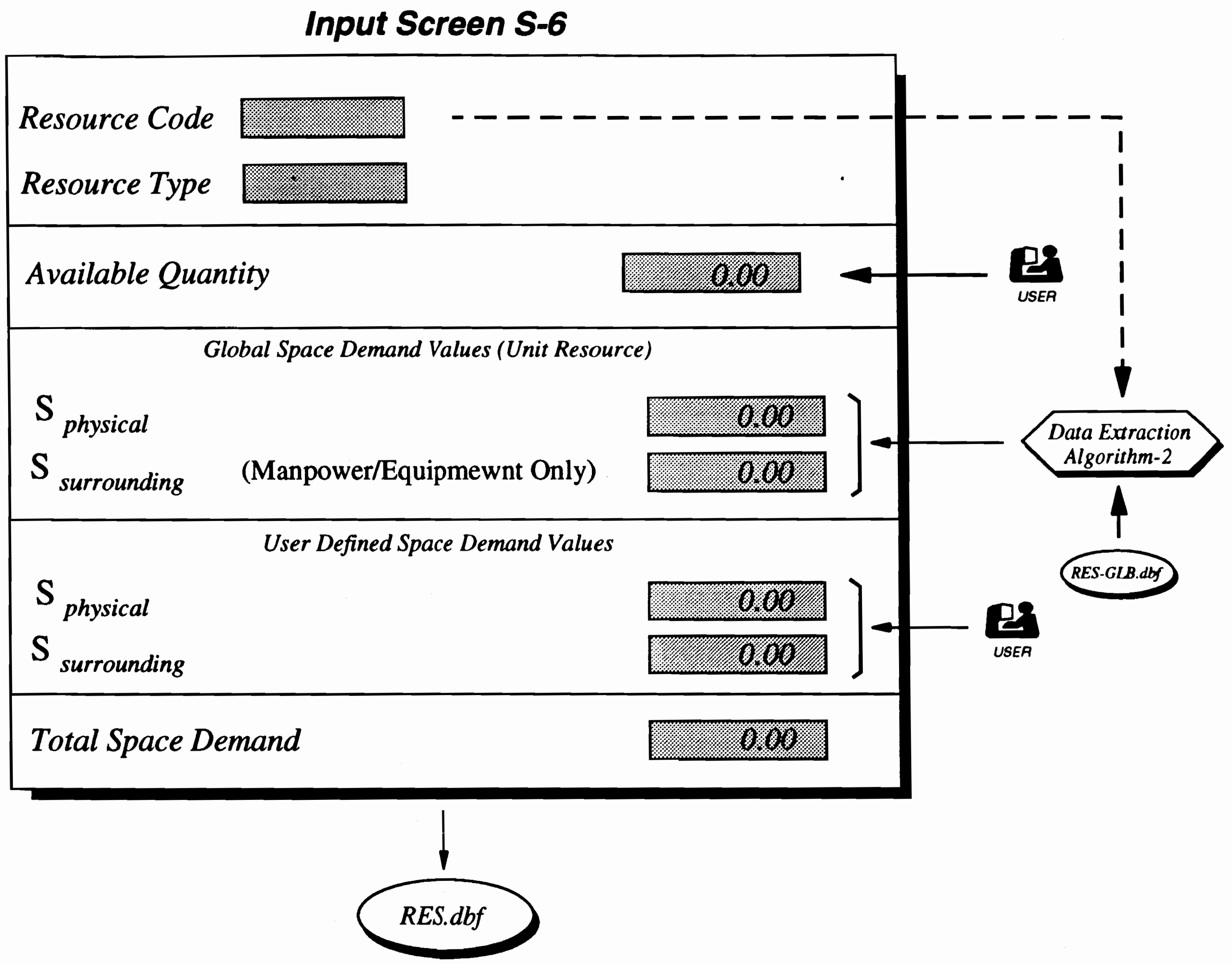


Figure 9.15 - Input Screen for Resource Data (S-6)

The *user defined space demand values* fields define the space demand for each resource based on user estimate. Estimated values for manpower and equipment are based on unit resource quantities while defined values for material are based on total quantity. This is because initial manpower and equipment quantities are defined for normal duration and may change during the scheduling process if the activity duration is adjusted. Once quantities are modified, total space demand for manpower and equipment is computed by multiplying the demand value for the unit resource (defined here) times the final quantity determined.

On the other hand, material quantities will not change as the activity duration is adjusted. Because of the different storage methods and arrangements of material in the floor, it is not possible to define total space demand of material by multiplying the unit value by the total quantity. Rather, it is more practical to define $S_{physical}$ and $S_{surrounding}$ values based on total quantity used, and taking into consideration other issues such as: storage characteristics of materials, and size of work area used for storage, etc. The unit demand values for $S_{physical}$ provided by the global file may assist the user in estimating the total physical space value for all materials used. It should be noted that if any material resource used is not stored on the work floor, a zero value should be entered in the storage space demand field for that material.

The *Total Space Demand* field defines the sum of the $S_{physical}$ and $S_{surrounding}$ values estimated by the user. Records containing resource data of screen S-6 will be stored in the **RES.dbf** database file.

• Selecting the last option from the resource menu transfers control back to the project data input menu (M-3).

9.2.6 The Work Block Data Input Menu (M-6)

The objective of this menu is to allow the user to define and generate work space availability data for different zone and layer work blocks of the typical floor. The menu is activated by selecting option 3 of the project data menu (M-3). As illustrated in Figure 9.16, the work block data input menu contains five options:

Option 1: Input Zone and Layer Work Block Numbers

Option 2: Extract Design Elements Data from CAD Model ASCII File

Option 3: Allocate Work Blocks to Design Elements

Option 4: Generate Total Work Space Availability for Work Blocks

Option 5: View Generated Space Availability for Work Blocks

Option 6: Return to Project Data Input Menu (M-3)

Selecting option 1 *Input Zone and Layer Work Block Numbers* from the work block data input menu activates input screen S-7 shown in Figure 9.17. This screen allows the user to define block numbers and total space availability for each work block. As illustrated in Figure 9.17, the screen contains two fields:

1- Zone and Layer Block Number

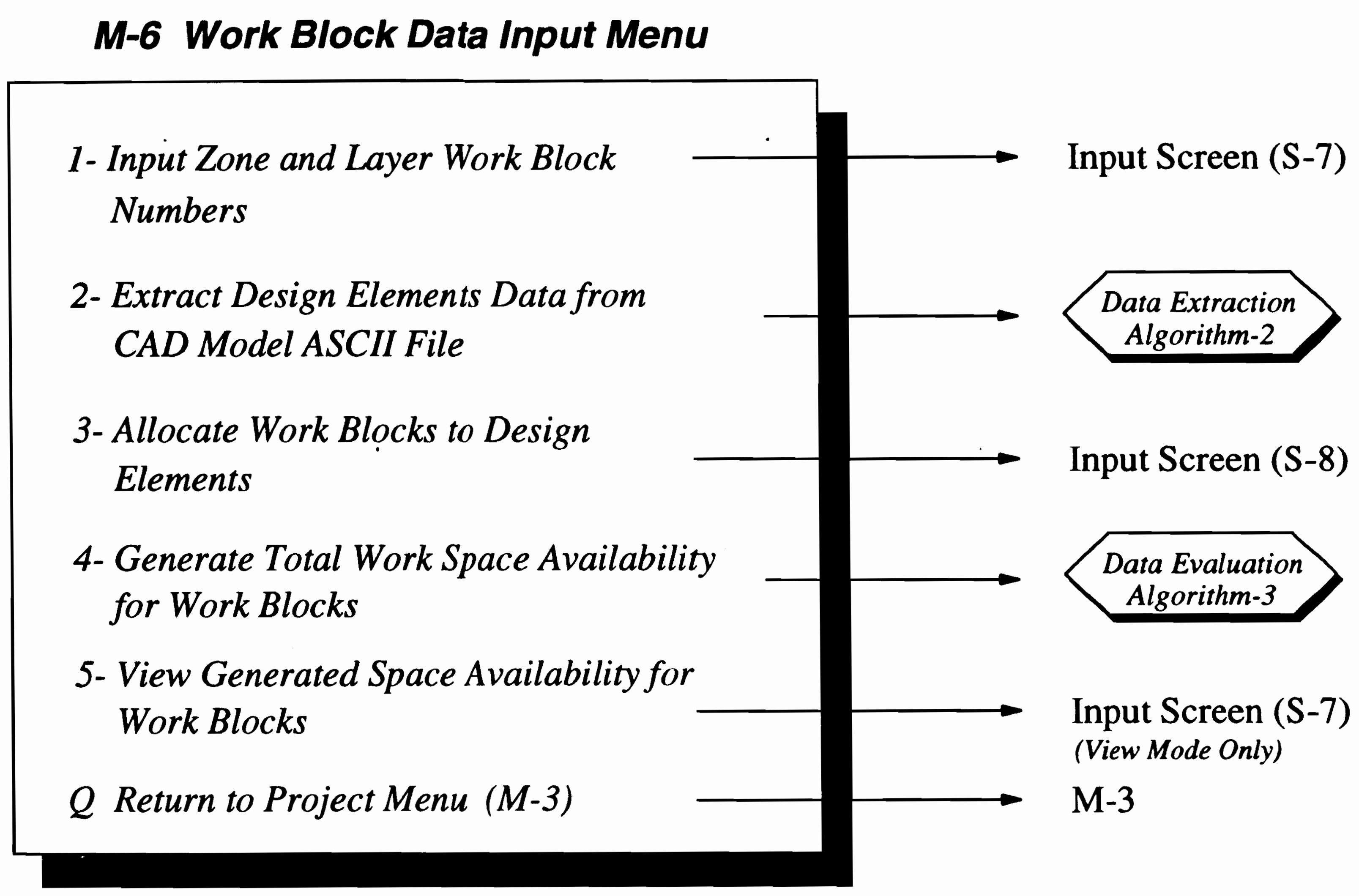

M-6 Work Block Data Input Menu
1- Input Zone and Layer Work Block Numbers
2- Extract Design Elements Data from CAD Model ASCII File
3- Allocate Work Blocks to Design Elements
4- Generate Total Work Space Availability for Work Blocks
5- View Generated Space Availability for Work Blocks
Q Return to Project Menu (M-3)
Input Screen (S-7)
Data Extraction Algorithm-2
Input Screen (S-8)
Data Evaluation Algorithm-3
Input Screen (S-7) (View Mode Only)
M-3

Figure 9.16 - The Work Block Data Input Menu (M-6)

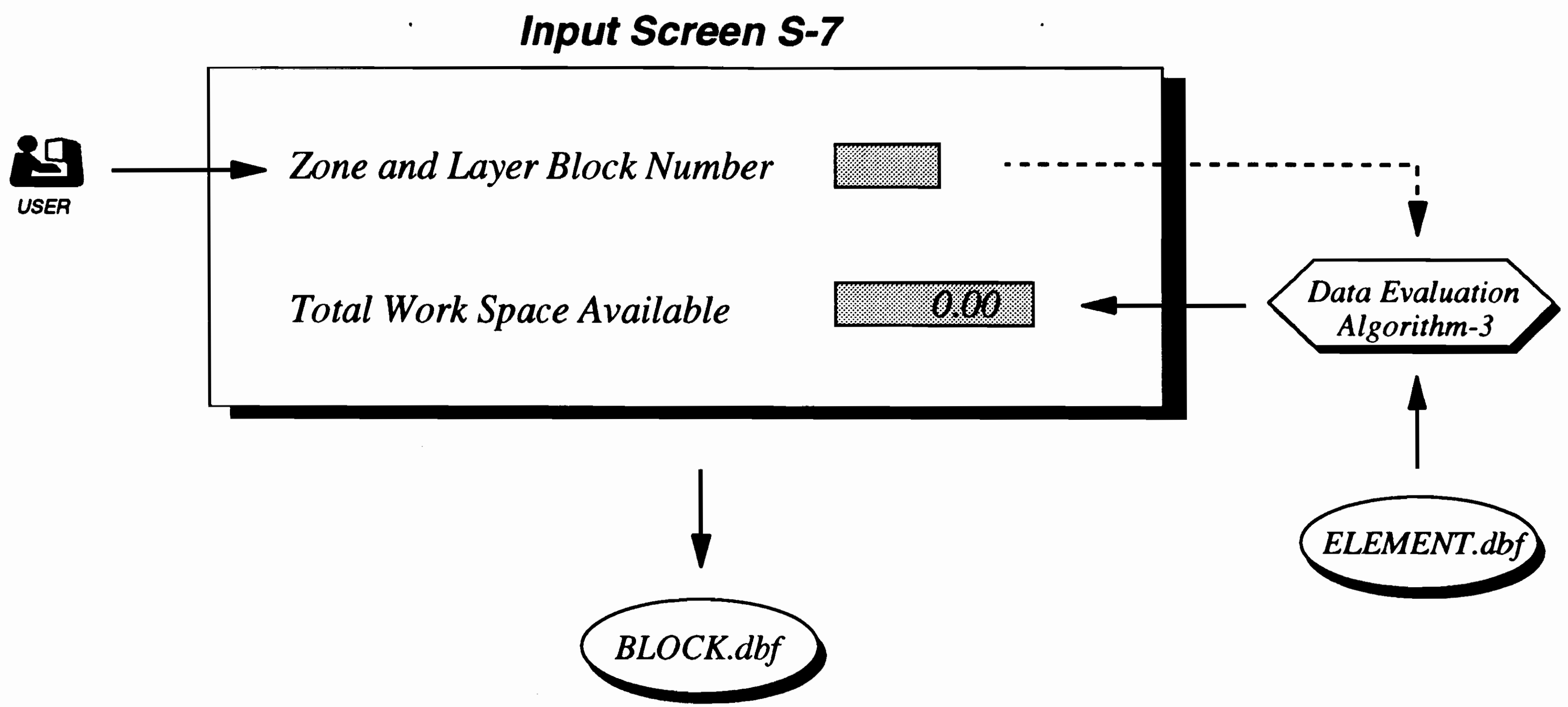

Input Screen S-7
USER
Zone and Layer Block Number
Total Work Space Available
0.00
Data Evaluation Algorithm-3
ELEMENT.dbf
BLOCK.dbf

Figure 9.17 - Input Screen for Zone and Layer Work Block Data (S-7)

2- Total Work Space Available

The *zone and layer block number* field contains an alphanumeric number for each zone and layer block. Work block numbers are input by the user for all defined zone and layer blocks of the typical floor. The *total work space available* field contains the total space availability value for the block. The value of the field is automatically computed for each block using evaluation algorithm-3. The algorithm utilizes the geometric data of the design elements in each work block to compute its total space availability value. This process is accomplished by selecting the forth option of the work block data input menu. At this stage, the user should only input block numbers of all work blocks defined for the typical floor.

Records of work block data will be stored in the **BLOCK.dbf** database file.

• Selecting option 2 *Extract Design Elements Data from CAD Model ASCII File* from the work block data menu activates a data extraction algorithm-2. The algorithm automatically extracts design elements geometric data stored in a CAD model ASCII file. Extracted data is stored in the ELEMENT.dbf database file. Data is stored in the appropriate fields of input screen S-8 shown in Figure 9.18. The screen contains eight fields:

1- Design Element Number

2- Design Element Type

3- Geometric Data (3 fields)

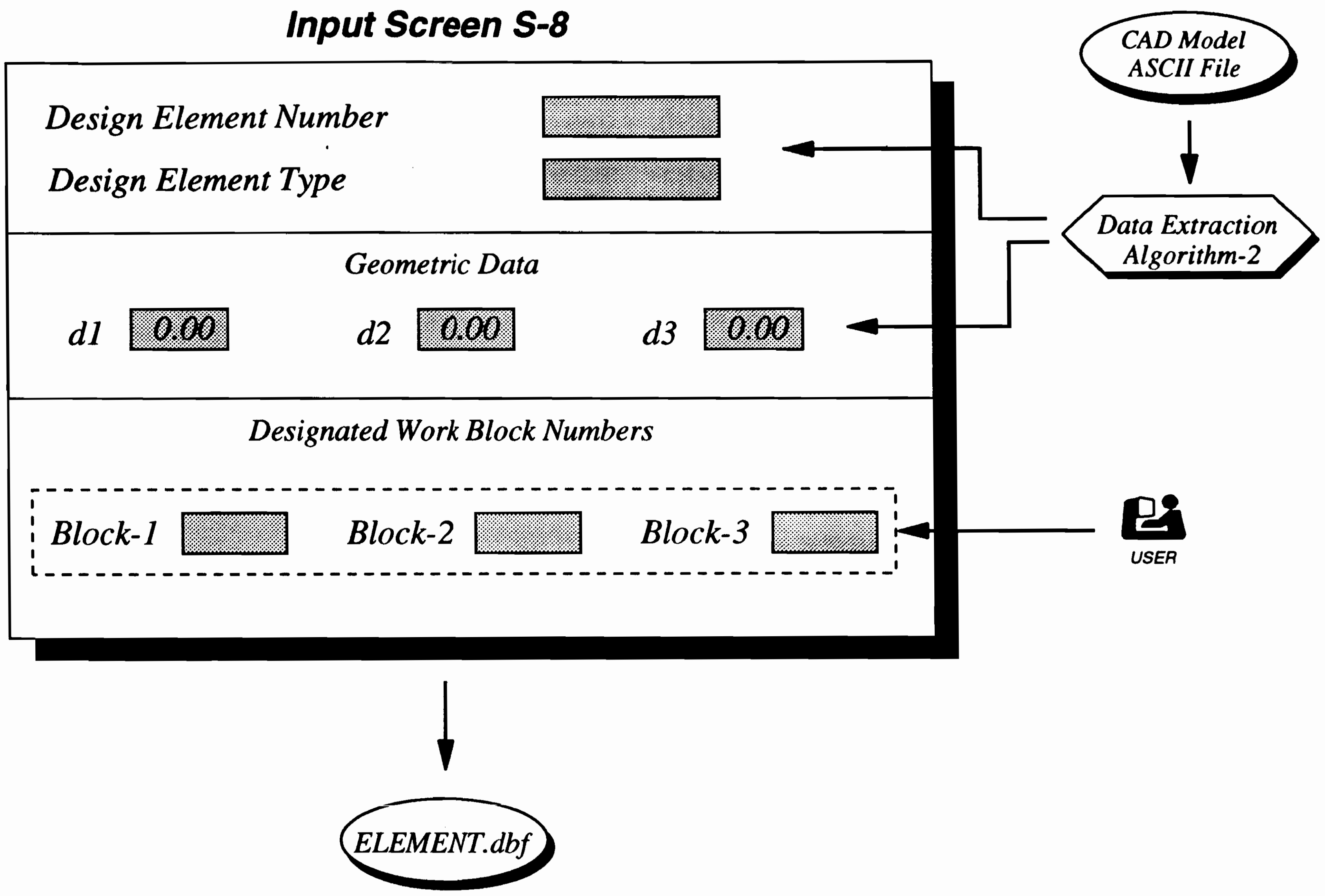


Figure 9.18 - Input Screen for Design Elements Data (S-8)

4- Designated Work Block Numbers (3 fields)

The *design element number* field defines an alphanumeric number to identify each element. The *design element type* field defines the type of the design element (e.g. slab, beam, column, etc.). The *geometric data d1, d2, and d3* fields define the three geometric parameters of each design element. For example, a column geometric data may be defined using three parameters: width, depth, and height (i.e. d1, d2, d3) or using two parameters: cross-section area, and height (i.e. d1, d2, and d3=0). The *designated work block numbers* fields define the number of the work block designated to the element. If a design element lies completely in one work block, only that block number will be defined. If a design element shares the boundary of two or more work block, then all block numbers are defined.

As illustrated in Figure 9.18, the design element number, type and geometric data are automatically extracted from the CAD model ASCII file utilizing the data extraction algorithm-2. Designated work block numbers are input by the user. This is accomplished when selecting option 3 of the work block menu.

• Selecting option 3 *Allocate Design Elements to Work Blocks* from the block menu activates input screen S-8 of Figure 9.18. This allows the user to manually define all work block numbers designated to the element. A work block is designated to a design element if part or all of the design element lies within the boundaries of the work block. As indicated earlier, if the element lies completely in one work block, the user will define the number of that block only. If the element shares the boundaries of two or more blocks, then all block numbers are defined. Input screen S-8 is set up to allow the user to define up to 3 blocks for each element.

Element records containing design elements data will be stored in the **ELEMENT.dbf** database file.

• Selecting option 4 *Generate Total Work Space Availability for Work Blocks* from the work block data input menu activates the data evaluation algorithm-3. For each work block record defined, the algorithm computes total space availability for the block. The algorithm utilizes the geometric data of design elements defined and generated using options 2 and 3 of this menu. The evaluation algorithm is discussed in detail in section 9.3.

• Selecting option 5 View *Generated Space Availability for Work Blocks* from the work block data input menu reactivates input screen S-7 of Figure 9.17 in view mode only. This option allows the user to only view the space availability value of each work block computed by the evaluation algorithm-3.

• Selecting option 6 *Return to Project Data Input Menu* from the project data input menu transfers control back to the project data input menu (M-3).

9.2.7 The Schedule Data Output Menu (M-7)

The objective of the *Schedule Data Output Menu* is to allow the user to view the results of the generated schedule. The menu is activated by selecting option 4 from the main menu (M-1). As illustrated in Figure 9.19, the schedule data output menu contains four options:

Option 1: Generate Output Data DBF File from ASCII File

Option 2: View Output Results (graphical format)

Option 3: View Output Results (screen format)

Option 4: View Output Results (table format)

Option 5: Return to The Main Menu (M-1)

• Selecting option 1 *Generate Output Data DBF File from ASCII File* from the output menu generates a DBF type file (OUTPUT.dbf) from the ASCII file (OUTPUT.art) generated by the knowledge-based system. This allows the user to view the results of the schedule in a database environment. This option should be selected by the user before attempting to view the scheduling results via options 2 and 3.

• Selecting option 2 *View Output Results (graphical format)* allows the user to view the results of the schedule using a graphical format. The graph is produced using C programming code developed using the Quick C 2.5 compiler.

• Selecting option 3 *View Output Results (screen format)* allows the user to view the

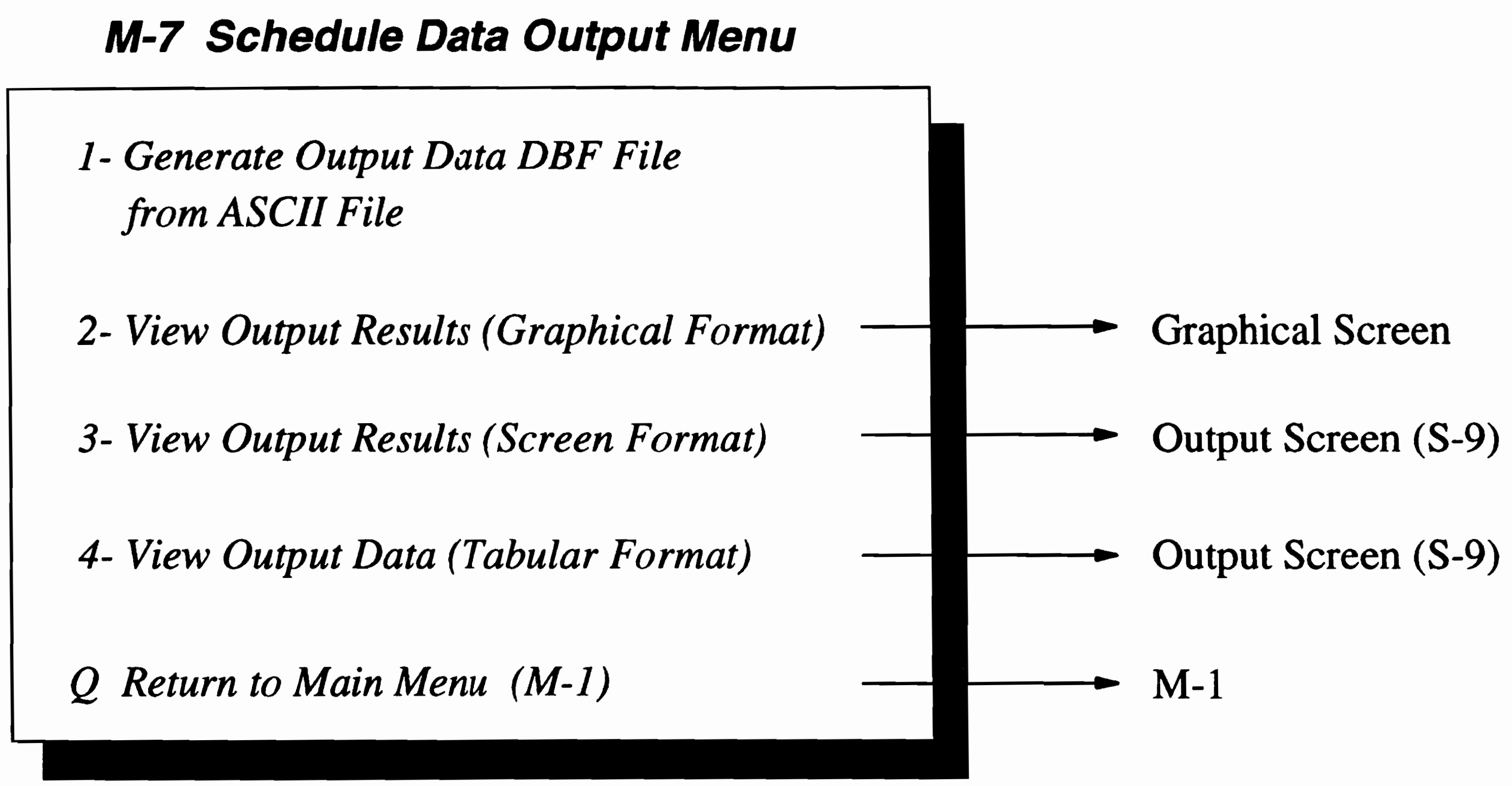
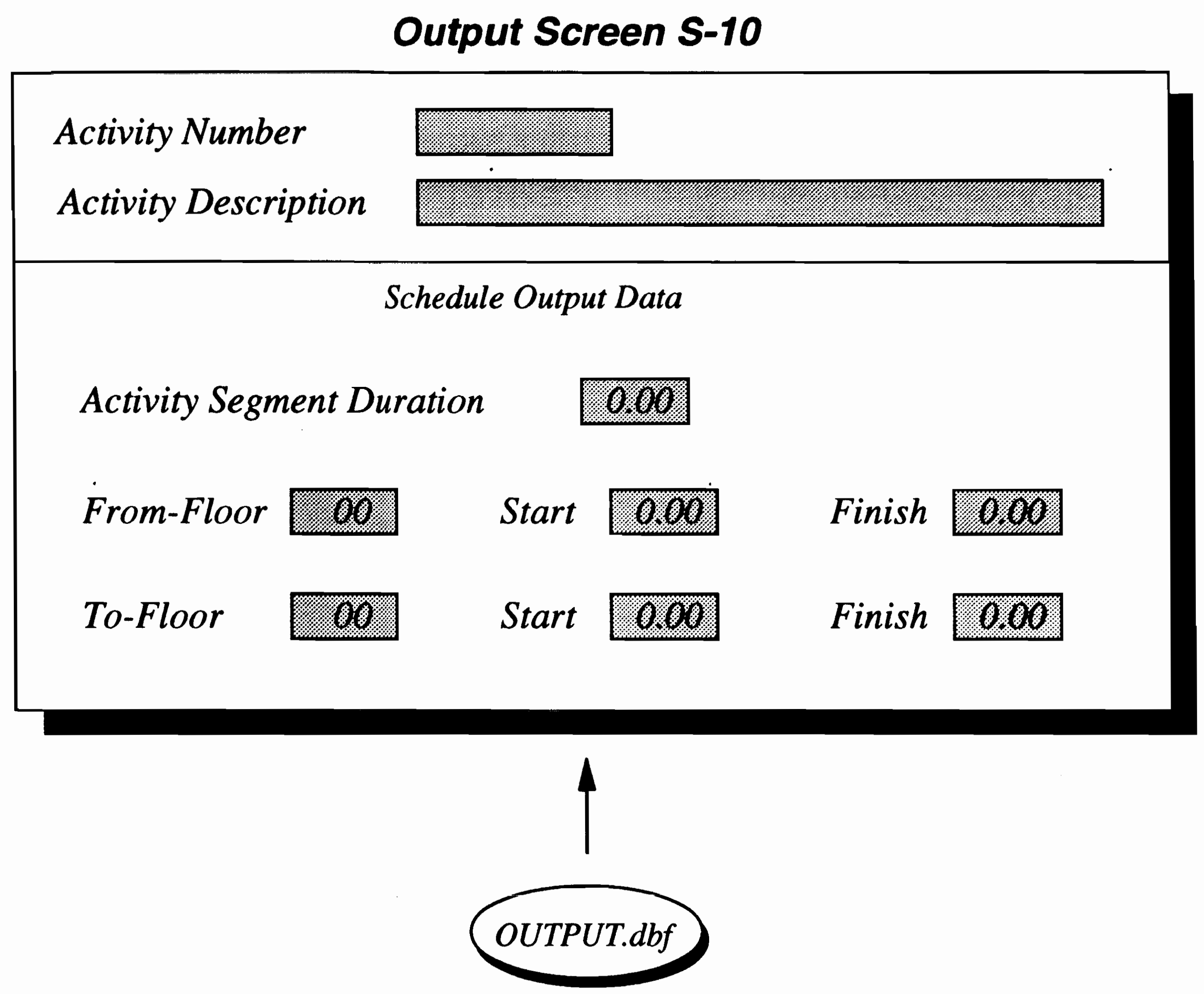

M-7 Schedule Data Output Menu
1- Generate Output Data DBF File from ASCII File
2- View Output Results (Graphical Format)
3- View Output Results (Screen Format)
4- View Output Data (Tabular Format)
Q Return to Main Menu (M-1)
Graphical Screen
Output Screen (S-9)
Output Screen (S-9)
M-1

Figure 9.19 - The Schedule Data Output Menu (M-7)



Output Screen S-10

Activity Number					
Activity Description					

Schedule Output Data

Activity Segment Duration	0.00				
From-Floor	00	Start	0.00	Finish	0.00
To-Floor	00	Start	0.00	Finish	0.00

Figure 9.20 - Output Screen for Schedule Results Data (S-10)

results of the schedule one activity at a time using a screen format. Selecting this option activates output screen S-10 illustrated in Figure 9.20. Output screen S-10 contains 9 fields:

1- Activity Number

2- Activity Description

3- Activity Segment Duration

4- From-Floor

5- To-Floor

6- From-Floor Start Date

7- From-Floor Finish Date

8- To-Floor Start Date

9- To-Floor Finish Date

The *activity segment duration* field defines the duration for each activity segment. If the activity is split along the floors, the duration of scheduled activity segments may vary. The *from-floor/to-floor number* fields defines the floor numbers of each scheduled activity section. As indicated in section 9.1, if the activity is scheduled continuously along the floors the floor numbers in these two fields will correspond to the first and last typical floors of the whole building project. If the activity is scheduled in segments by allowing splitting, the floor numbers of these fields will correspond to the start and end floors for each activity segment. In this case, each activity segment will be represented in one screen (i.e. record). The *start and finish dates* fields define the start and finish times of the start/end floors for each activity segment.

• Selecting option 4 *View Output Results (tabular format)* from the output menu allows the user to view the scheduling data in a tabular format. Schedule data for all activities is organized in a tabular form and sorted by activity and floor number. This tabular format allows the user to view scheduled data of all activities simultaneously to compare and verify results.

• Selecting the last option of the output menu transfers control back to the main menu (M-1).

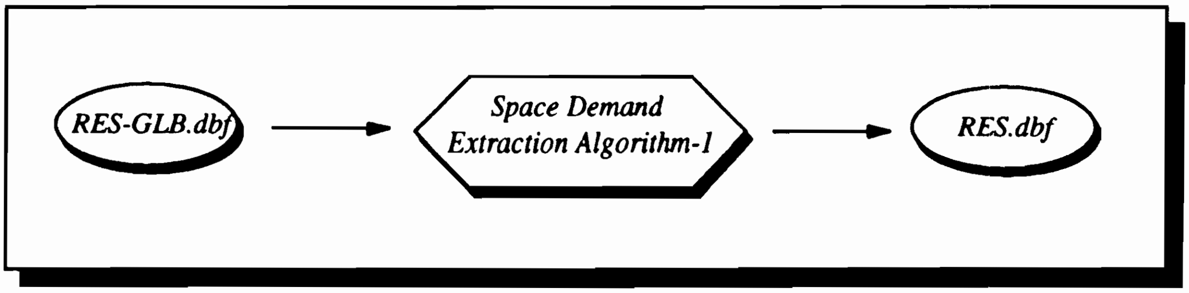
9.3 DataBase Algorithms

Several algorithms are developed within the database environment. The objective of these algorithms is to generate and manipulate data within the database files and to import other data from external files into the database. As depicted in Figure 9.21, there are 3 algorithms:

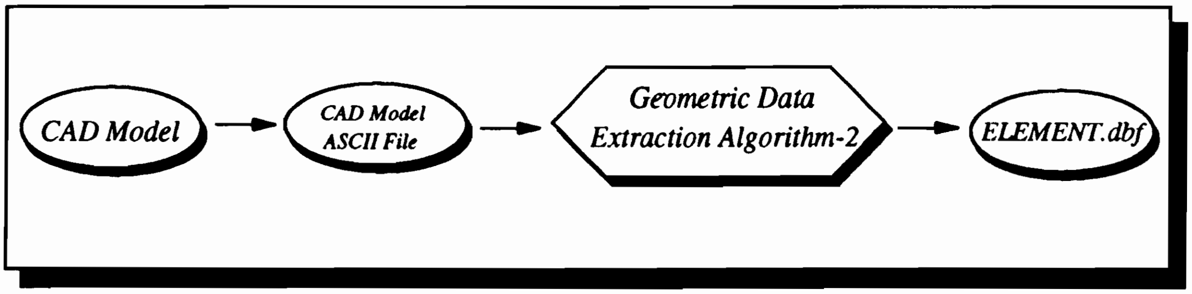
1- Space Demand Extraction Algorithm-1

2- Geometric Data Extraction Algorithm-2

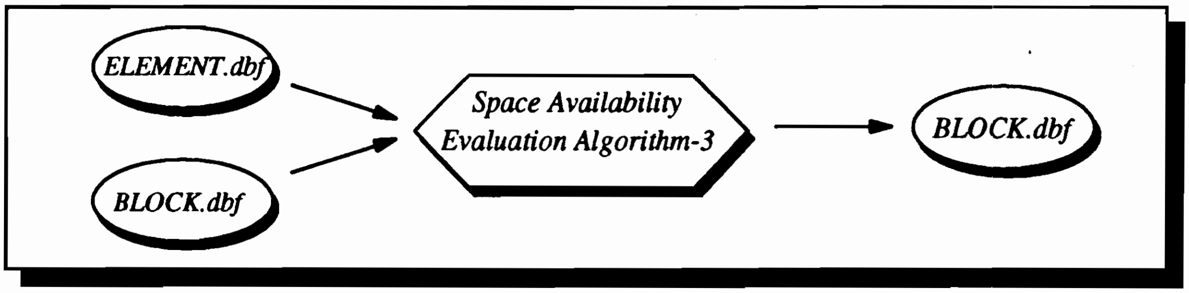
3- Space Availability Evaluation Algorithm-3



Space Demand Extraction Algorithm-1



Geometric Data Extraction Algorithm-2



Space Availability Evaluation Algorithm-3

Figure 9.21 - DataBase Algorithms

9.3.1 The Space Demand Extraction Algorithm-1

The purpose of this algorithm is to extract global work space demand values for all resources of the project from the RES-GLB.dbf file and store it in the RES-.dbf file. For each resource defined in the RES.dbf database file, the algorithm extracts its corresponding global space demand values from the RES-GLB.dbf file. This process is illustrated in Figure 9.22.

The extraction algorithm-1 is activated by selecting option 2 from the resource data input menu (M-5).

9.3.2 The Geometric Data Extraction Algorithm-2

The purpose of this algorithm is to extract the geometric data of design elements from a CAD model ASCII file and store it into the ELEMENT.dbf database file. The extracted geometric data is utilized to compute the space availability of each work block. This computation is accomplished by the third algorithm described in section 9.3.3 below.

Since the output format of geometric data stored in ASCII files varies for different CAD systems, the format shown in Figure 9.23 is assumed for the purpose of the research. Geometric data must be stored in the format shown in Figure 9.23 for the algorithm to be able to read the data. The algorithm can later be modified for any specific format corresponding to a specific CAD systems.

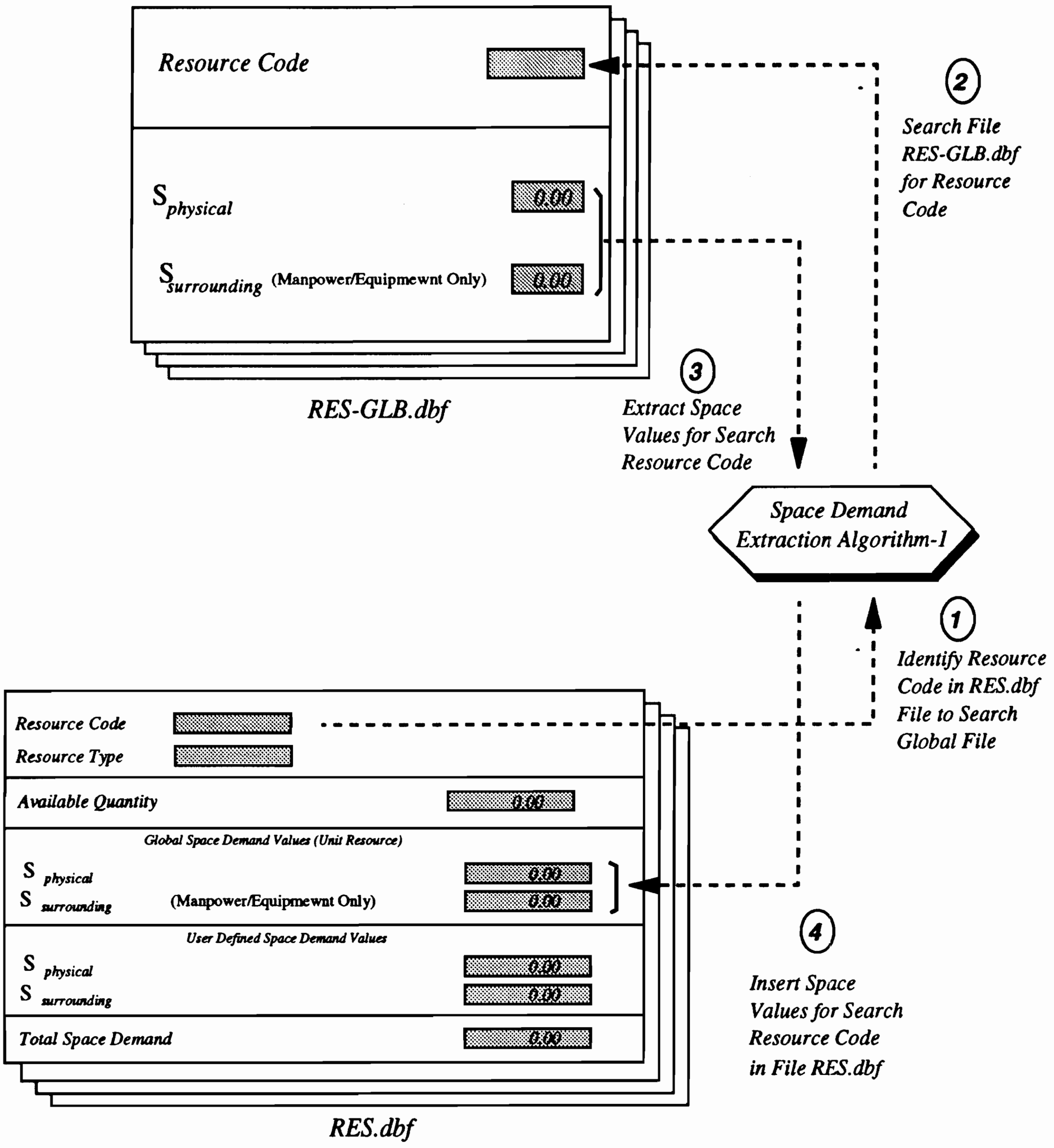
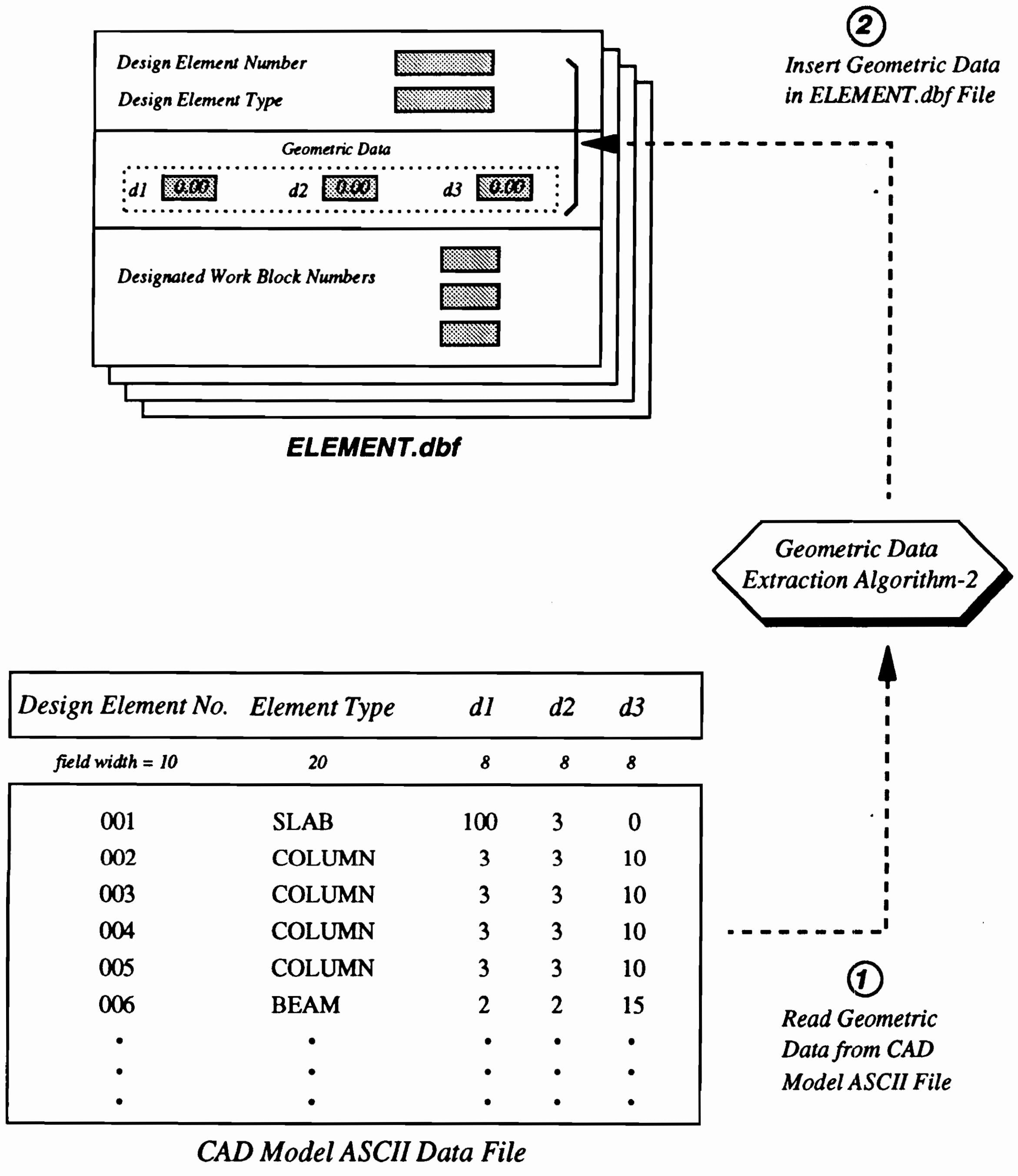

Resource Code
$S_{physical}$
0.00
$S_{surrounding}$ (Manpower/Equipmewnt Only)
0.00
RES-GLB.dbf
2
Search File RES-GLB.dbf for Resource Code
3
Extract Space Values for Search Resource Code
Space Demand Extraction Algorithm-1
1
Identify Resource Code in RES.dbf File to Search Global File
Resource Code
Resource Type
Available Quantity
0.00
Global Space Demand Values (Unit Resource)
$S_{physical}$
0.00
$S_{surrounding}$ (Manpower/Equipmewnt Only)
0.00
User Defined Space Demand Values
$S_{physical}$
0.00
$S_{surrounding}$
0.00
Total Space Demand
0.00
4
Insert Space Values for Search Resource Code in File RES.dbf
RES.dbf

Figure 9.22 - The Space Demand Extraction Algorithm-1



Design Element Number
Design Element Type
Geometric Data
d1 0.00
d2 0.00
d3 0.00
Designated Work Block Numbers
ELEMENT.dbf
②
Insert Geometric Data in ELEMENT.dbf File
Geometric Data Extraction Algorithm-2
①
Read Geometric Data from CAD Model ASCII File
CAD Model ASCII Data File

Design Element No.	Element Type	d1	d2	d3
field width = 10	*20*	*8*	*8*	*8*
001	SLAB	100	3	0
002	COLUMN	3	3	10
003	COLUMN	3	3	10
004	COLUMN	3	3	10
005	COLUMN	3	3	10
006	BEAM	2	2	15
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•

Figure 9.23 - The Geometric Data Extraction Algorithm-2

9.3.3 The Space Availability Evaluation Algorithm-3

The purpose of this algorithm is to compute the total work space availability for each work block. The algorithm computes the space of a work block based on the geometric data of all design elements allocated to that block. As illustrated in Figure 9.24, this process is executed as follows:

1- *Identify all design elements allocated to the work block.* Each design element is allocated to one or more block. The algorithm will identify all design elements allocated to each work block.

2- *Determine Gross space of work block based on geometric data of element 'slab'.* Excluding all other elements, the algorithm computes the space of the block based on the geometric data of the slab. Using the slab area and the height of the floor, the algorithm computes a gross space value for the block. In order to utilize the slab area, it is required that the floor slab in the CAD model is broken into several segments during its drawing to define the smallest block.

3- *Determine the net available space for the block.* To determine the net space available of the work block, space consumed by other design elements is deducted from the gross value computed in step 2.

9.4 Database/Knowledge-Base Interface

The data in the overall scheduling system flows in two directions. Input data defined in the database system flows from the database to the knowledge-based scheduling system. Once the schedule generation process is completed, output data of the generated schedule flows in the opposite direction back to the database system for user review. This two directional flow of data is illustrated in Figure 9.25.

To allow the flow of data from the database to the knowledge-based system, some database input files (*.dbf) are copied to corresponding ASCII files. As illustrated in Figure 9.25, seven database files are copied. These files contain the required data for the generation of the schedule. Generating the ASCII files from the database DBF files is accomplished by selecting option 5 from the project data input menu (M-3).

Similarly, to allow the flow of data from the knowledge-based scheduling system back to the database system, the output ASCII file (OUTPUT.art) generated by the scheduling system is transferred to a database DBF type file (OUTPUT.dbf). This allows the user to view the results of the schedule within the database environment. Generating the database file is accomplished by selecting option 1 of the schedule data output menu (M-7).

9.6 Chapter Summary

This chapter describes the structure and components of the central database system. The database represents the first component of the SCaRC scheduling system. All data needed by the 9-25

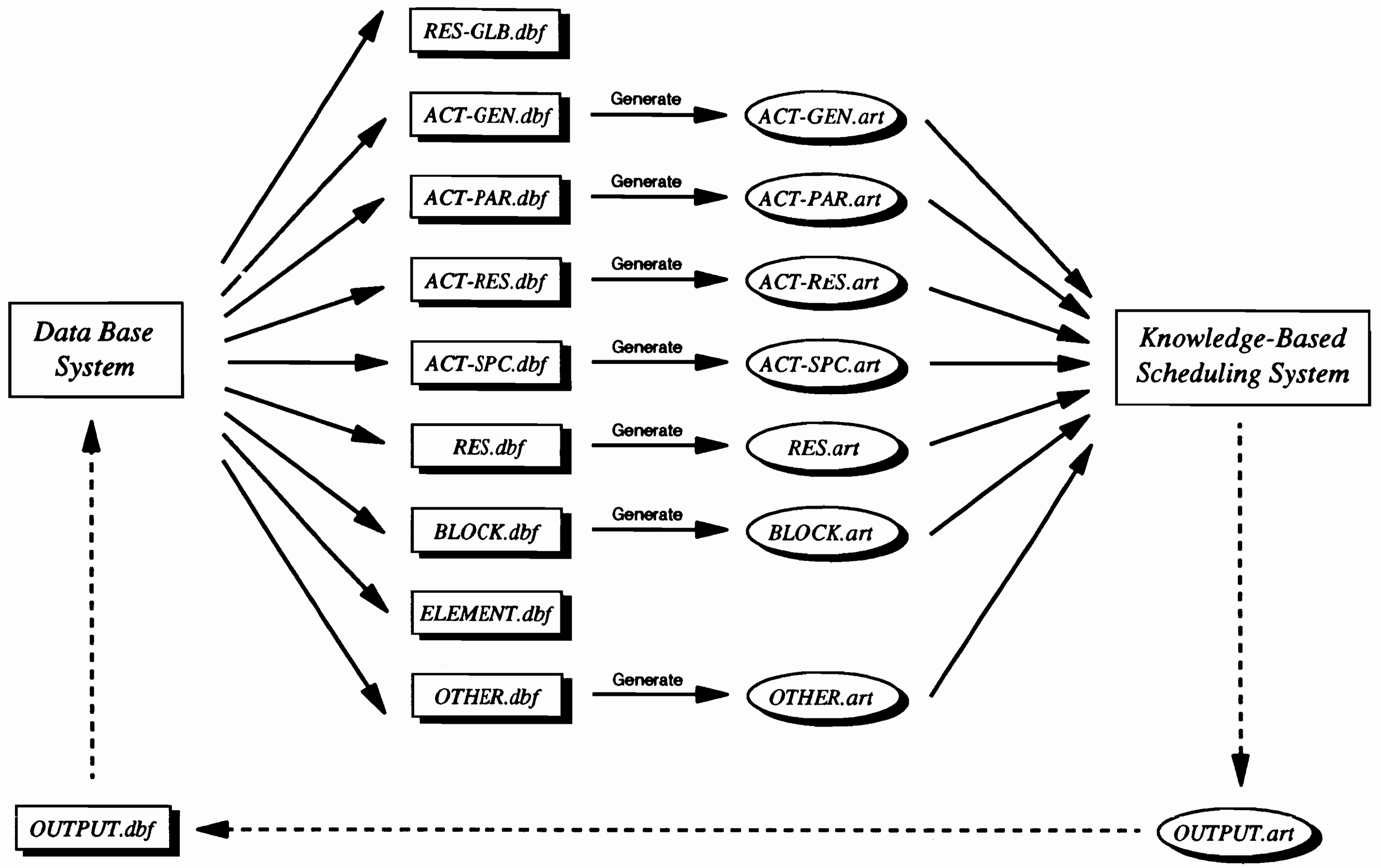


Figure 9.25 - Generate ASCII Files from DBF Database Input Files

knowledge-based scheduling system (second component) to generate a schedule is defined in the database system.

The chapter first discusses the different categories of data classification defined in the database system. There are six categories of data defined: global resource demand system data, activity input data, resource input data, work block input data, other project input data, and schedule results output data.

The chapter also presents the main components of the database system; the file structure and the user interactive menu system. There are ten database files defined for storage of data. Files are accessed by the user from a structured menu system via a set of input/output screens. The screens allow the user to define and view the data stored.

In addition to the file structure and menu system, the database comprises of several algorithms defined within the database environment. The purpose of these algorithms is to generate and manipulate data within the database files. There are three database algorithms developed; a space demand extraction algorithm-1, a geometric data extraction algorithm-2, and a space availability evaluation algorithm-3.

Finally, the chapter discusses the flow of data between the database system and the knowledge-based scheduling system. Data exchange between the two components is established using ASCII type interface files.

The database file structure, input/output screens, menu system, and algorithms are developed using the dBASE IV database programming language. The graphical output format viewed from within the database environment is produced by a C program developed using Microsoft Quick C Compiler (ver 2.5). The program is linked to the database program to allow the user to view the graph while in the database environment.

The source code listing of the C program is provided in Appendix E. The source code listing of the database program is provided in Appendix F.

10.0 SUMMARY and CONCLUSION

The research presented in this dissertation investigates work space requirements as a new constraint in the scheduling process. The research studies the conflict between work space demand of any activity and available space of the work area and examines and analyses the impact of such conflict on the generation of schedules.

A model is developed for scheduling repetitive work in multi-story projects. The model allows to quantify space demand and availability values for each activity and proposes space-based scheduling decisions to sequence the activities based on a comparison of these values.

The model also allows to take into account resource constraints to generate the schedules.

The model recognizes the conflict between resources that can be made available to the project and those needed to execute each activity. Resource-based scheduling decisions are integrated with the space-based decisions to generate the schedules.

Horizontal and vertical logic constraints associated with repetitive work in multi-story projects are incorporated in the scheduling process of the model. Work continuity issues and varying productivity rates are used as main scheduling decision options. The model adopts a procedure to schedule non-continuous activities using variable length segments along the typical floors. The procedure is based on a "maximum number of splits" parameter defined by the user. In addition, the model allows for adjusting the initial defined resource demand pools for different activities to account for any modifications that may occur to the activity durations during scheduling. Loss of productivity as a result of the combined effect of travel time and learning curve phenomena is also incorporated in the generation of the schedule by the model.

The model has been taken to a prototype proof of concept by developing the **SCaRC** (Space Constrained and Resource Constrained) scheduling system. The system demonstrates the use of construction scheduling knowledge about limited resources and work space availability along with other constraints to generate a construction schedule for the repetitive floors. The system takes as input a logical network plan for a typical floor. The system produces as output a space-constrained and resource-constrained schedule for all the typical floors of the facility. The system is implemented using a knowledge-based approach.

This chapter provides a summary and conclusion for the dissertation. The chapter first summarizes the basic structure of the SCaRC scheduling system. The chapter then outlines the contribution of the research to the body of knowledge. Recommendations and future extensions to meet the overall objectives of the SCaRC scheduling system are listed. Finally, the chapter provides a conclusion of the work.

10.1 Summary of the SCaRC Scheduling System

Figure 10.1 depicts a diagram of the overall SCaRC scheduling system as it is currently implemented. The system consists of two main components; a database component, and a knowledge-based scheduling component.

The database component acts as a user interface and provides for defining all input data necessary for the production of the schedule. Data is defined either by direct input by the user or by automatic extraction from external ASCII type files. The database also allows the user to view the results of the schedule generated by the knowledge-based scheduling system (second component). Several output formats are provided including a graphical format. The interaction between the user and the overall system is accomplished through the database system.

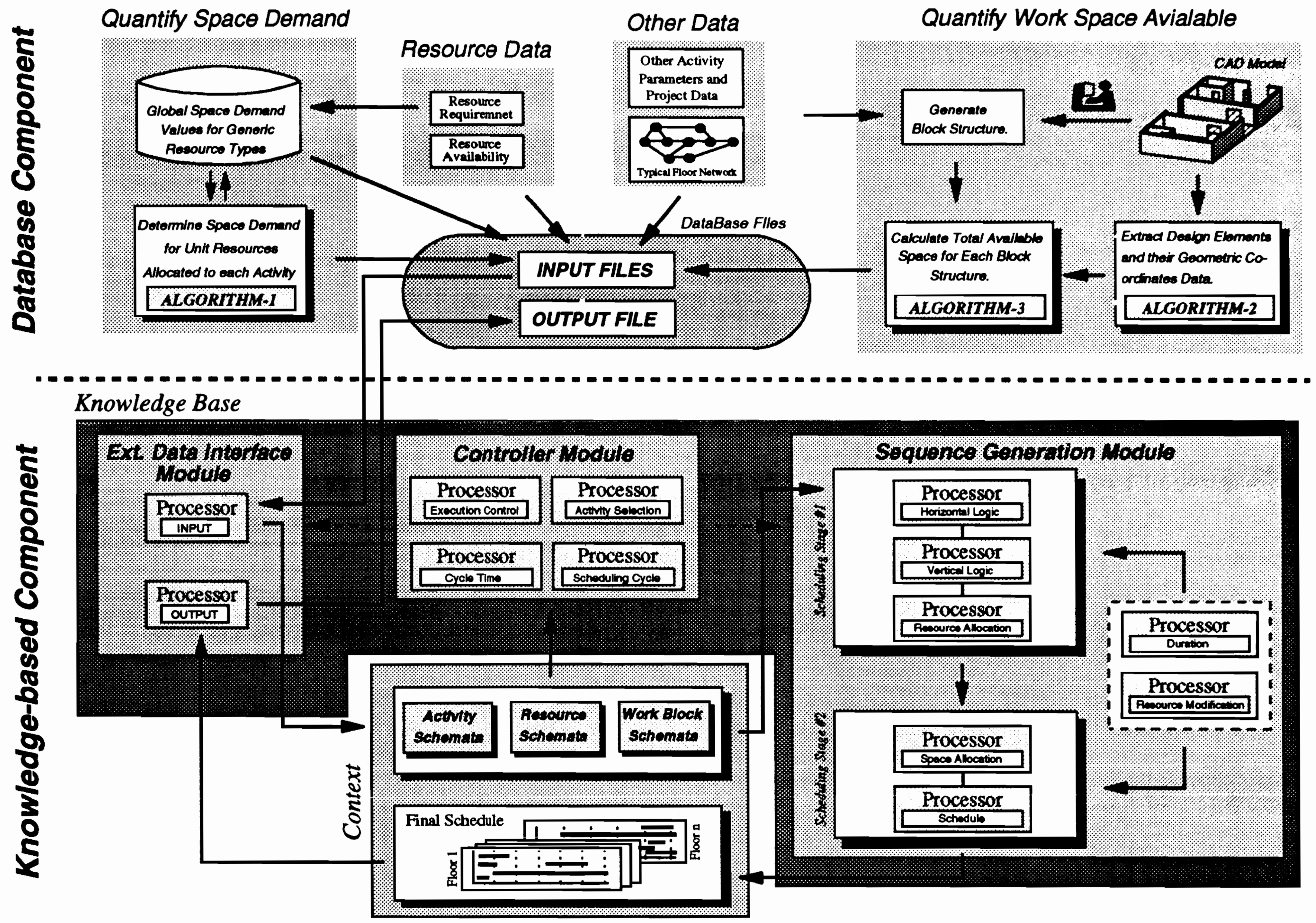

Database Component
Quantify Space Demand
Global Space Demand Values for Generic Resource Types
Determine Space Demand for Unit Resources Allocated to each Activity
ALGORITHM-1
Resource Data
Resource Requiremnet
Resource Availability
Other Data
Other Activity Parameters and Project Data
Typical Floor Network
DataBase Files
INPUT FILES
OUTPUT FILE
Quantify Work Space Avialable
CAD Model
Generate Block Structure.
Calculate Total Available Space for Each Block Structure.
ALGORITHM-3
Extract Design Elements and their Geometric Co-ordinates Data.
ALGORITHM-2
Knowledge Base
Knowledge-based Component
Ext. Data Interface Module
Processor INPUT
Processor OUTPUT
Controller Module
Processor Execution Control
Processor Activity Selection
Processor Cycle Time
Processor Scheduling Cycle
Sequence Generation Module
Scheduling Stage #1
Processor Horizontal Logic
Processor Vertical Logic
Processor Resource Allocation
Scheduling Stage #2
Processor Space Allocation
Processor Schedule
Processor Duration
Processor Resource Modification
Context
Activity Schemata
Resource Schemata
Work Block Schemata
Final Schedule
Floor 1
Floor n

Figure 10.1 - Overall SCaRC Scheduling System

The second component of the SCaRC system is the knowledge-based scheduling system. This is the main component of the system and is responsible for the actual production of the schedule. There is no interaction between the user and this component. As illustrated in Figure 10.1, the knowledge-based scheduling system comprises of three modules. These modules are:

1- An External Data Interface Module.

2- A Controller Module.

3- A Sequence Generation Module.

The ***External Data Interface Module*** is responsible for extracting scheduling data stored in the system's database. The module is also responsible for transferring the outcome of the scheduling process (i.e. the generated schedule) to an output ASCII file. Data stored in this file is extracted by the user from the data base system to review the results. These functions are accomplished by an input and output processors that constitute this module.

The ***Controller Module*** controls the overall scheduling process of the system and consists of many processors. These processors are responsible for determining activities that are eligible for scheduling at any time period and the continuous updating of the scheduling time. The processors are also responsible for the heuristic prioritization process of the activities to rank those competing for resources and/or space. Another main function of the controller module is to manipulate or control the sequence of the scheduling process. The module is responsible for sending the appropriate instructions to the sequence generation module (third module) to carry out the appropriate scheduling step and verify the specific scheduling constraint. The scheduling process is continuously monitored by the controller module. Once the overall schedule is generated, the controller module transfers control to the external data interface module to print the generated schedule to the output file.

The ***Sequence Generation Module*** is the center piece of the knowledge-based scheduling system. The module is responsible for the actual sequencing of each activity along the typical floors in order to establish the schedule. The module generates the sequence based on different constraints including limited resource and work space availabilities. As illustrated in Figure 10.1, this module comprises of many processors that generate the schedule in two consecutive stages.

Processors of the first stage are responsible for checking and verifying horizontal and vertical logic constraints among the different activities along with satisfying continuity issues. Activity duration value is selected based on the user defined normal value and activity resource demand pools are modified, if necessary, to correspond to the selected duration. The processors are also responsible for verifying resource availability to insure that enough resources are available to perform the activity for its entire duration along the typical floors.

In the second scheduling stage, a Space Allocation Processor is responsible for verifying the availability of space for each activity and carrying out the different scheduling decisions necessary to schedule or delay the activity based on space availability considerations.

Constraints verification in both stages are continuously repeated until each activity satisfies all applicable constraints and its final start and finish dates are determined. The process continues until all activities are scheduled. At this point, the control is shifted by the controller module back to the external data interface module to output the scheduling results to the system's database.

10.1.1 Computer Software

The SCaRC scheduling system utilizes the following software systems:

1- dBASE IV database programming software is used to develop the database component of the system.

2- Quick C 2.5 compiler is used to develop the graphical output linked to the database system to allow the user to view the results in a graphical format.

3- ART-IM (Automated Reasoning Tool for Information Management) expert system development environment by Inference Corporation is used to develop the knowledge-based component of the model responsible for the actual production of the schedule.

10.1.2 Computer Hardware

The SCaRC scheduling system is developed using an IBM-PC (or compatible). At least 4 megabytes of memory (RAM) is required to load and operate the system. The PC must have a floppy drive (a:) to manipulate input/output files exchanged between both components of the system during the generation of the schedule. The main program files must be stored on a hard disk drive.

10.2 Contribution of the Research

The research presented in this dissertation takes the construction scheduling process a step further by acknowledging the requirements of activities for work area or space necessary for material storage and movement of manpower and equipment. The developed model recognizes work space as a new constraint in the production of the schedule.

The research characterizes work space in terms of two parameters; work space demand and work space availability. Space demand for an activity defines space required to accommodate the activity in a work area. Space availability for the activity defines available space of work area at a particular location during a particular time period.

The developed scheduling model comprises a method to quantify these parameters. Space demand for an activity is quantified based on space required to accommodate each resource allocated to the activity. A mathematical formulation is proposed to estimate required space for any resource. To quantify space availability, a zone/layer work block structure for any work area is proposed. A work block defines a particular location in the work area (zone) during a particular time period of construction (layer). Space availability for any activity is evaluated based on the total available space of work block(s) to which the activity is allocated.

The model also comprises a space-based scheduling procedure to implement these space parameters for sequencing of activities. The proposed procedure compares space demand with availability for any activity and considers several scheduling decisions and actions to account for lack of work space. The space-based scheduling procedure is integrated with other scheduling procedures in order to generate the overall schedule. The result is a more feasible and realistic schedule.

In addition to recognizing and incorporating work space as a constraint in the scheduling process, the developed model allows for the following:

- The scheduling model recognizes the combined effect of travel time and learning curve phenomena associated with repetitive work in multi-story buildings on the overall crew productivity. A step function is implemented in the scheduling procedure to account for loss in productivity rates.

- The model provides for a step to modify the initial resource demand pool defined by the user for each activity. During the scheduling process, activity duration may be increased above its normal value to satisfy specific constraints. The model allows to adjust the resource demand pool to account for the new duration determined.

- The model allows for scheduling non-continuous activities in segments with variable number of floors. Based on a maximum number of splits parameter defined by the user, a procedure is adopted to schedule each segment independently. Segments that are continuous are grouped in one segment.

10.3 Recommendations and Future Extensions

During the development of SCaRC, several issues became obvious that need further enhancement and investigation. Implementation of these issues in the current state of SCaRC system will make it a more comprehensive project scheduling tool.

This section describes various issues that can enhance the capabilities of the system. These issues are grouped under two categories,

1- Issues that require additional investigation and research, and

2- Issues that require additional programming either of the database system and/or the knowledge-based scheduling system.

10.3.1 Issues Requiring Additional Investigation and Research

- Validation of system. The developed SCaRC system is a first attempt to recognize work space as a new constraints in the scheduling process. The system needs to be validated by testing it to real case scenarios in order to verify the developed space-based scheduling concepts and rules.

- Enhance the space-based scheduling rules that utilizes the space capacity factor concept to take into consideration the decline of productivity for activities that are already scheduled in any work block. The space-based scheduling rules of the developed system only modify the productivity of the activity being tested and does not modify the productivity of other activities already scheduled and sharing the same work block with the current activity.

- Develop actual Productivity-SCF relationship curves in order to model the actual behavior of construction crews in congested work areas. The curves can be developed for single activities or for groups of activities that may react similarly to limited work space conditions. The developing of such curves will require site observations and data gathering from construction operations. The SCaRC system currently uses a single theoretical step function to model the increase in duration as a result of limited work space for all activities considered for scheduling.

- Develop actual functions to model the variation of crew productivity as a result of the combined effect of travel time and learning curve phenomena on the duration of various construction activities. Again, this will require site observation and data gathering. Currently, the SCaRC system uses a simple step function to model the loss in crew productivity under such conditions.

- Develop actual functions to model the availability of work space as stored material in the floor is turned into work in place. Such functions are required for activities with a class 'C' space demand.

- Acquire global space demand data for unit construction resources. These data will be stored in the global system file of the database system. The data will assist the user to make a more realistic estimation of space demand values for resources taking into account the specific conditions of the project under consideration (e.g. physical characteristics of the work area in the floor).

- Separate the priority ranking criteria of resource scheduling from space scheduling. The current scheduling procedure adopted in SCaRC prioritizes the activities using heuristic ranking criteria tested for resource constraints scheduling procedures. Each ranked activity is scheduled for resource and space constraints before the next activity in order is considered. A future version of SCaRC may schedule all activities first based on resource constraints using the current ranking criteria. Then the entire resource-constrained schedule should be checked for space constraints using a different set of ranking criteria to be developed specifically for space constraints.

- Develop an interactive graphical interface for the system to allow the user to allow the user to manipulate input data using the graphical output. This feature will allow the user to conduct "what-if" scenarios on the interactive graphical output by clicking and changing the schedule bar of any activity or changing specific data from popup menus.

10.3.2 Issues Requiring Additional Programming

- Automate the process of defining the zone/layer block structure for the typical floor to occur during the development of the CAD model. Zones and layers can be defined as attributes during the creation of the CAD model of the floor and the information can be extracted together with the geometric data from the CAD database file.

- Modify the data extraction interface of the database to allow the system to extract geometric data from a variety of CAD systems (e.g. AutoCAD, CifeCAD, etc.). The current status of the database only allows to extract geometric data from an ASCII file with a specific defined format. Data from CAD systems must be arranged in such format before it can be extracted.

- Improve the geometric computing capabilities of the space availability evaluation algorithm-3 to quantify work space of any block structure. The current algorithm adopts a simple procedure that requires the slab to be drawn separately for each zone and layer. The modified algorithm should be able to compute the space of the block from the geometric data of the end elements regardless to the slab.

- Allow for the storage of material of several floors on one floor. The current state of SCaRC does not allow the user to specify material storage at specific floors. Rather, the program considers that material required for each floor is stored in that floor.

- Allow to store material in each floor in various work areas by allowing the user to define more than one block for resource material. During the scheduling process, the scheduling procedure should verify the different blocks and make decisions about choosing the best work block to store the material to avoid work congestion.

- Enhance the scheduling procedure of SCaRC to be able to schedule each activity inconsistently along the typical floors rather than to be confined to a specific direction. The SCaRC system currently schedules each activity in an upward direction or a downward direction only based on the attribute of the "workflow direction" parameter defined for the activity.

- Provide a smart algorithm that can verify the savings in time that may be gained from allowing some activities to be scheduled during limited work space conditions with a reduced productivity rate. The algorithm should test whether this scheduling option will provide savings in time over the option of delaying the scheduling of the activity to a later time period to avoid such conditions.

- Modify the resource availability database file to provide for defining variable availability values for each resource during different duration periods. The current state of SCaRC allows the user to define one availability value for each resource that remains constant for the entire duration of the project.

- Modify the activity resource demand database file to allow to define different resource demand quantity values for different ranges of activity duration. This will be necessary while adjusting the resource pool of the activity when the activity normal duration is increased during scheduling to satisfy a specific constraint. The current state of SCaRC allows the user to define one demand quantity per resource. The system then uses a simple algorithm to compute the decrease in the resource quantity as the activity duration is increased.

- Provide a module to schedule activities of the non-repetitive portion of the multi-story building. This will provide for generating a complete schedule for the overall project.

- Allow the user to modify key scheduling parameters initially defined during data input from within the knowledge-based component of SCaRC in order to test different case scenarios. The user interaction with the current SCaRC system that allows him/her to make any variation to the input data is limited to the database system. This is very time consuming as the user must leave the ART-IM environment and return to the database environment to perform such data changes.

10.4 Current Related Research Efforts

Two research efforts directly related to the work presented in this dissertation are currently in progress. The first research work is in progress at the University of Michigan [Tommelein et al, 92] and aims at investigating space scheduling for material handling and storage on construction sites. The second research effort has been initiated at Virginia Tech to study the learning and unlearning curve effect on construction productivity for activities characterized with repetitive work. A short description of these research efforts is as follows:

Research at The University of Michigan

The current research at the University of Michigan [Tommelein et al, 92] studies space scheduling of material during handling and storage on construction sites. Different factors that control the space scheduling procedure are studied. Such factors include: space need (demand), timing of resources (during arrival handling and removal, or storage), and location of the work area.

A prototype knowledge-based scheduling system [MovePlan] is under development to assist field staff with the space scheduling of material on site. Given activity procedures and durations along with activity space needs, the program provides the user with a schedule showing material movement over time. The system comprises an interactive graphical interface that provides the user with two-dimensional schedule layouts (time versus quantity).

Research at Virginia Tech

Research at Virginia Tech [Jost, 92] aims at studying the behavior and performance of repetitive activities under work conditions that may involve interruption of the work. When a repetitive activity is interrupted, an unlearning or forgetting curve is observed. Due to the lack of empirical data that define such phenomena, additional research is needed to investigate such effect.

The current research aims at investigating this unlearning curve phenomena and how the amount of unlearning is related to the location on the learning curve at which a break occurs. The research may also investigate the effect of the duration of an interruption on the learning curve.

10.5 Conclusion

It is highly desirable to integrate work space requirements as a construction constraint along with other constraints in developing schedules. At times, it may be even advantageous to develop a critical path simply based on congestion when all other influencing variables remain unchanged. The integration of work space availability as a constraint in the scheduling process provides a number of benefits such as: generation of a more realistic schedule, reducing delays and improving productivity.

The research presented in this dissertation provides a significant step in developing more effective schedules by recognizing work space as a new constraint which has been ignored by current planning and scheduling tools. The developed SCaRC scheduling system provides for a method that integrates this new space constraint with resource constraints and other constraints to develop schedules.

The system demonstrates the potential to tackle the problem of congestion. Although several assumptions and simplifications underlay the current prototype, the system can be used to generate reliable and effective schedules. Construction planning and scheduling of multi-story buildings can greatly benefit from the system.

The main component of the system is comprised of several processors or procedures each responsible for verifying a specific constraint. The processors are linked in a defined sequence to a main module that controls the execution of these processors. This structuring of the system allows the ease of adding, modifying, or delaying one or more of the processors. This allows for adapting the system to be utilized for a multitude of domains.

Although the domain of the system focuses on the repetitive floors of multi-story buildings, much of the concept and theory developed and implemented can be adopted for other domains. The scheduling knowledge and decision factors stored in the knowledge base of the system can be easily modified and customized to other types of work.

Bibliography

[Arditi, 86]	Arditi, D., and Zeki, A., *"Line-of-Balance Scheduling in Pavement Construction,"* Journal of Construction Engineering, ASCE, Vol. 112, No. 3, Sept 1986, pp. 411-424.
[Assad and Wasil, 86]	Assad, A.A., and Wasil, E.A., *"Project Management using a Microcomputer,"* Computers and Operations Research, Vol 13, No. 2/3, pp. 231-260, 1986.
[Barrie & Paulson, 84]	Barrie, Donald S., and Paulson Jr., Boyd C., *Professional Construction Management,* McGraw Hill, 2nd ed., 1984.
[Borcherding et al., 87]	Borcherding, J.D., Sebastian, S.J., and Samelson, N.M., *"Improving Motivation and Productivity on Large Projects,"* Journal of the Construction Division, Vol 106, No. C01, March 1980, pp. 73-82.
[Burgess et al., 62]	Burgess, A. R. and Killebrew, J. B., *"Variation in Activity Level on a Cyclic Arrow Diagram,"* Journal of Industrial Engineering, March-April 1962.
[Carr, 74]	Carr, R. I., and Mayer, W. L., *"Planning Construction of Repetitive Building Units,"* Journal of the Construction Division, ASCE, Vol 100, No, CO3, Sept. 1974, pp 403-412.
[Charazanwski, 86]	Charaznwski, E. N., and Johnston, D., *"Application of Linear Scheduling,"* Journal of the Construction Division, ASCE, Vol. 112, No. 4, Dec 1986, pp. 476-491.

[Clough, 81]	Clough, R. H., *Construction Contracting*, John Wiley & Sons, 4th edition, 1981.
[Darwiche, 89]	Darwiche, A., Levitt, R. E., and Hayes-Roth, B., *"OARPLAN: Generating Project Plans by Reasoning about Objects, Actions and Resources,"* Stanford University, Draft Copy, 1989.
[Davis and Patterson, 75]	Davis, E. and Patterson, J.h., *"A Comparison of Heuristic and Optimum Solutions in Resource-constrained Project Scheduling,"* Management Science, Vol 21, No. 8, pp. 944-955, April 1975.
[De La Garza, 88]	De La Garza, J. M., *"A Knowledge Engineering Approach to the Analysis and Evaluation of Schedules for Mid-Rise Construction,"* Ph.D. Dissertation submitted to the University of Illinois at Urbana-Champaign, Department of Civil Engineering, 1988.
[Echeverry, 89]	Echeverry, D., Ibbs, C. W., and Kim, S., *"A Knowledge-Based Approach to Support the Generation of Construction Schedules,"* submitted to the International Conference: Application of Artificial Intelligence Techniques to Civil and Structural Engineering, Sept 19-21 1989, London, UK.
[Fenves, 86]	Fenves, S. J., *"What is an Expert System,"* Expert Systems in Civil Engineering, Edited by C. N. Kostem and M. L. Maher, ASCE, 1986, pp. 1-6. [GRAY,86] Gray, C., *"Intelligent Construction Time and Cost Analysis,"* Construction Management and Economics Journal, 1986.
[Frenzel, 87]	Frenzel, Louis E., Jr., *"Understanding Expert Systems,"* Sams Understanding Series, Howard W. Sams & Company, Indianapolis, 1987.

[Gordon, 74] Gordon, J. H., *"Heuristic Methods in Resource Allocation,"* Proceedings of the 4th Internet Conference, Paris, 1974.

[Harbison-Briggs, 89] Harbison-Briggs, K., and MaGraw, K. L., "*Knowledge Acquisition: Principles and Guidelines,* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[Higazi, 89] Higazi, A. M., *"Simulation Analysis of Linear Construction Processes,"* Ph.D. Dissertation submitted to the University of Purdue, Civil Engineering Department, April 1989.

[Hoffman, 87] Hoffman, R. R., *"The Problem of Extracting Knowledge from Experts,"* The AI Magazine, Vol. 8, No. 2, Summer 1987, pp. 53-67.

[Johnston, 81] Johnston, D. W., *"Linear Scheduling Method for Highway Construction,"* Journal of the Construction Division, ASCE, Vol. 107, No. CO2, June 1981, pp. 247-261.

[Jost, 92] Jost, D., "The Effect of Interruptions on Repetitve Construction Activities," a Preliminary Proposal, Virginia Tech, Blacksburg, VA, 1992.

[Kartam, 89] Kartam, N., Levitt, R. E., *"Intelligent Planning of Construction Projects with Repeated Cycles of Operation,"* submitted to the Journal of Computing in Civil Engineering, ASCE, Special Issue: Knowledge-based Approaches to Planning and Design, Fall 1989.

[Kavanagh et al., 81] Kavanagh, T.C., Okamoto, R. Y., Friedman, R., *Planning and Environmental Criteria for Tall Buildings,* American Society of Civil Engineers, 1981.

[Khisty, 70]	Khisty, C. J., *"The Application of the Line of Balance Technique to the Construction Industry,"* Indian Concrete Journal, Vol. 44, No. 7, July 1970, pp. 297-300 and 319-320.
[Levitt, 87]	Levitt, R. E., *"Expert Systems in Construction, State of the Art,"* in Expert Systems for Civil Engineering: Technology and Applications, Maher, M. L., Ed., ASCE, New York, Ch.6, pp. 85-112, 1987.
[Levitt, 87]	Levitt, R. E., Kunz, J., *"Using Artificial Intelligence Techniques to Support Project Management,"* Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Vol. 1, No.1, 1987, pp. 3-24.
[Lumsden, 65]	Lumsden, P., *The Line-of-Balance Method,* Pergamon Press Limited, 1965.
[Maher, 87]	Maher, Mary Lou, *"Expert Systems for Civil Engineers: Technology and Application,"* ASCE, 1987.
[Mawesley et al., 90]	Mawdesley, M.J., Askew, W.H., Lees, J., Taylor, J., and Stevens, C., *"Time Change Charts for Scheduling Linear Projects,"* Computing in Civil Engineering, pp 613-620.
[Moder & Philips, 70]	Moder, J. J., and Philips, C. R., *Project Management with CPM and PERT,* Van Nostrand Reinhold Company, 2nd edition, 1970.
[Moder, Philips, & Davis, 83]	Moder, J. J., Philips, C. R., and Davis, E.W., *Project Management with CPM and PERT,* Van Nostrand Reinhold Company, 2nd edition, 1983.
[Morad, 90]	Morad, A., *"Geometric-based Reasoning System for Project Planning Utilizing AI and CAD*

	Technologies," Dissertation submitted to the Virginia Polytechnic Institute and State University, Civil Engineering, May 1990.
[Navinchandra, 88]	Navinchandra, D., Sriram, D., and Logcher, R. D., *"GHOST: Project Network Generator,"* Journal of Computing in Civil Engineering, ASCE, Vol. 2, No. 3, 1988.
[O'Brien, 75]	O'Brien, J. J., *"VPM Scheduling for High Rise Buildings,"* Journal of the Construction Division, ASCE, Vol 101, No. CO4, Dec 1975, pp. 105-116.
[Patterson and Huber, 74]	Patterson, J.H., and Huber,W.D., *"A Horizon-Varying, Zero-One Approach to Project Scheduling,"* Management Science, Feb., 1974.
[Polya, 57]	Polya, G., *"How to Solve it,"* Double-day Anchor Books, N.Y. 1957.
[Stradal, 82]	Stradal, O., Cacha, J., *"Time Space Scheduling Method,"* Journal of the Construction Division, ASCE, Vol. 108, No. CO3, Sept. 1982, pp. 445-457.
[Smith, 87]	Smith, D. M., *"An Investigation of the Space Constraint Problem in Construction Planning,"* Masters Project Paper, submitted to the Faculty of the Virginia Polytechnic Institute and State University, June 1987.
[Stinson et. al., 78]	Stinson, J.P., Davis, E.W., and Khumawala, B., *"Multiple Resource Constrained Scheduling using Branch and Bound,"* AIIE Transactions, September 1978.
[Tommelein et al, 92]	Tommelein, I. D., Castillo, J. G., Zouein, P. P.,

"Space-Time Characterization for Resource Management on Construction Sites," Proceedings of the Eighth Conference on Computing in Civil Engineering, pp. 623-630, Dallas, Texas, June 1992.

[Waterman, 86] Waterman, D. A., *A Guide to Expert Systems*, Addison-Wesley Publishing Company 1986.

[Wright, 79] Wright, Paul H., Paquette, Radnor J., *"Highway Engineering,"* 4th edition, John Wiley & Sons, 1979.

[Zozaya et al, 89] Zozaya-Gorostiza, C., Hendrickson, C., Rehak, D. R., *"Knowledge-based Process Planning for Construction and Manufacturing,"* Academic Press, San Diego, CA, 1989.

Appendix A

I/O DATA FLOW IN THE KNOWLEDGE-BASE:
The External Data Interface Module

As discussed earlier in chapter 9, all required input data defined in the database system is stored in several ASCII type files. The files are generated from the database system after all input data definition is completed. There are seven interface files generated: *ACT-GEN.art, ACT-PAR.art, ACT-RES.art, ACT-SPC.art, RES.art, BLOCK.art, and OTHER.art.* Data stored in these files is extracted by the knowledge-based scheduling system for processing and generating a schedule. Once a schedule is generated, scheduling results are printed by the knowledge-based system to an output ASCII file (OUTPUT.art). Data stored in the file is then extracted from the database system to review the scheduling results.

As illustrated in Figure A.1, the flow of input and output (I/O) data through the knowledge-Based system is manipulated by the *External Data Interface Module*. The module is one of three modules that constitute the knowledge-based scheduling system. The external data interface module performs two functions. The module is responsible for extracting data from the input ASCII files and storing it in the context of the scheduling system. The module is also responsible for printing the results of the generated schedule to an output ASCII file. These functions are performed by two processors; a *Data Input Processor*, and a *Data Output Processor*.

The execution of the data input and data output processors of the external data interface module is controlled by another module; the controller module. The module controls the sequence of execution of these processors with respect to other processors of the scheduling system. This is accomplished by manipulating special facts in the contexts that initiates the start of these two processors. The controller module is responsible for defining these facts in the context.

The following sections will discuss the data input and output (I/O) processors in more detail.

A.1 THE DATA INPUT PROCESSOR

The *Data Input Processor* is one of two processors that make up the external data interface module. The objective of the input processor is to extract all input data from the

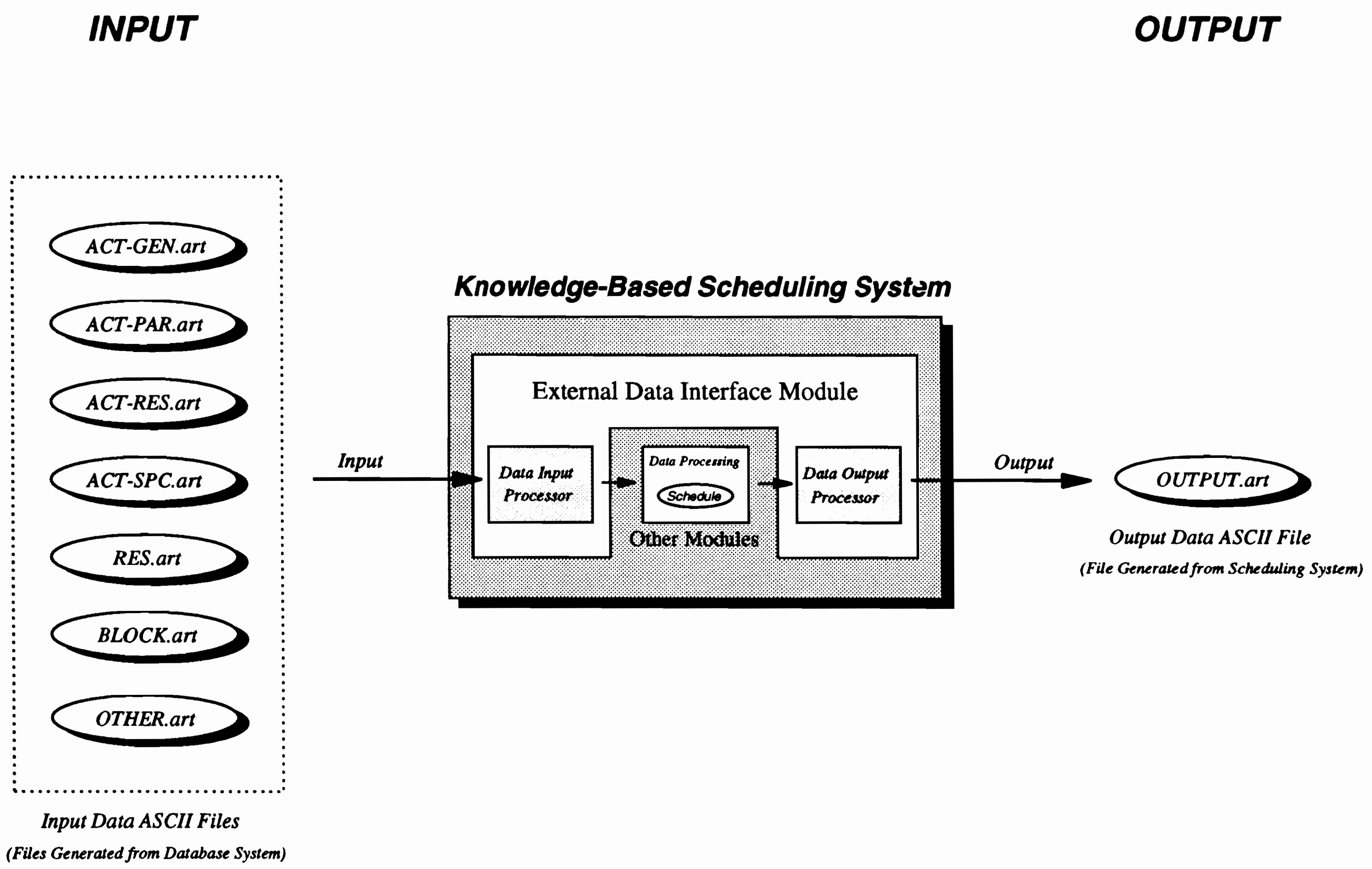


Figure A.1 - Input/Output Data Flow in the Knowlege-Based System

external ASCII files generated from the database system and store the data in the context using mainly a schemata structure. Three schemata structures are created by this processor to store the extracted data: activity schemata, resource schemata, and work-block schemata. Data stored in these schemata is then processed by the other scheduling modules (i.e. the controller module and the sequence generation module) to produce a schedule.

A.1.1 The Data Input Processor Execution Rules

The data input processor consists of several ART-IM rules that extracts the data from the external ASCII files and creates the schemata structure. There are 7 rules classified as follows:

Rule-1: Activity schemata creation and activity general data extraction rule

Rule-2: Activity special data extraction rule

Rule-3: Activity resource demand data extraction rule

Rule-4: Activity space data extraction rule

Rule-5: Resource schemata creation and data extraction rule

Rule-6: Work block schemata creation and data extraction rule

Rule-7: Other project data extraction rule

The first rule (rule-1) is responsible for creating the activity schemata structure and extracting the general activity data from the ACT-GEN.art ASCII file. The second rule (rule-2) is responsible for extracting the special activity data from the ACT-PAR.art

ASCII file. The third rule (rule-3) is responsible for extracting activity resource demand data from the ACT-RES.art ASCII file. The forth rule (rule-4) is responsible for extracting activity work space data from the ACT-SPC.art ASCII file. All extracted activity data is stored in the appropriate slots of each activity schema. Rule-5 creates the resource schemata structure and extracts resource data from RES.art ASCII file. Rule-6 creates the work block schemata structure and extracts work block data from the BLOCK.art ASCII file. The last rule of the input processor extracts other project data stored in the OTHER.art ASCII file. Other project data is stored in the context using an ART-IM global variables format. The data extraction rules of the input processor are illustrated in Figure A.2.

Although the actual format varies from one rule to another, the basic structure of each rule is the same. Each rule has seven main ART-IM functions:

1- ***Def-External-Data*** function. This function is responsible for defining the format of the external data records stored in the external ASCII files (e.g. general activity data records stored in ACT-GEN.art file).

2- ***Alloc-Buffer*** function. This function is responsible for allocating a memory storage area (buffer) to hold each activity record before storing the record data in the context.

3- ***Open*** function. The purpose of this function is to open a stream or path from

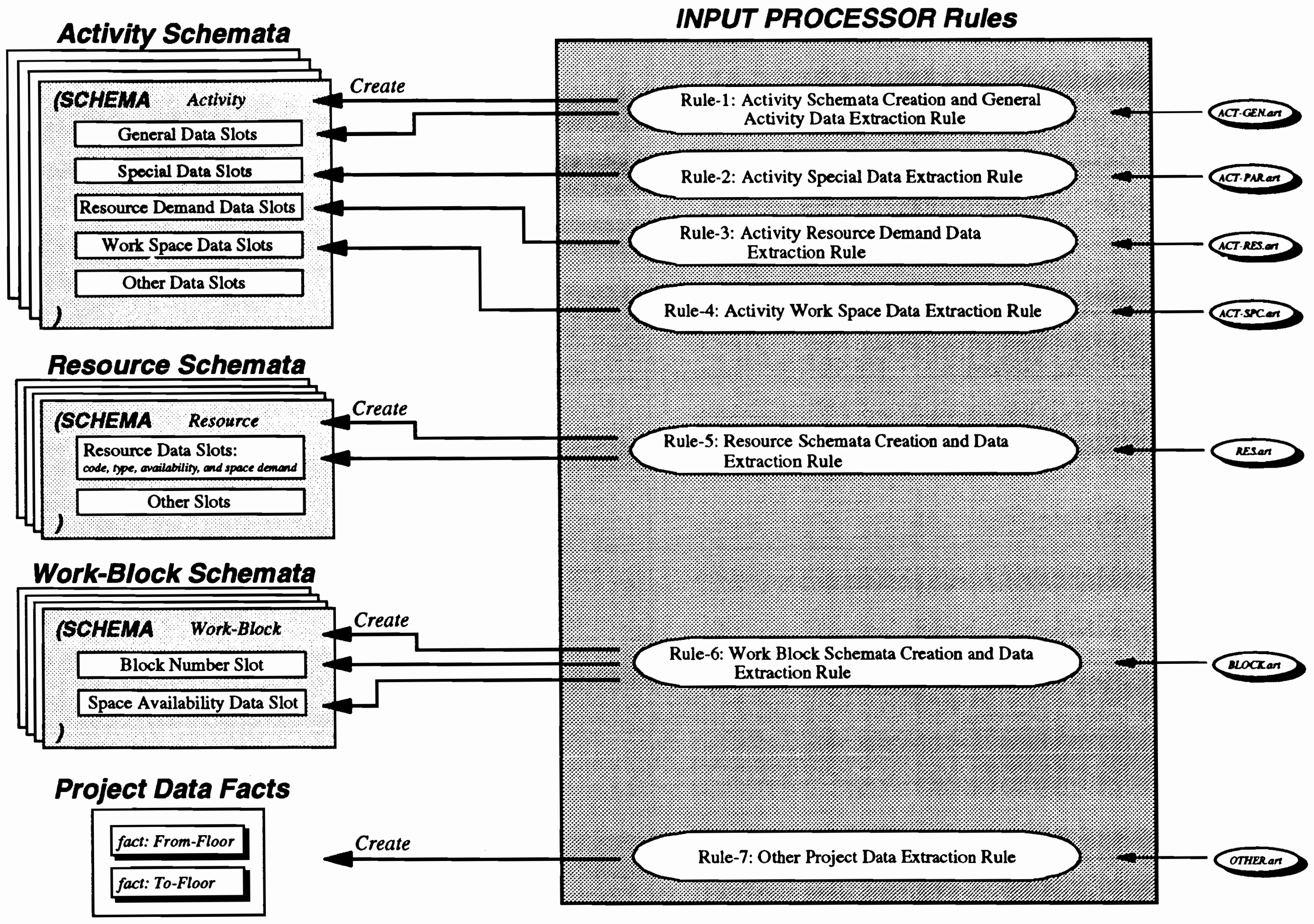

INPUT PROCESSOR Rules
Rule-1: Activity Schemata Creation and General Activity Data Extraction Rule
Rule-2: Activity Special Data Extraction Rule
Rule-3: Activity Resource Demand Data Extraction Rule
Rule-4: Activity Work Space Data Extraction Rule
Rule-5: Resource Schemata Creation and Data Extraction Rule
Rule-6: Work Block Schemata Creation and Data Extraction Rule
Rule-7: Other Project Data Extraction Rule
ACT-GEN.art
ACT-PAR.art
ACT-RES.art
ACT-SPC.art
RES.art
BLOCK.art
OTHER.art
Create
Create
Create
Create
Activity Schemata
(SCHEMA Activity
General Data Slots
Special Data Slots
Resource Demand Data Slots
Work Space Data Slots
Other Data Slots
)
Resource Schemata
(SCHEMA Resource
Resource Data Slots:
code, type, availability, and space demand
Other Slots
)
Work-Block Schemata
(SCHEMA Work-Block
Block Number Slot
Space Availability Data Slot
)
Project Data Facts
fact: From-Floor
fact: To-Floor

Figure A.2 - Input Processor Data Extraction Rules

the ART-Im to the external data file. The external file name and file path are defined using this function.

4- ***Read-Line*** function. This function is responsible for reading each file one record (or line) at a time.

5- Various mapping functions: ***Map-Schema, Peek, and Poke***. These functions are responsible for manipulating data of each record. With these functions data of each record is transferred from the external file to the memory buffer and then to the appropriate slots of each schema.

6- ***While*** function. This function is used to iterate over the data records of each file to perform the different data extraction tasks.

7- ***Assert*** function. The purpose of this function is to insert an ART-IM fact in the context.

The basic steps performed by each rule to extract data from the external ASCII files and store it in the context are as follows:

Step-1 Define the format of the external data records of each file. This is accomplished using the ART-IM function *Def-External-Data.*

Step-2 Allocate a memory buffer for records of each external file. The ART-IM function *Alloc-Buffer* is used to define the memory buffer for each file. Each allocated memory buffer must be big enough to hold the largest record to which it is allocated. The size of the buffer to be allocated is defined by the ART-IM function *Def-External-Data.*

Step-3 Open a stream for each data file to read its records. The stream should specify the file name and path of the external data file (e.g. a:\data\ACT-GEN.art). The ART-IM function *Open* is used to define a stream for each file.

Step-4 Read the external data file record by record. Using the ART-IM function *Read-Line*, read the data in the external file one record at a time.

Step-5 Place each record in the allocated buffer. Each record is temporary placed in the buffer using the ART-IM function *Poke.*

Step-6 Send the data from the buffer to the context. Using the function *Map-Schema*, data is transferred from the buffer to the proper slots of each schema. If the schema is not defined, it must be first created. The schema name is defined using the ART-IM function *Peek.*

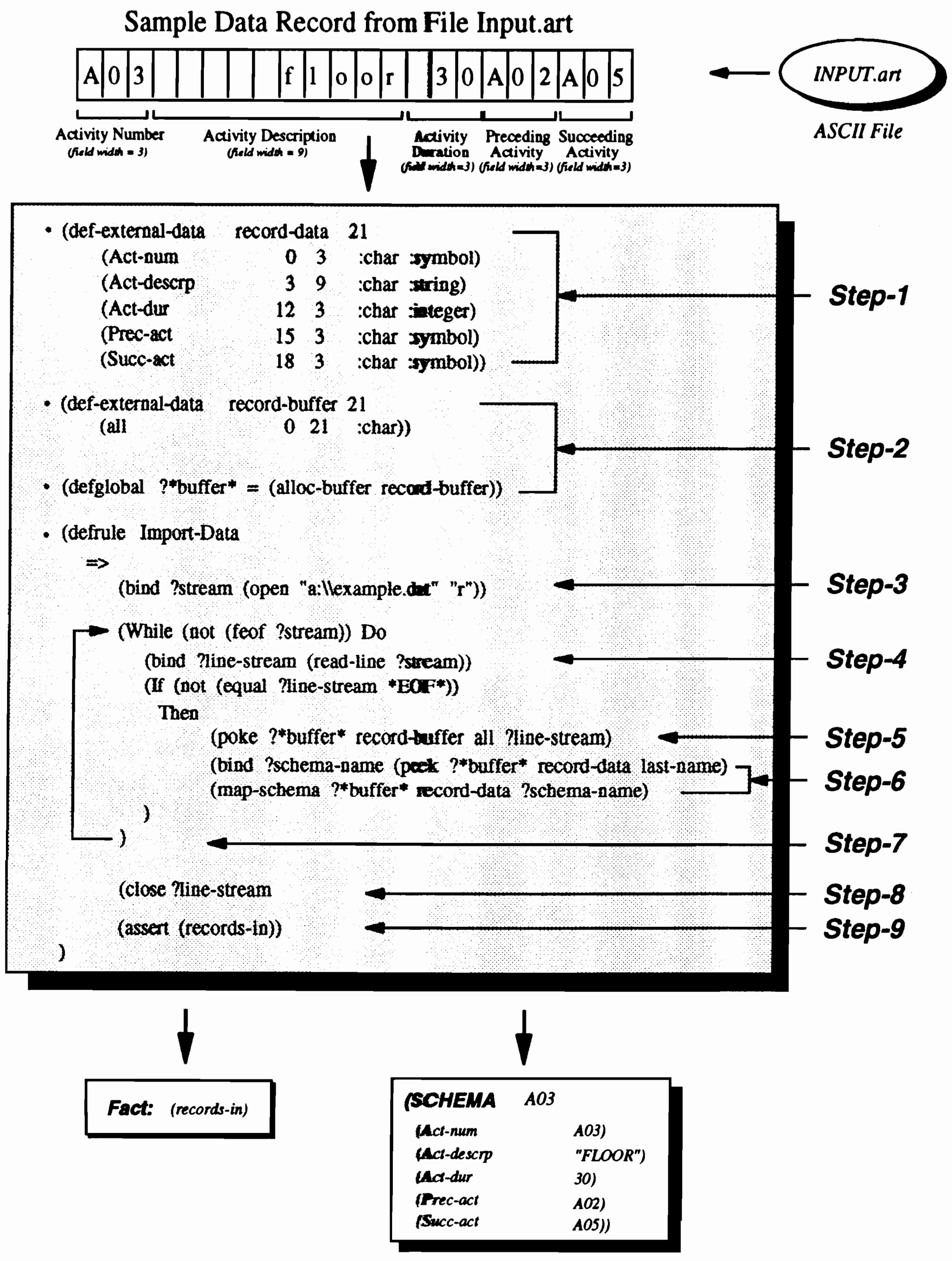
Step-7 Repeat steps 4, 5 and 6 for all records of the external data file. The ART-IM function *While* is used to iterate over all records of each file. The function terminates the data extraction process once all records are read and the end of file is reached.

Step-8 Close the open stream to the external file. This is accomplished using the ART-IM function *Close*.

Step-9 Insert a flag in the context in the form of an ART-IM fact using the function *Assert*. The inserted fact is specific to each rule. The purpose of each fact is to indicate to the controller module that the execution of the rule is completed and data is extracted.

Figure A.3 depicts an example code listing illustrating these steps. The example extracts data from an external file (Input.art) stored on drive (a:). The sample data record to be extracted contains five fields: activity number, activity descriptions, activity duration, preceding activity, and succeeding activity with field widths of 3, 9, 3, 3, and 3 respectively. The extracted data from all fields is stored using a schema format.

Figures A.4 through A.10 depict flow charts of the execution logic of each rule. Some ART-IM terminology is used to link the flow chart to the actual code of each rule. The complete code listing of the input processor rules is provided in Appendix G.


Sample Data Record from File Input.art
A 0 3 f l o o r 3 0 A 0 2 A 0 5
Activity Number (field width = 3)
Activity Description (field width = 9)
Activity Duration (field width=3)
Preceding Activity (field width=3)
Succeeding Activity (field width=3)
INPUT.art
ASCII File
Step-1
Step-2
Step-3
Step-4
Step-5
Step-6
Step-7
Step-8
Step-9
Fact: (records-in)

```
• (def-external-data   record-data  21
      (Act-num              0   3    :char :symbol)
      (Act-descrp           3   9    :char :string)
      (Act-dur             12   3    :char :integer)
      (Prec-act            15   3    :char :symbol)
      (Succ-act            18   3    :char :symbol))

• (def-external-data   record-buffer  21
      (all                  0  21    :char))

• (defglobal  ?*buffer* = (alloc-buffer record-buffer))

• (defrule  Import-Data
      =>
        (bind ?stream (open "a:\\example.dat" "r"))

        (While (not (feof ?stream)) Do
          (bind ?line-stream (read-line ?stream))
          (If (not (equal ?line-stream *EOF*))
           Then
                (poke ?*buffer* record-buffer all ?line-stream)
                (bind ?schema-name (peek ?*buffer* record-data last-name)
                (map-schema ?*buffer* record-data ?schema-name)
          )
        )

        (close ?line-stream

        (assert (records-in))
  )
```

```
(SCHEMA    A03
  (Act-num        A03)
  (Act-descrp     "FLOOR")
  (Act-dur        30)
  (Prec-act       A02)
  (Succ-act       A05))
```

Figure A.3 - Example Code for External Data Extraction

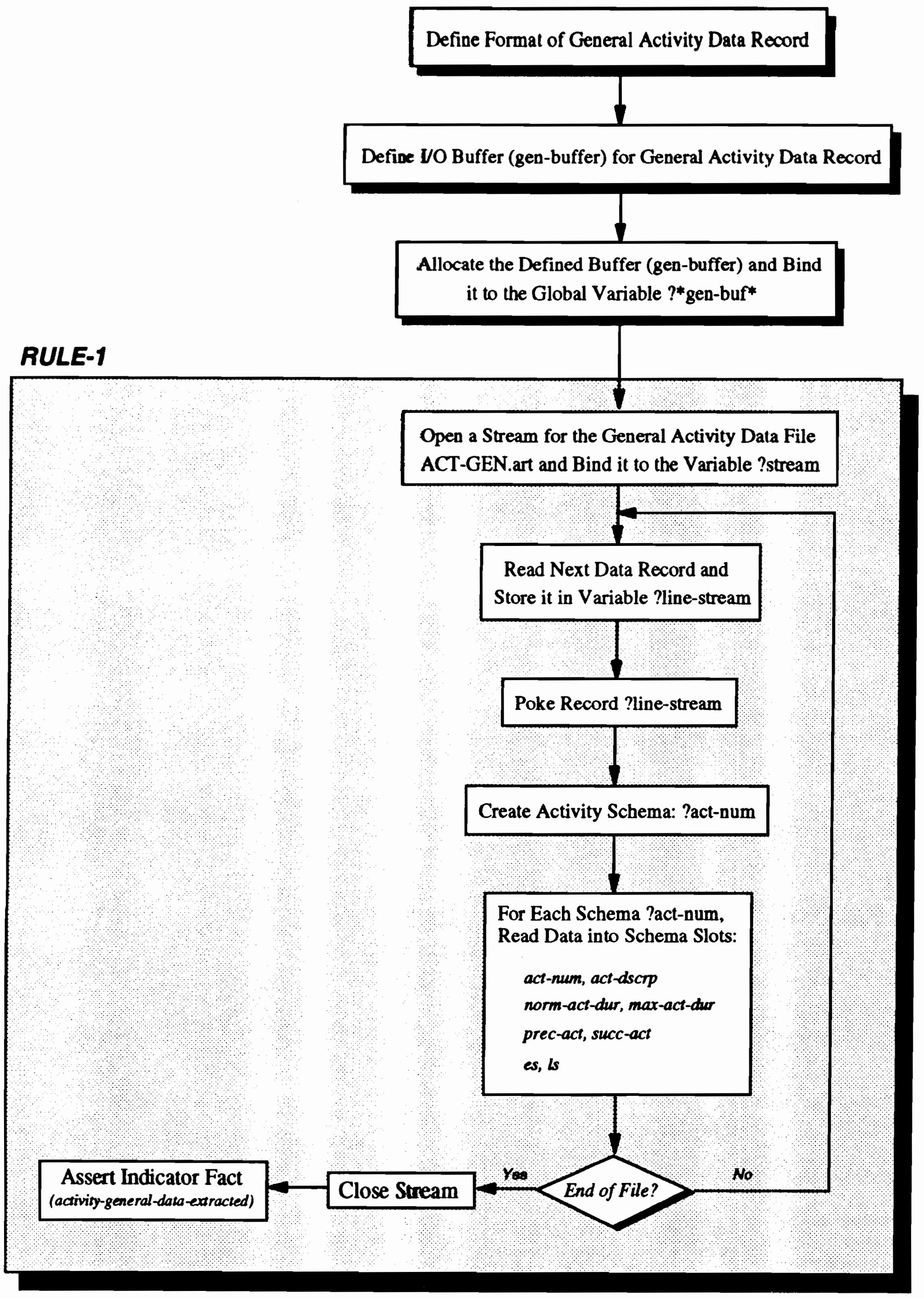

Define Format of General Activity Data Record
Define I/O Buffer (gen-buffer) for General Activity Data Record
Allocate the Defined Buffer (gen-buffer) and Bind it to the Global Variable ?*gen-buf*
RULE-1
Open a Stream for the General Activity Data File ACT-GEN.art and Bind it to the Variable ?stream
Read Next Data Record and Store it in Variable ?line-stream
Poke Record ?line-stream
Create Activity Schema: ?act-num
For Each Schema ?act-num, Read Data into Schema Slots:
act-num, act-dscrp
norm-act-dur, max-act-dur
prec-act, succ-act
es, ls
End of File?
Yes
No
Close Stream
Assert Indicator Fact
(activity-general-data-extracted)

Figure A.4 - External Data Extraction Rule-1

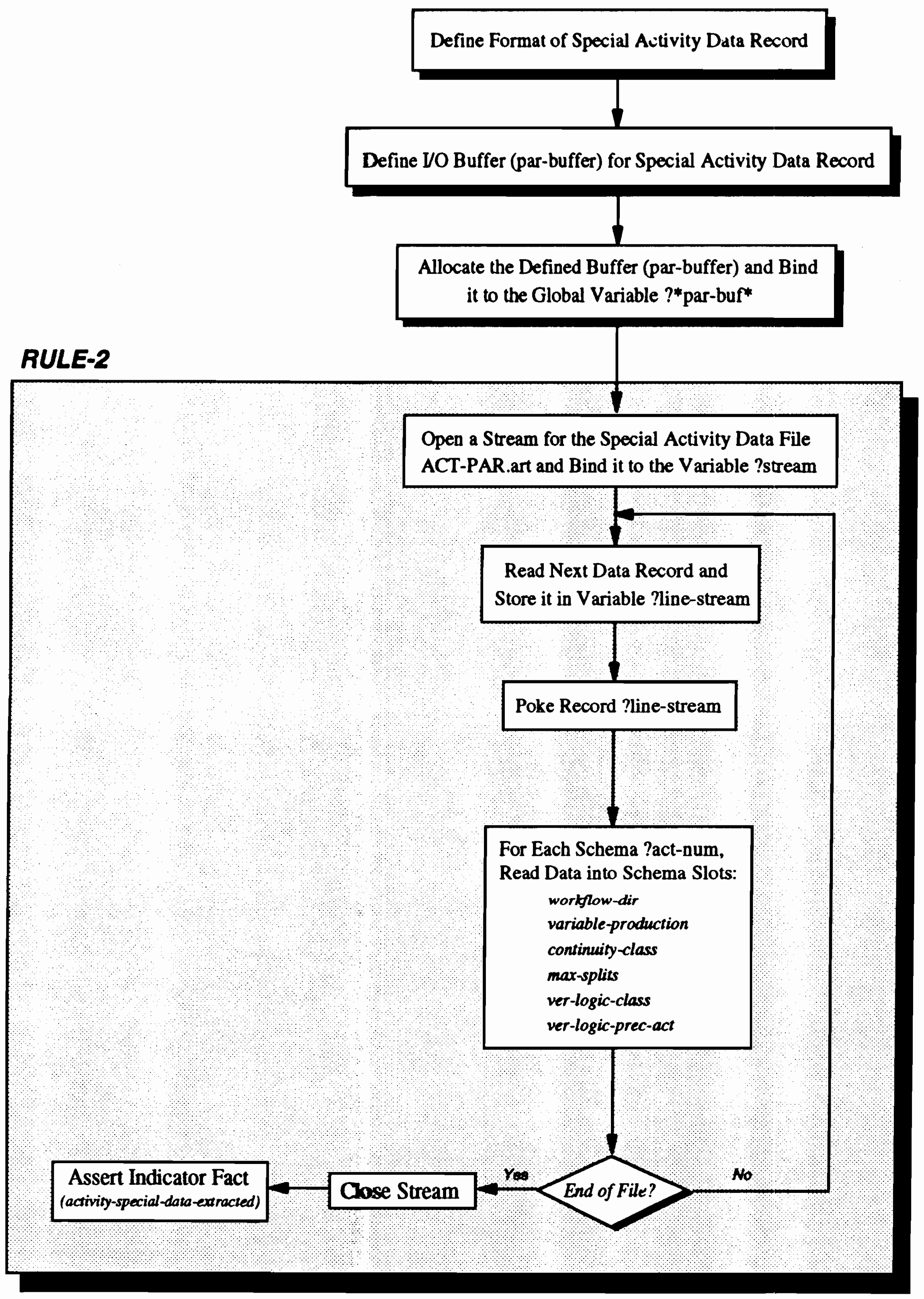

Define Format of Special Activity Data Record
Define I/O Buffer (par-buffer) for Special Activity Data Record
Allocate the Defined Buffer (par-buffer) and Bind it to the Global Variable ?*par-buf*
RULE-2
Open a Stream for the Special Activity Data File ACT-PAR.art and Bind it to the Variable ?stream
Read Next Data Record and Store it in Variable ?line-stream
Poke Record ?line-stream
For Each Schema ?act-num, Read Data into Schema Slots:
workflow-dir
variable-production
continuity-class
max-splits
ver-logic-class
ver-logic-prec-act
End of File?
Yes
No
Close Stream
Assert Indicator Fact (activity-special-data-extracted)

Figure A.5 - External Data Extraction Rule-2

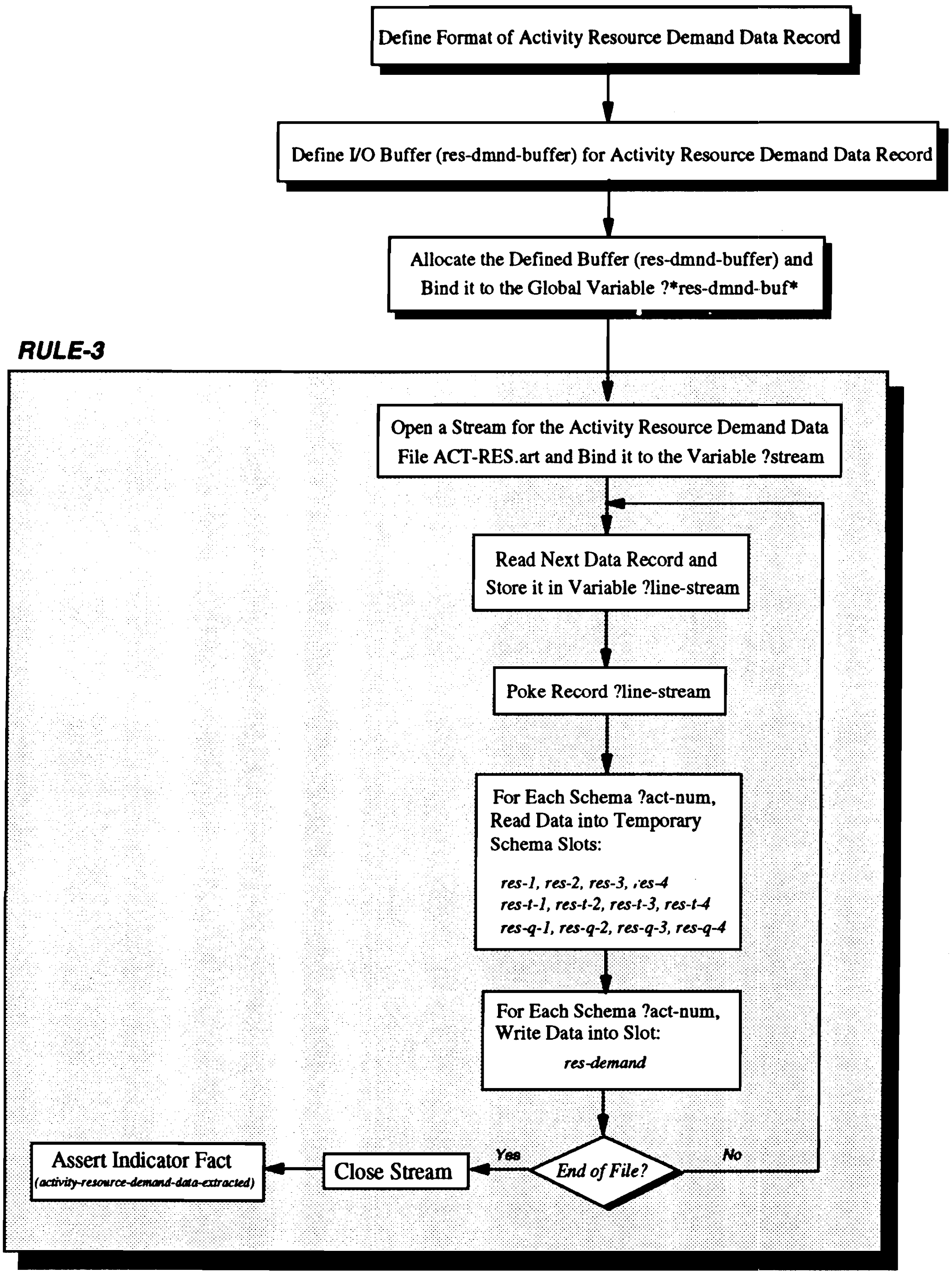

Define Format of Activity Resource Demand Data Record
Define I/O Buffer (res-dmnd-buffer) for Activity Resource Demand Data Record
Allocate the Defined Buffer (res-dmnd-buffer) and Bind it to the Global Variable ?*res-dmnd-buf*
RULE-3
Open a Stream for the Activity Resource Demand Data File ACT-RES.art and Bind it to the Variable ?stream
Read Next Data Record and Store it in Variable ?line-stream
Poke Record ?line-stream
For Each Schema ?act-num, Read Data into Temporary Schema Slots:
res-1, res-2, res-3, res-4
res-t-1, res-t-2, res-t-3, res-t-4
res-q-1, res-q-2, res-q-3, res-q-4
For Each Schema ?act-num, Write Data into Slot:
res-demand
End of File?
Yes
No
Close Stream
Assert Indicator Fact
(activity-resource-demand-data-extracted)

Figure A.6 - External Data Extraction Rule-3

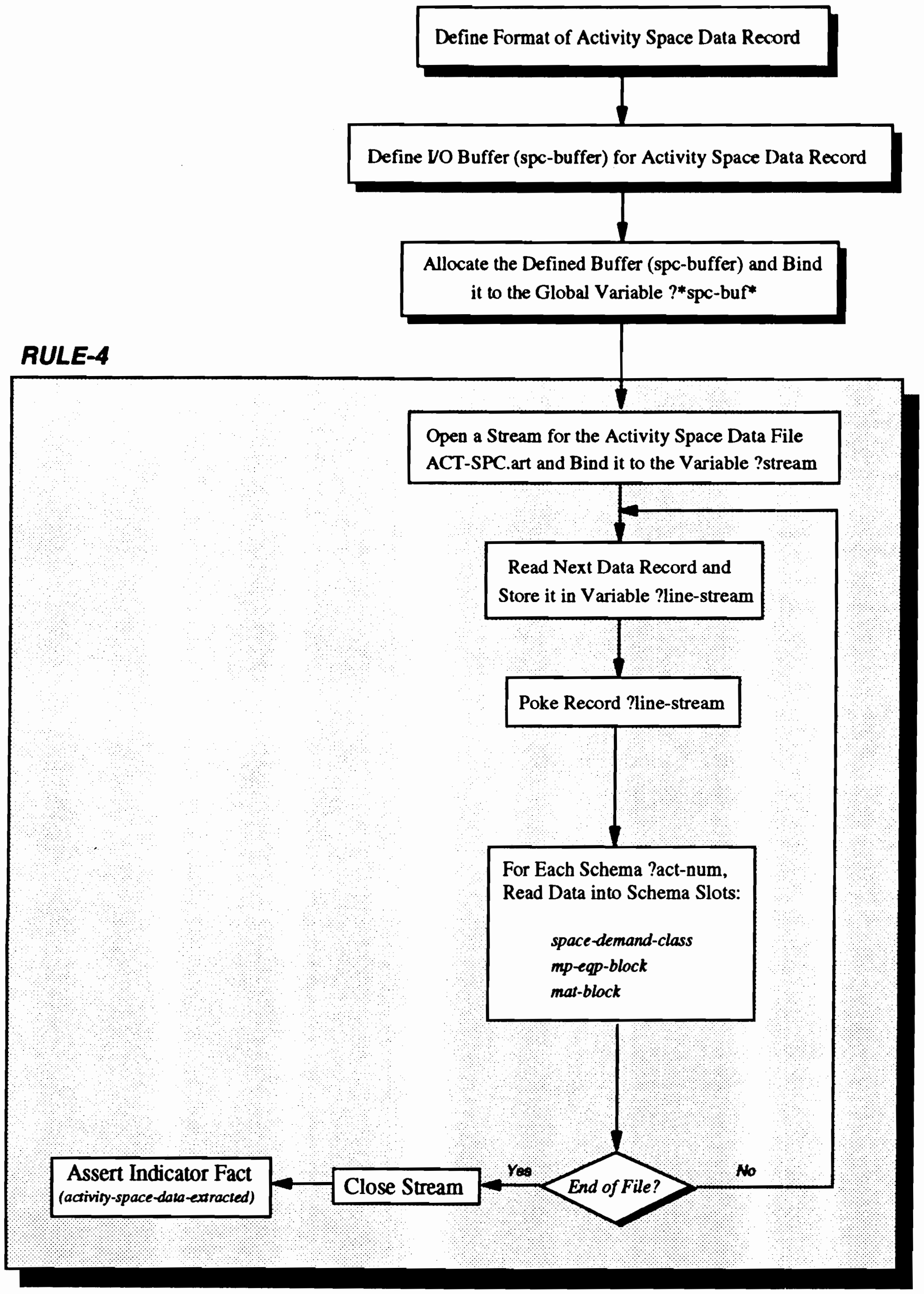

Define Format of Activity Space Data Record
Define I/O Buffer (spc-buffer) for Activity Space Data Record
Allocate the Defined Buffer (spc-buffer) and Bind it to the Global Variable ?*spc-buf*
RULE-4
Open a Stream for the Activity Space Data File ACT-SPC.art and Bind it to the Variable ?stream
Read Next Data Record and Store it in Variable ?line-stream
Poke Record ?line-stream
For Each Schema ?act-num, Read Data into Schema Slots:
space-demand-class
mp-eqp-block
mat-block
End of File?
Yes
No
Close Stream
Assert Indicator Fact
(activity-space-data-extracted)

Figure A.7 - External Data Extraction Rule-4

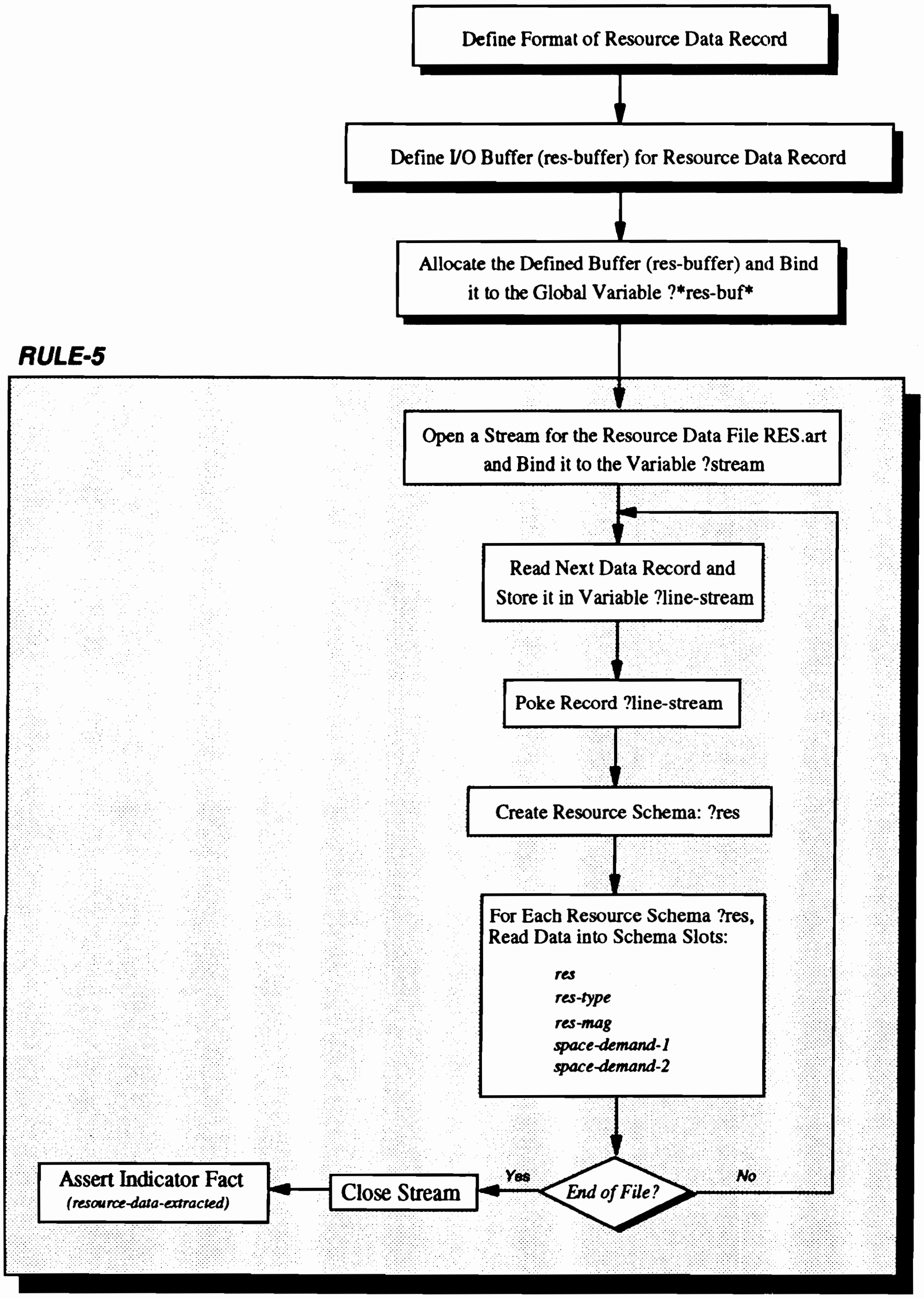

Define Format of Resource Data Record
Define I/O Buffer (res-buffer) for Resource Data Record
Allocate the Defined Buffer (res-buffer) and Bind it to the Global Variable ?*res-buf*
RULE-5
Open a Stream for the Resource Data File RES.art and Bind it to the Variable ?stream
Read Next Data Record and Store it in Variable ?line-stream
Poke Record ?line-stream
Create Resource Schema: ?res
For Each Resource Schema ?res, Read Data into Schema Slots:
res
res-type
res-mag
space-demand-1
space-demand-2
End of File?
Yes
No
Close Stream
Assert Indicator Fact
(resource-data-extracted)

Figure A.8 - External Data Extraction Rule-5

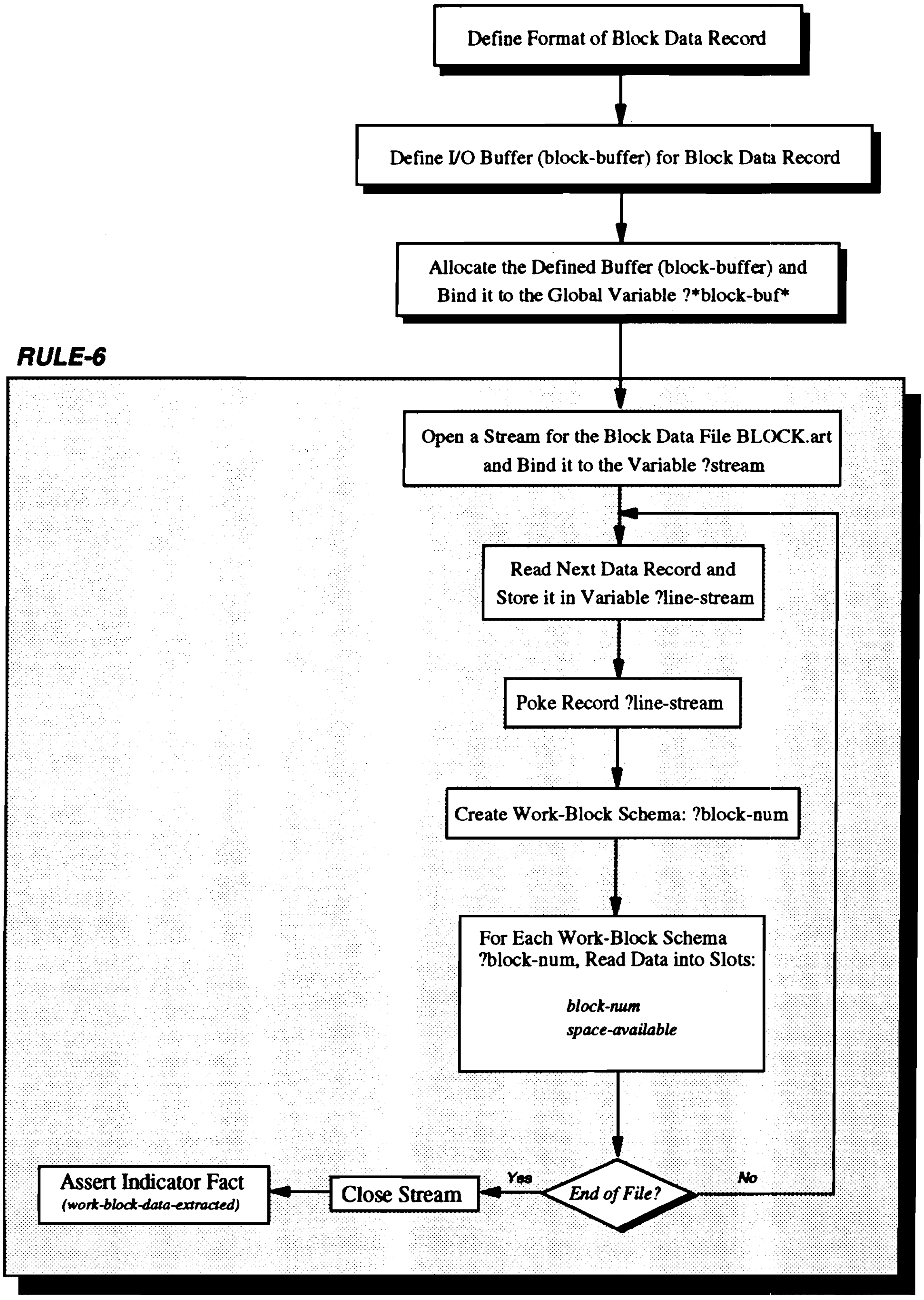

Define Format of Block Data Record
Define I/O Buffer (block-buffer) for Block Data Record
Allocate the Defined Buffer (block-buffer) and Bind it to the Global Variable ?*block-buf*
RULE-6
Open a Stream for the Block Data File BLOCK.art and Bind it to the Variable ?stream
Read Next Data Record and Store it in Variable ?line-stream
Poke Record ?line-stream
Create Work-Block Schema: ?block-num
For Each Work-Block Schema ?block-num, Read Data into Slots:
block-num
space-available
End of File?
Yes
No
Close Stream
Assert Indicator Fact
(work-block-data-extracted)

Figure A.9 - External Data Extraction Rule-6

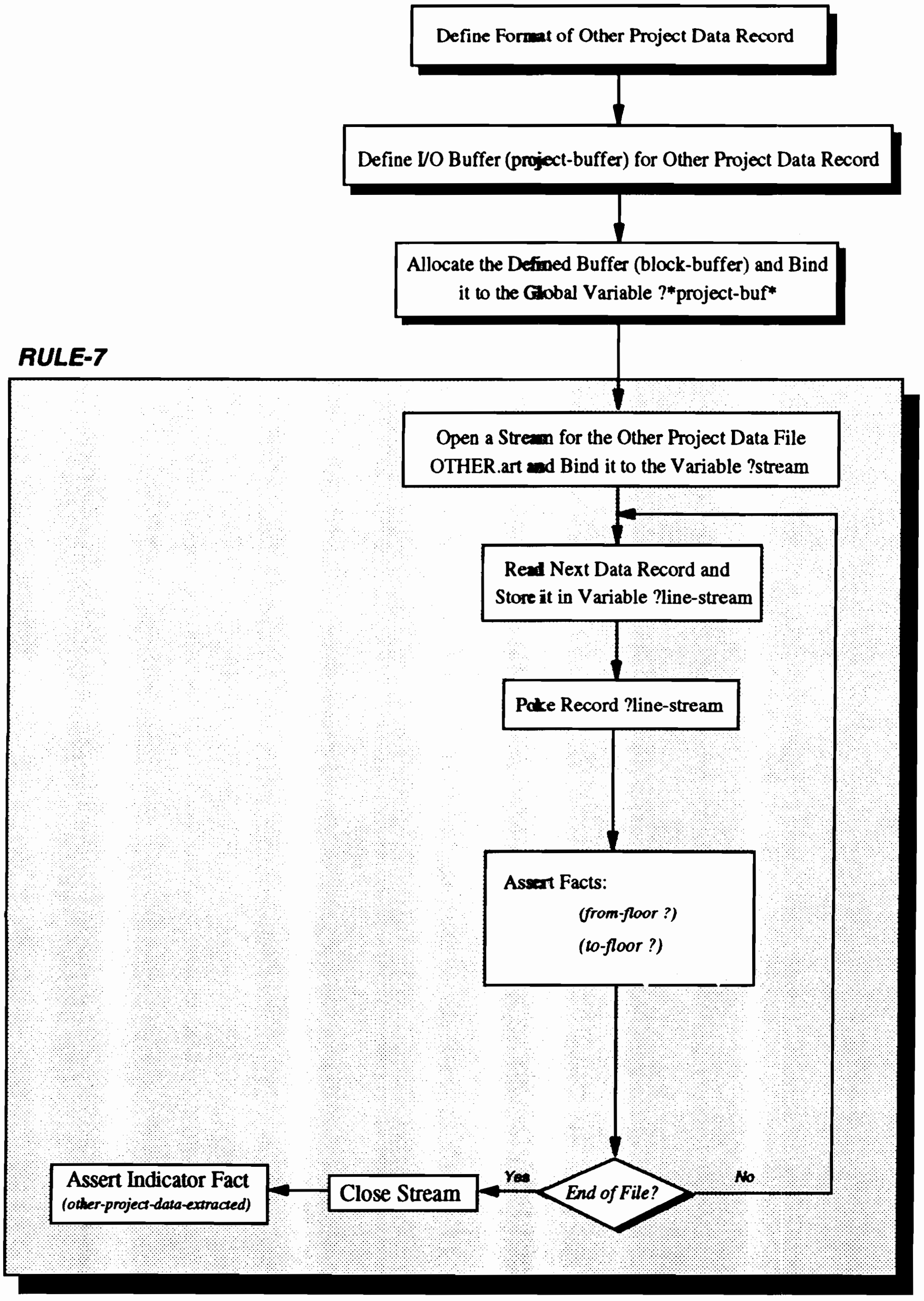

Define Format of Other Project Data Record
Define I/O Buffer (project-buffer) for Other Project Data Record
Allocate the Defined Buffer (block-buffer) and Bind it to the Global Variable ?*project-buf*
RULE-7
Open a Stream for the Other Project Data File OTHER.art and Bind it to the Variable ?stream
Read Next Data Record and Store it in Variable ?line-stream
Poke Record ?line-stream
Assert Facts:
(from-floor ?)
(to-floor ?)
End of File?
Yes
No
Close Stream
Assert Indicator Fact
(other-project-data-extracted)

Figure A.10 - External Data Extraction Rule-7

A.1.2 Control of Execution of the Data Input Processor

When the scheduling system is started, the data input processor rules are the first to be executed (or fired). This priority of execution over other rules is controlled by the controller module. The objective is to extract all input data and create the schemata structure before other rules responsible for the schedule generation can be executed. Once all seven rules of the input processor are fired, control of execution is shifted by the controller module to other processors to generate the schedule.

The control of execution of the input processor by the controller module is achieved by manipulating temporary facts in the context. As illustrated in Figure A.11, when the scheduling system is started, the controller module inserts a fact in the context to initiate the execution of the input processor rules. The inserted fact has the format

fact: *(start-data-extraction)*

As illustrated in Figure A.11, the inserted fact initiates the start of execution of rules 1, 5, 6, and 7. Rules 2, 3, and 4 are only executed when rule-1 is fired. This is because data extracted by these rules requires the activity schemata structure created by rule-1.

Once each rule is fired, it inserts a fact in the context. The purpose of these facts is to indicate to the controller module that the execution of the rule is completed. As depicted in Figure A.11, a total of seven facts are inserted:

fact-1: *(activity-general-data-extracted)*

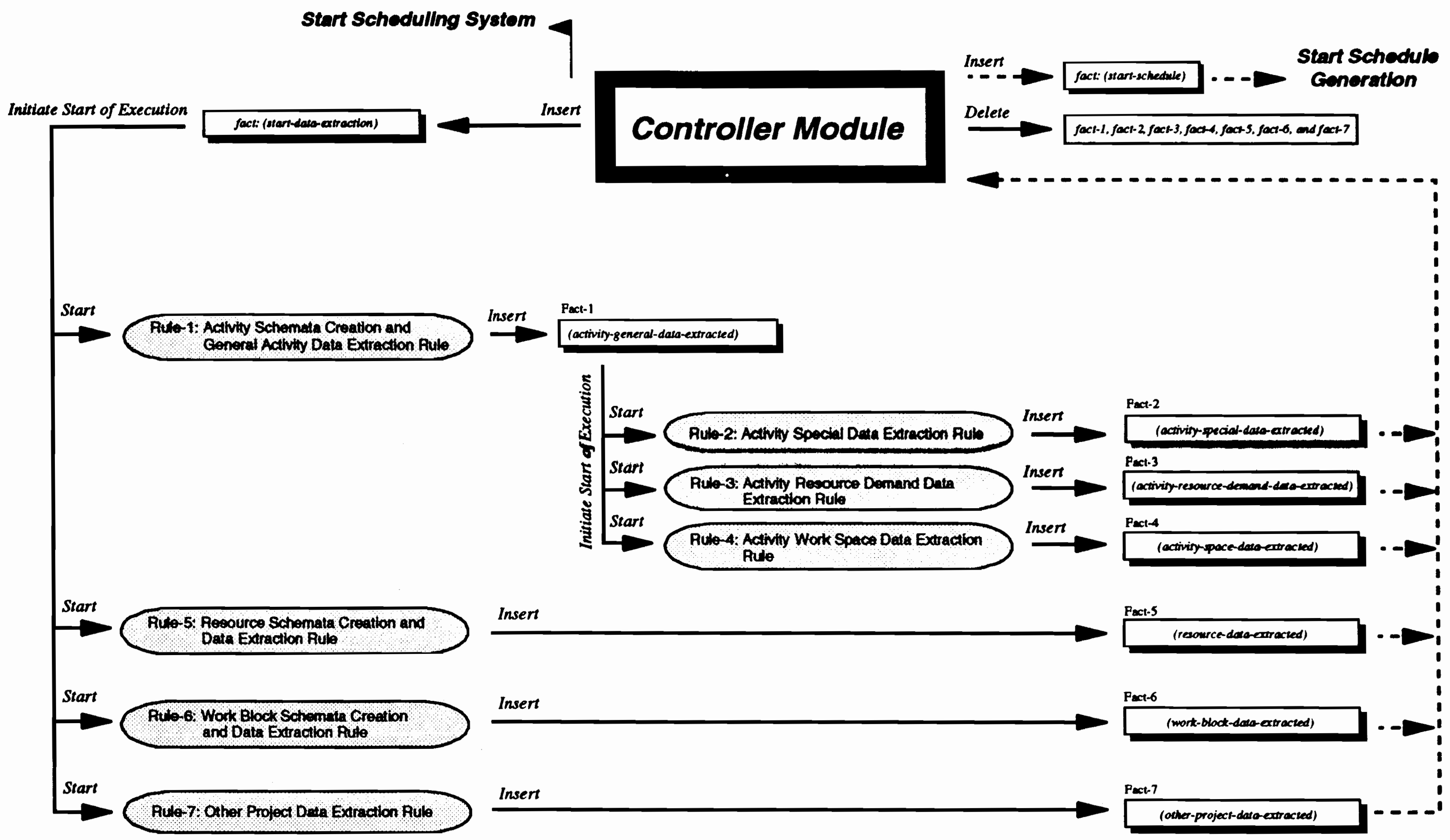

Start Scheduling System
Initiate Start of Execution
fact: (start-data-extraction)
Insert
Controller Module
Insert
fact: (start-schedule)
Start Schedule Generation
Delete
fact-1, fact-2, fact-3, fact-4, fact-5, fact-6, and fact-7
Start
Rule-1: Activity Schemata Creation and General Activity Data Extraction Rule
Insert
Fact-1
(activity-general-data-extracted)
Initiate Start of Execution
Start
Rule-2: Activity Special Data Extraction Rule
Insert
Fact-2
(activity-special-data-extracted)
Start
Rule-3: Activity Resource Demand Data Extraction Rule
Insert
Fact-3
(activity-resource-demand-data-extracted)
Start
Rule-4: Activity Work Space Data Extraction Rule
Insert
Fact-4
(activity-space-data-extracted)
Start
Rule-5: Resource Schemata Creation and Data Extraction Rule
Insert
Fact-5
(resource-data-extracted)
Start
Rule-6: Work Block Schemata Creation and Data Extraction Rule
Insert
Fact-6
(work-block-data-extracted)
Start
Rule-7: Other Project Data Extraction Rule
Insert
Fact-7
(other-project-data-extracted)

Figure A.11 - Control of Execution of Input Processor Rules

fact-2: *(activity-special-data-extracted)*

fact-3: *(activity-resource-demand-data-extracted)*

fact-4: *(activity-space-data-extracted)*

fact-5: *(resource-data-extracted)*

fact-6: *(work-block-data-extracted)*

fact-7: *(other-project-data-extracted)*

Once all seven facts are inserted, the controller module initiates the start of the schedule generation process by deleting all seven facts and inserting a new fact in the context:

fact: *(start-schedule)*

A.2 The Data Output Processor

The *Data Output Processor* is the second processor of the external data interface module. The objective of the output processor is to print the results of the schedule to the output file OUTPUT.art. Data stored in the OUTPUT.art file is extracted from the database system for review of results.

As illustrated in Figure A.12, the processor extracts the scheduling results from each activity schema and prints it to the output file. The output file is directed to the (a:) drive.

For each activity, the processor extracts the following data:

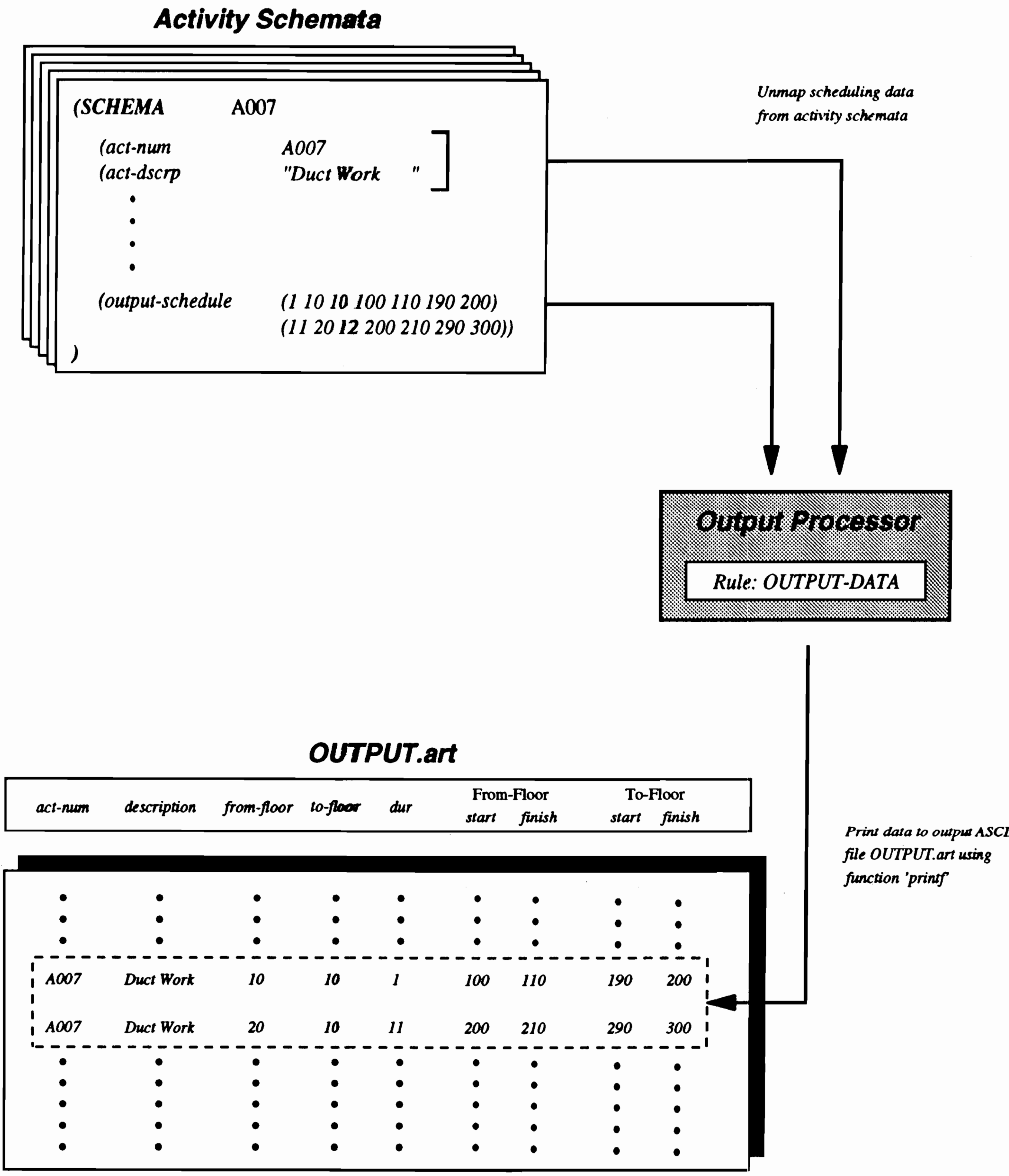


Figure A.12 - Generation of OUTPUT.art ASCII File

1- Activity number (from act-num slot)

2- Activity description (from act-dscrp slot)

3- From-floor number (from output-schedule slot)

4- To-floor number (from output-schedule slot)

5- Activity segment duration (from act-dur slot)

6- From-floor start date (from output-schedule slot)

7- From-floor finish date (from output-schedule slot)

8- To-floor start date (from output-schedule slot)

9- To-floor finish date (from output-schedule slot)

The scheduling data is stored in the file OUTPUT.art using the format shown in Figure A.12. The format is defined by the output processor.

A.2.1 The Data Output Processor Execution Rule

The data output processor consists of one rule (rule: Output-Data). The ART-IM function *Printf* is the main component of the rule. The general format of the function is:

```
(printf  output-stream  field-format

    Data Variables: (activity number)
                    (activity description)
                    (from-floor number)
                    (to-floor number)
                    (activity duration)
                    (from-floor start date)
                    (from-floor finish date)
                    (to-floor start date)
                    (to-floor finish date) )
```

The *Printf* function prints the data to an external file through a defined output stream. The stream directs the data to the output file based on the specified file name and file path. The output stream is defined using the ART-IM function *Open*. The general format of the function is

```
(Open "file-path\\file-name" "w")
```

For example, to direct output data to a file output.??? on drive a:, the format of the function would be:

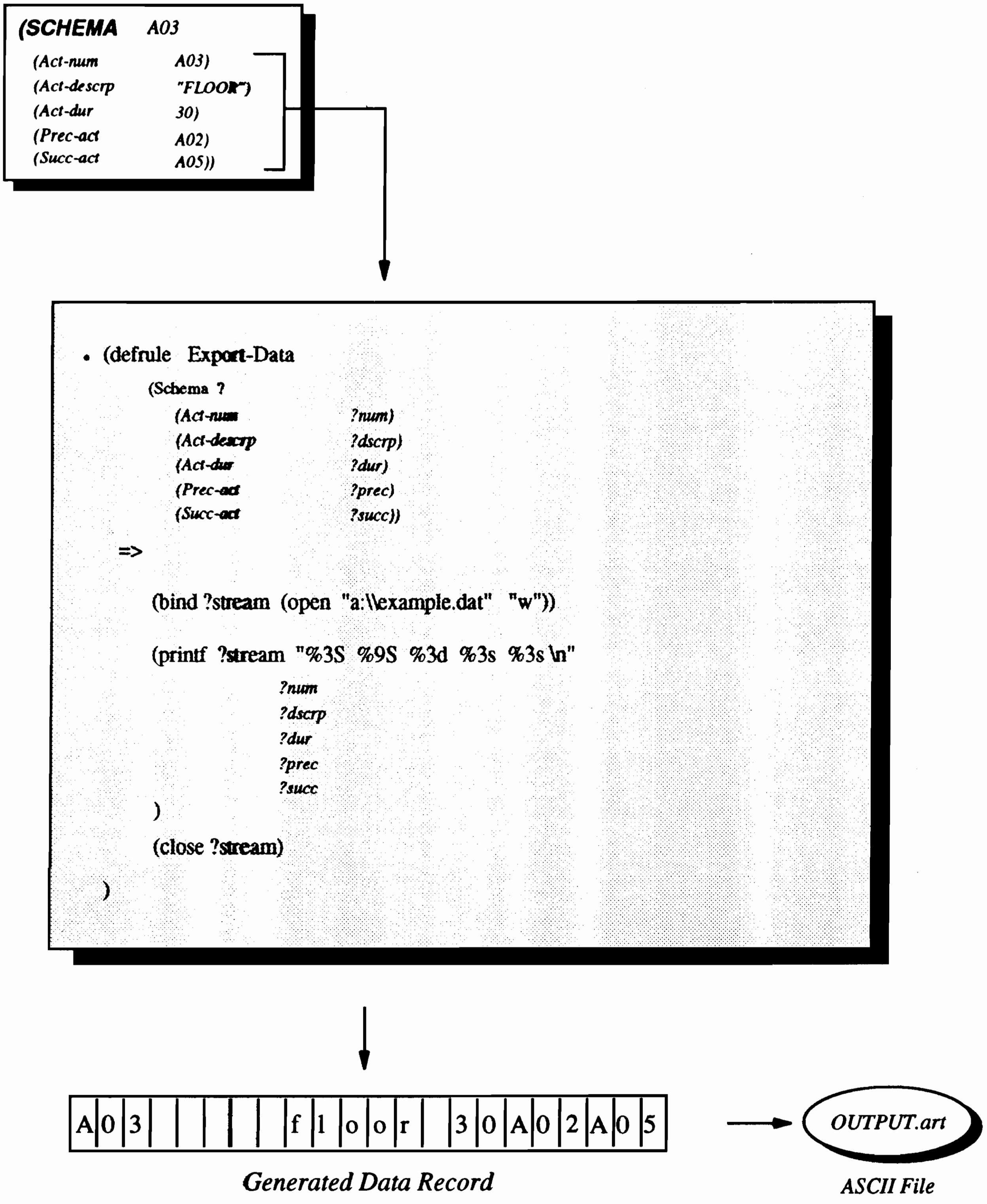
```
(Open "a:\\output.???" "w")
```

The field-format defines the characteristics of each data field printed to the output file. These characteristics include: field width, and type of data (e.g. symbol, integer, float, etc.). Using ART-IM terminology, a field of width 5 defining data of type symbol has the format:

```
field-format:  %4S
```

Figure A.13 depicts an example code listing for printing data to an output file. The example code uses the schema created in the previous example of Figure A.3. Data stored in the schema is printed to the output file Output.art.

Figure A.14 depicts a general flow chart of the execution logic for rule Output-Data. Some ART-IM terminology is used to link the flow chart to the actual code of the rule. The complete code listing of the rule is provided in Appendix G.



```
(SCHEMA   A03
   (Act-num        A03)
   (Act-descrp     "FLOOR")
   (Act-dur        30)
   (Prec-act       A02)
   (Succ-act       A05))
```

```
• (defrule  Export-Data
        (Schema  ?
            (Act-num              ?num)
            (Act-descrp           ?dscrp)
            (Act-dur              ?dur)
            (Prec-act             ?prec)
            (Succ-act             ?succ))
   =>

        (bind ?stream  (open  "a:\\example.dat"   "w"))

        (printf  ?stream  "%3S  %9S  %3d  %3s  %3s \n"
                        ?num
                        ?dscrp
                        ?dur
                        ?prec
                        ?succ
        )
        (close ?stream)
  )
```

Figure A.13 - Example Code for Printing Data to an Output file

Rule: OUTPUT-DATA

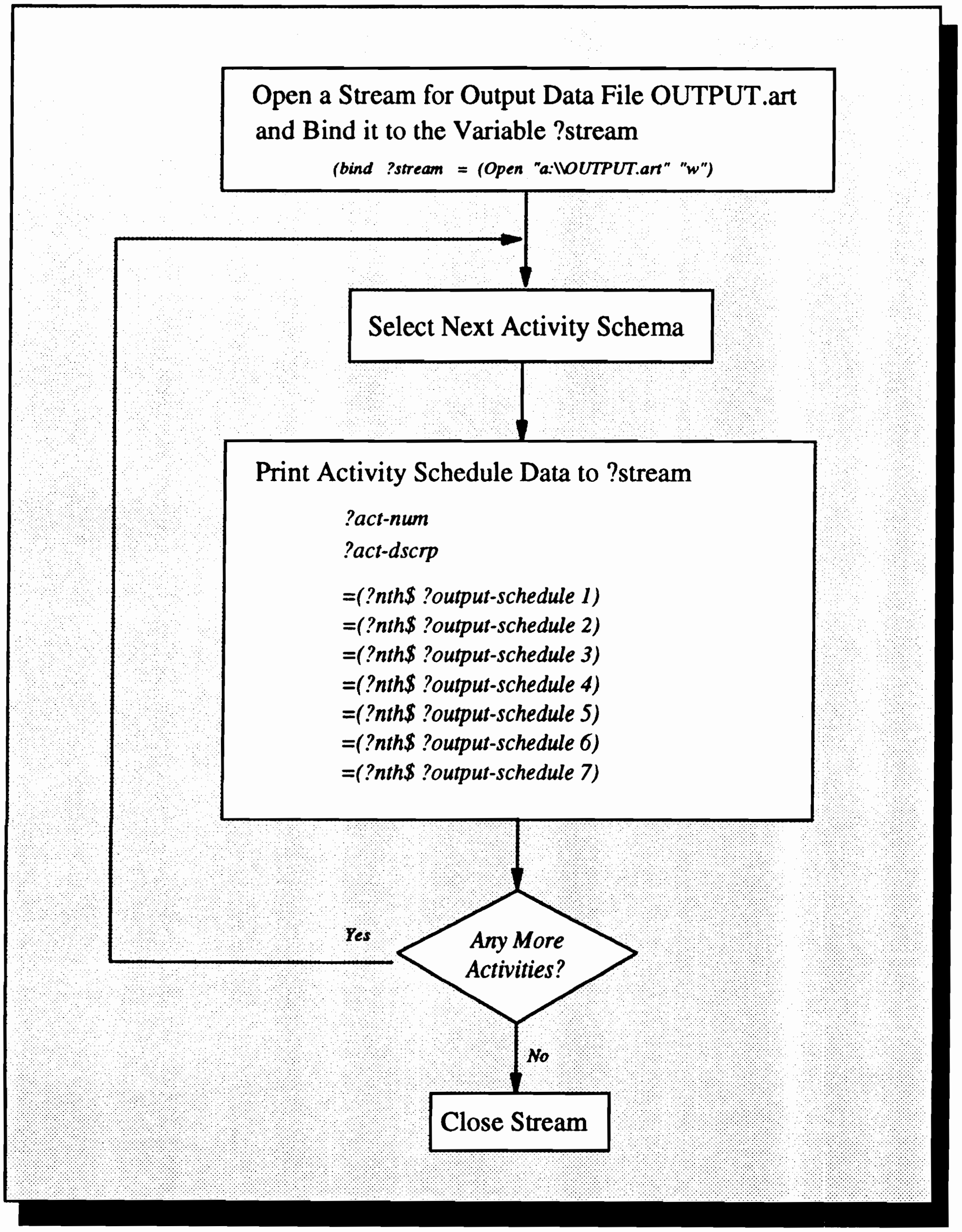

Open a Stream for Output Data File OUTPUT.art and Bind it to the Variable ?stream
(bind ?stream = (Open "a:\\OUTPUT.art" "w")
Select Next Activity Schema
Print Activity Schedule Data to ?stream
?act-num
?act-dscrp
=(?nth$?output-schedule 1)
=(?nth$?output-schedule 2)
=(?nth$?output-schedule 3)
=(?nth$?output-schedule 4)
=(?nth$?output-schedule 5)
=(?nth$?output-schedule 6)
=(?nth$?output-schedule 7)
Yes
Any More Activities?
No
Close Stream

Figure A.14 - Output Processor Rule: OUTPUT-DATA

A.2.2 Control of Execution of the Data Output Processor

The data output processor rule has the lowest priority of execution and will be the last rule to fire. The objective is to allow rules of other processors to extract all input data and generate a schedule before printing the results to the output file OUTPUT.art is started. This process is also controlled by the controller module.

As illustrated in Figure A.15, the execution of the output processor requires the ART-IM fact

fact: *(schedule-completed)*

to be present in the context. This fact is inserted by the controller module after all activities are scheduled. The inserted fact is responsible for initiating the execution of the output processor.

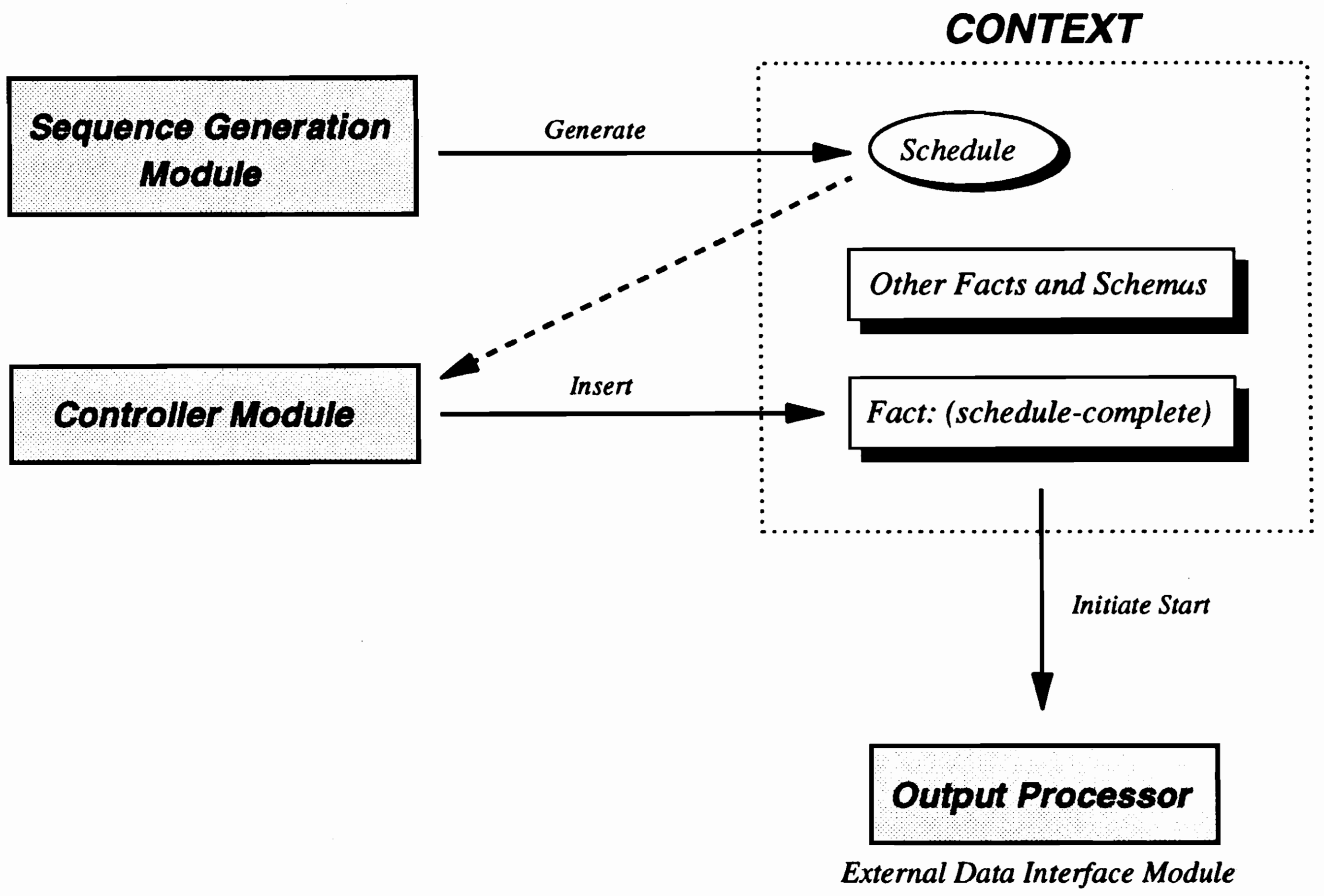


Figure A.15 - Control of Execution of the Data Output Processor

Appendix B

CONTROL OF THE SCHEDULING PROCESS:
The Controller Module

The second module of the knowledge-based scheduling system is the *Controller Module*. The controller module controls the overall scheduling process. The module comprises of several processors and has two basic functions:

1- Control the execution of the external data interface module.

2- Processing of the scheduling cycles.

In the first function, the controller module is responsible for the control of execution of the data input and data output processors of the external data interface module. The execution of these processors is dependent on specific ART-IM facts that should be present in the context. The controller module is responsible for inserting these facts to initiate the start of execution of these processors.

When the scheduling system is started, the controller module initiates the start of execution of the input processor to begin the extraction of input data from the external data files. Once data is extracted and the different schemata structures are created, the controller module begins the production of the schedule. At this point, control of execution is shifted to the sequence generation module to start verification of constraints for each activity. Once all activities are scheduled and the final schedule is generated, the controller module initiates the start of execution of the output processor to print the results of the generated schedule to the output file.

The second function of the controller module is the actual processing or execution of the scheduling cycles. During each cycle, the module executes the sequence generation module to verify the scheduling constraints and schedule the activities. The processing of the scheduling cycles is accomplished by performing a set of tasks:

1- Determine the list of activities that are eligible for scheduling in every cycle. The eligible activities define the eligible activity set (EAS).

2- Define the scheduling time for each scheduling cycle.

3- Prioritize the activities of the EAS during every scheduling cycle. The objective is to determine the order in which each activity competing for resources and work space is considered for scheduling. The prioritizing process is accomplished using the defined criteria discussed in chapter 8. During each scheduling cycle, the activity with the highest priority is placed in the Ordered Scheduled Set (OSS) to be considered for scheduling.

4- Execute the sequence generation module to verify the scheduling constraints for the activity placed in the OSS. Using an object oriented approach, instructions are sent to the different processors of the sequence generation module to verify the scheduling constraints and schedule each activity.

The functions of the controller module are performed by several processors that constitute the module. There are four processors:

1- An Execution Control Processor

2- An Activity Selection Control Processor

3- A Cycle Time Control Processor

4- A Scheduling Cycle Control Processor

The *Execution Control Processor* performs the first function of the controller module. The processor controls the execution of the data input and data output processors of the external data interface module.

The other three processors perform the second function of the controller module. The *Activity Selection Control Processor* determines the list of activities that are eligible for scheduling at the beginning of each scheduling cycle (EAS). An activity becomes eligible for scheduling when all its preceding activities are scheduled and its early start value is less than or equal to the scheduling cycle time. The processor verifies each activity and places those eligible for scheduling in the EAS.

The *Cycle Time Control Processor* determines the start time for each scheduling cycle. The processor computes the next possible time to start each cycle based on the list of activities that is considered for scheduling in that cycle.

The *Scheduling Cycle Control Processor* is the main processor of the controller module. This processor is responsible for processing each scheduling cycle based on the cycle time defined and eligible activities selected. During each cycle, the processor is responsible for prioritizing activities of the EAS. The processor uses heuristic criteria to define the order in which these activities are considered for scheduling.

The processor is also responsible for sending instructions to the sequence generation module to verify the different scheduling constraints for each prioritized activity placed in the ordered scheduled set (OSS). Instructions are sent to the sequence generation module in the form of messages using an object-oriented programming approach. Each message executes a method or procedure responsible for verifying a specific constraint. The sequence generation module consists of several procedures (i.e several processors). As each procedure is executed, a new message is sent to execute another procedure. The procedures are executed in a specific order. The process is continued until all procedures of the sequence generation module are executed and all constraints are verified. The following sections discusses the processors of the controller module in more detail.

B.1 The Execution Control Processor

The basic function of this processor is to control the execution of the data input and data output processors of the external data interface module. The data input processor should be executed prior to any other processor responsible for the production of the schedule. Similarly, the data output processor should only be executed after all input data is extracted and a schedule is generated. This order of execution is controlled by the execution control processor.

The control process is accomplished by manipulating special facts in the context. Each fact or group of facts define a condition to start or terminate the input and output processors. This is illustrated in Figure B.1. When the scheduling system is started, the execution control processor inserts the ART-IM fact

(start-data-extraction)

in the context. The inserted fact allows the data input processor to start execution and extract the input data from the external files. At this point, the status of the context is as shown in 1 of Figure B.1. Once all data is extracted and the schemata structures are created, additional facts are inserted in the context by the data input processor. As indicated previously in chapter 10, the purpose of inserting these facts is to indicate that

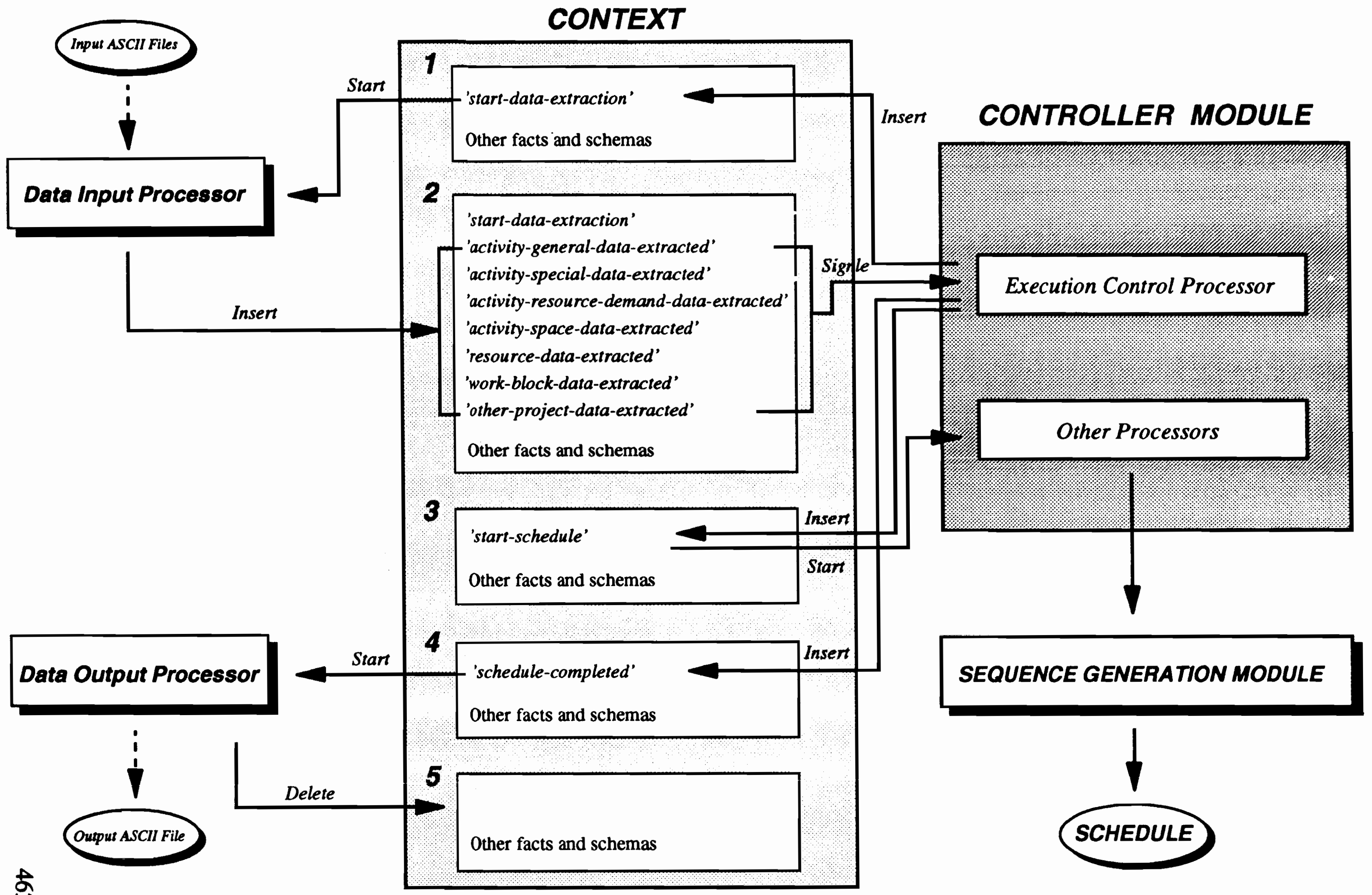


Figure B.1 - Control of Execution of the External Data Interface Module

the execution of the data input processor is completed. The status of the context becomes as shown in 2 of Figure B.1. Based on the status of the context, the execution control processor inserts the fact

(start-schedule)

in the context. The processor also deletes the other unnecessary facts to reduce the memory storage area. At this point, the status of the context is as shown in 3 of Figure B.1. Based on the new inserted fact, the execution of the data input processor is terminated and the schedule generation process is started.

Once a schedule is generated, the execution control processor inserts the ART-IM fact

(schedule-completed)

in the context. The inserted fact (start-schedule) is deleted to release memory area. At this point, the status of the context becomes as shown in 4 of Figure B.1. The new inserted fact initiates the start of the data output processor to print results of the generated schedule to the output ASCII file (OUTPUT.art). Once the output file is generated, the fact is deleted and the status of the context becomes as in 5 of Figure B.1.

The execution control processor consists of several rules that perform its control function. There are three rules: Control-Rule-1, Control-Rule-2, and Control-Rule-3.

The purpose of the *Control-Rule-1* rule is to initiate the execution of the data input processor of the external data interface module. This rule is the first rule to execute (or fire) once the scheduling system is started. The execution or firing of this rule results in inserting the ART-IM fact *(start-data-extraction)* in the context. The general format of the Control-Rule-1 is

Rule: **Control-Rule-1**

IF: *(no condition)*

THEN: *insert fact: (start-data-extraction)*

The *Control-Rule-2* is the second rule of the execution control processor. This rule is responsible for starting the schedule production process once all required input data is extracted. The execution or firing of this rule results in inserting the fact *(start-schedule)* in the context. The general format of the Control-Rule-2 is:

Rule: **Control-Rule-2**

IF: *(all data is extracted and stored in the context)*

THEN: *insert fact: (start-schedule)*
delete fact: (start-data-extraction)
delete fact: (activity-general-data-extracted)
delete fact: (activity-special-data-extracted)
delete fact: (activity-resource-demand-data-extracted)
delete fact: (activity-space-data-extracted)
delete fact: (resource-data-extracted)
delete fact: (work-block-data-extracted)
delete fact: (other-project-data-extracted)

The third rule of this processor is the *Control-Rule-3*. This rule is responsible for initiating the start of execution of the data output processor to print schedule results to an output file. The Control-Rule-3 inserts the ART-IM fact *(schedule-complete)* in the context to indicate that all activities are scheduled. The general format of the rule is

Rule: **Control-Rule-2**

IF: *(all activities are scheduled)*

THEN:*insert fact: (schedule-completed)*
delete fact: (start-schedule)

The source code listing of the execution control processor rules is provided in Appendix G of this document.

B.2 The Activity Selection Control Processor

The objective of this processor is to determine the list of activities eligible for scheduling at the beginning of each scheduling cycle. Any activity becomes eligible for scheduling when its predecessors are scheduled and its early start value is less or equal to the schedule cycle time. During any scheduling cycle, the eligible activities define the *Eligible Activity Set* (EAS).

The processor determines the eligible activity set in two steps. In the first step, the processor defines the *Initial Activity Set* (IAS). If the activity has no preceding activities or if all its preceding activities are scheduled, it is placed in the IAS. In the second step, the activity's early start (ES) value is verified with the schedule cycle start time (t_i). If the early start (ES) value is less than or equal to the cycle's start time (t_i), the activity is placed in the EAS.

The EAS is determined in two steps because the activities defining the IAS are first needed to determine the start cycle time (t_i). Once the cycle time is determined, the EAS can be defined. Figure B.2 depicts this 2-step procedure.

The activity selection control processor consists of two ART-IM rules; the *Define-IAS* rule, and the *Define-EAS* rule. The first rule determines the initial activity set (IAS). The second rule determines the eligible activity set (EAS).

The general format of the Define-IAS rule is:

Rule: **Define-IAS**

IF: *(no preceding activities)* or
(preceding activities scheduled)

THEN: *(put activity number in initial activity set)*

The general format of the Define-EAS rule is:

Rule: **Define-EAS**

IF: *(IAS defined)* and

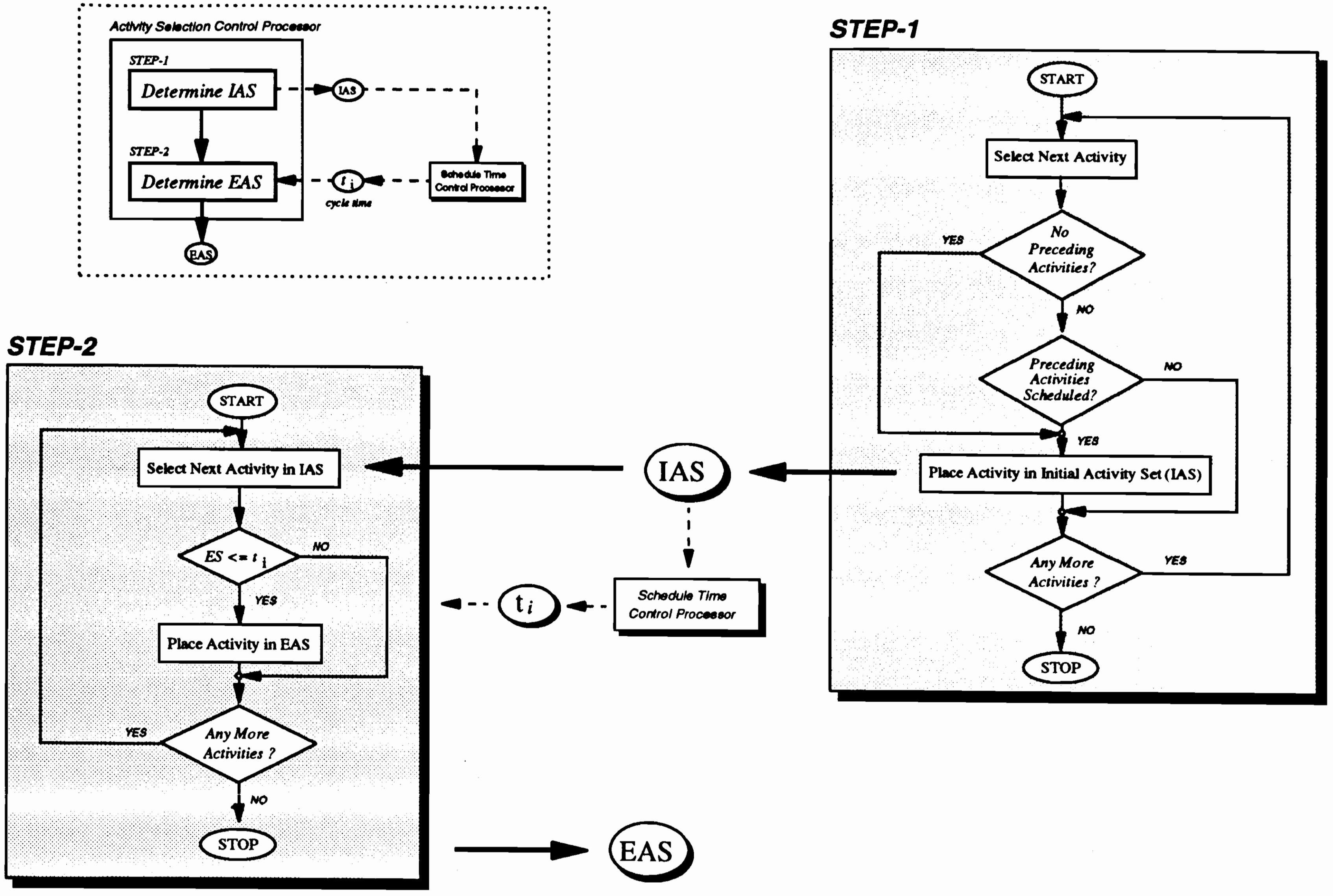


Figure B.2 - A 2-Step Procedure to Define the Eligible Activity Set (EAS)

```
                (schedule cycle time $t_i$ defined)
        THEN: for each activity in the (IAS)
                IF:     (ES) of activity <= $t_i$
                THEN: (place activity in EAS)
```

The source code listing of this activity selection control processor rules is provided in Appendix G of this document.

B.3 The Cycle Time Control Processor

The *Cycle Time Control Processor* is the third processor of the controller module. The purpose of this processor is to compute the schedule time for every scheduling cycle. The processor determines the next possible time to start each cycle based on the early start values of the activities defined in the initial activity set (IAS).

The early start value of each activity is monitored during the verification of the scheduling constraints. If any constraint is found unsatisfactory and the activity is delayed, the early start value of the activity is modified to the next possible time computed. For example, during resource constraints verification, if available resources are found unsatisfactory, the activity is dropped from the current scheduling cycle and is delayed. The next possible time to consider the activity for scheduling is computed based on when these required resources will be released by other activities using these resources. The activity early start value is then modified to the computed time.

Utilizing the early start value of activities of the initial activity set (IAS) to determine the cycle time reduces the total scheduling cycles required to schedule all activities of the project. This, in turn, results in considerable savings in memory usage and overall processing time. The flow chart of Figure B.3 depicts the process of computing the schedule time for any schedule cycle.

The cycle time control processor comprises of one rule, the *Define-Cycle-Time* rule. The general format of this rule is:

Rule: **Define-Cycle-Time**

IF: *Fact (old-time = ?t) of previous cycle*
THEN: *1- Determine minimum early start (?min-es) value of all activities defined in the IAS.*

2- Assert fact (new-time = ?min-es)

3- Remove fact (old-time = ?t) from the context

The rule first determines the minimum early start value for activities of the IAS defined for each scheduling cycle. Based on the minimum early start value (?min-es), the schedule time for the scheduling cycle is determined. A new fact (new-time ?min-es) is inserted in the context and the old fact (old-time ?t) is removed.

The source code listing of this rule is provided in Appendix G of this document.

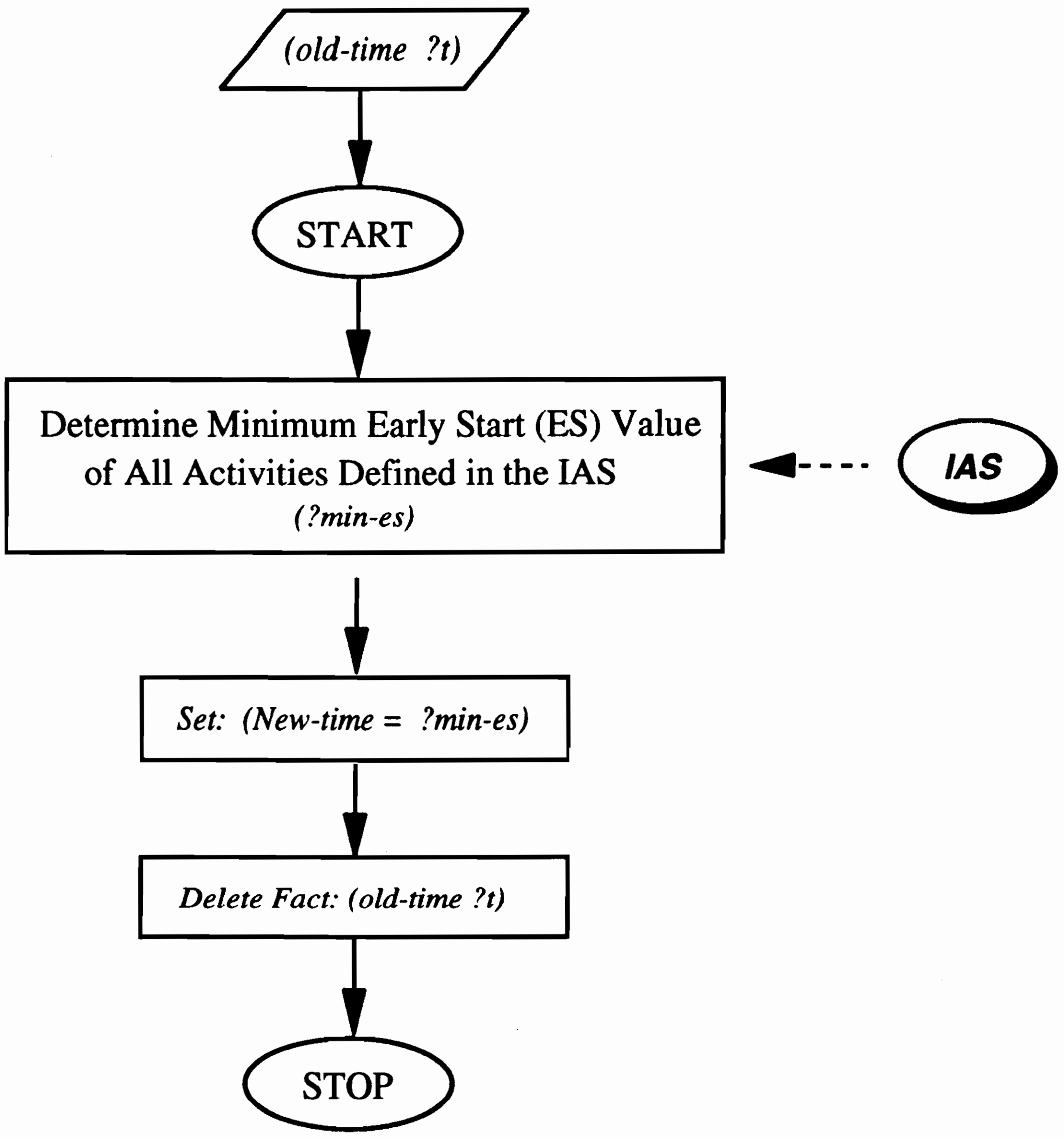

(old-time ?t)
START
Determine Minimum Early Start (ES) Value of All Activities Defined in the IAS
(?min-es)
IAS
Set: (New-time = ?min-es)
Delete Fact: (old-time ?t)
STOP

Figure B.3 - FlowChart for ART-IM Rule: Define-Cycle-Time

B.4 The Scheduling Cycle Control Processor

The *Scheduling Cycle Control Processor* is the main processor of the controller module and is responsible for the actual processing or execution of the scheduling cycles. Figure B.4 depicts a general flow chart of a scheduling cycle as executed by the processor.

With reference to Figure B.4, once an eligible activity set (EAS) is defined, and a cycle start time (t_i) is selected, the processor starts a scheduling cycle to attempt to schedule all activities of the EAS defined.

The processor first prioritizes the activities defined in the EAS and determines the next activity in order (i.e with the highest priority) to be considered for scheduling. Once an activity is selected, it is placed in the ordered scheduled set (OSS). The scheduling cycle control processor then executes the different processors of the sequence generation module to verify the scheduling constraints for the activity. Based on the constraints verification process, the activity is either scheduled or delayed. If the activity is scheduled, it is dropped from the OSS, EAS, and the IAS. However, if the activity can not be scheduled and is delayed to another time period, it is dropped from the OSS and EAS but remains in the IAS.

From the remaining list of activities in the eligible activity set (EAS), the next activity in order is selected and placed in the ordered scheduled set (OSS) and is checked for the different scheduling constraints. This process is continued for all activities defined in the

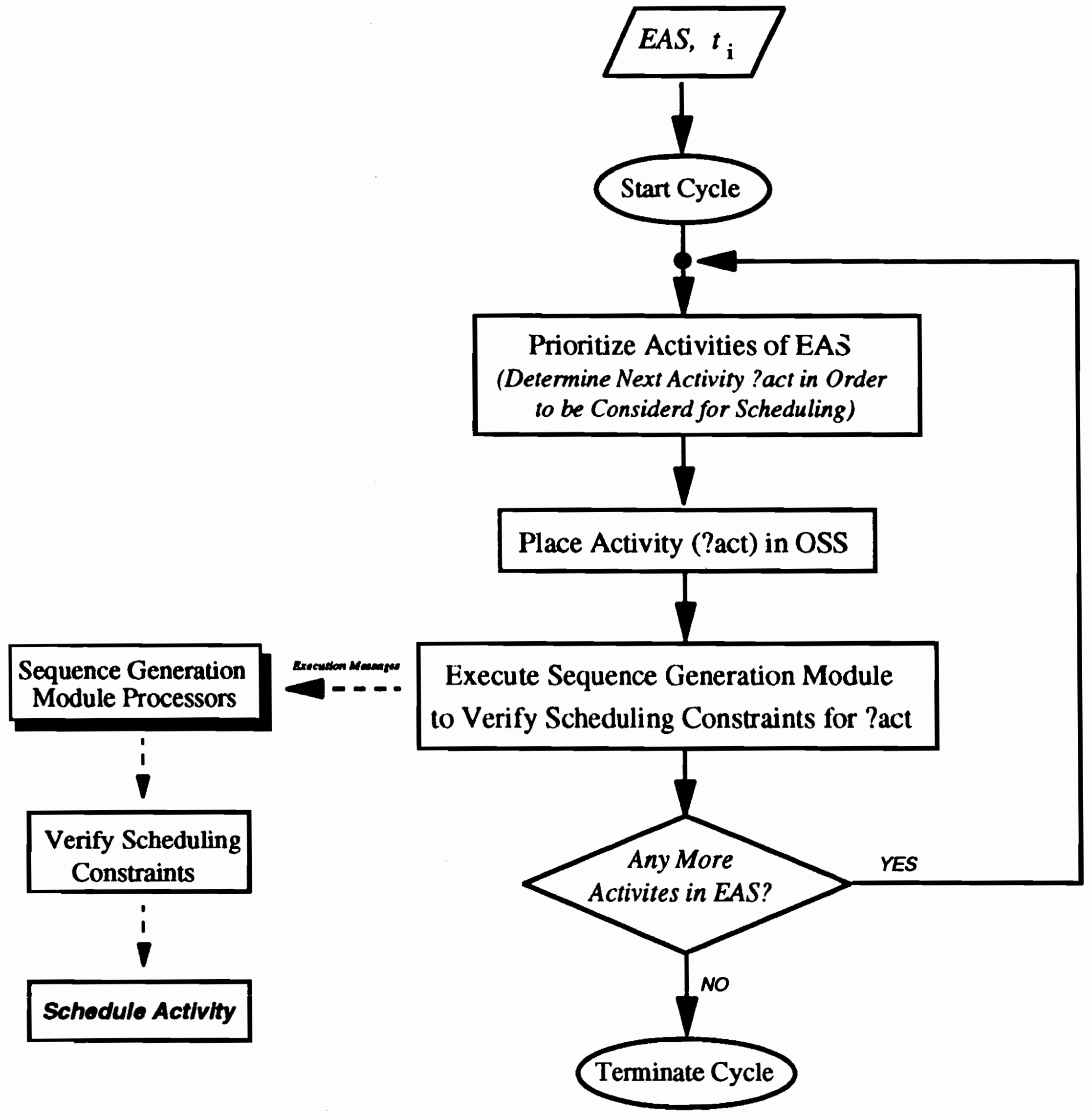

EAS, t_i
Start Cycle
Prioritize Activities of EAS
(Determine Next Activity ?act in Order to be Considerd for Scheduling)
Place Activity (?act) in OSS
Execute Sequence Generation Module to Verify Scheduling Constraints for ?act
Execution Messages
Sequence Generation Module Processors
Verify Scheduling Constraints
Schedule Activity
Any More Activites in EAS?
YES
NO
Terminate Cycle

Figure B.4 - The Overall Scheduling Cycle

EAS. The cycle is terminated when all activities defined in the EAS are considered for scheduling.

B.4.1 Prioritizing Activities of the EAS

The *Scheduling Cycle Control Processor* is responsible for prioritizing activities of the (EAS) during each scheduling cycle. The processor determines the order in which each activity of the EAS will be considered for scheduling. This order is based on the priority of each activity determined by a specified heuristic criteria. Activities with a higher priority will be considered for scheduling first.

From all the activities placed in the EAS, the processor determines the first activity with the highest priority. The activity is placed in the ordered scheduled set (OSS). Once the activity is considered for scheduling, it is dropped from the OSS and EAS list. From the remaining activities in the EAS, the processor determines the next activity with the highest priority to be considered for scheduling. This process is repeated for all the activities of the EAS.

As illustrated in Figure B.5, the prioritizing process of activities uses three basic rules or ranking criteria. These criteria and the order in which they are applied are as follows:

1- Activity late start value. This is the activity late start value determined from an initial schedule for a typical floor.

Criterion 1: Activities with the minimum late start value will be considered for

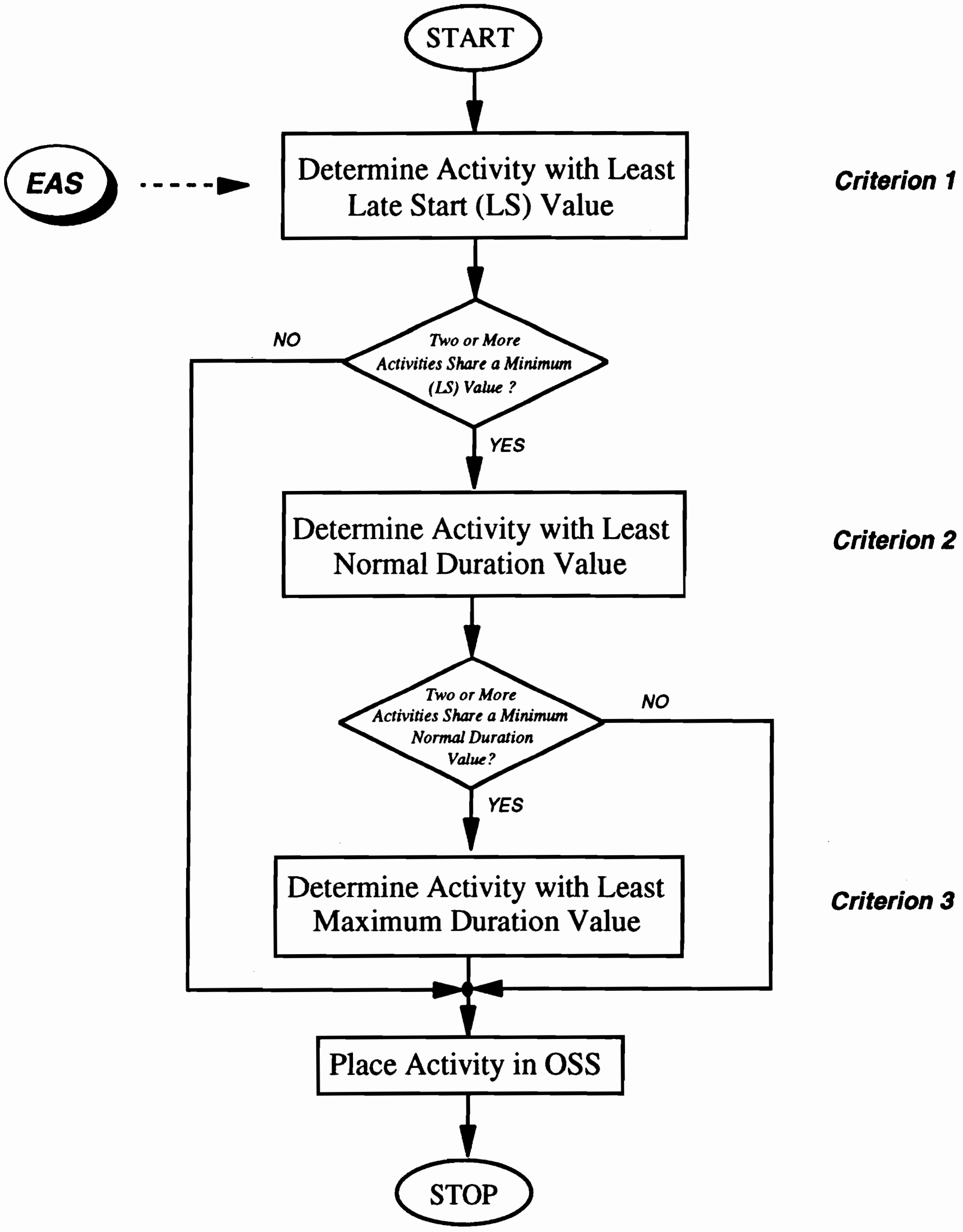

START
EAS
Determine Activity with Least Late Start (LS) Value
Criterion 1
Two or More Activities Share a Minimum (LS) Value ?
NO
YES
Determine Activity with Least Normal Duration Value
Criterion 2
Two or More Activities Share a Minimum Normal Duration Value?
NO
YES
Determine Activity with Least Maximum Duration Value
Criterion 3
Place Activity in OSS
STOP

Figure B.5 - Prioritizing Process Using Defined Criteria

scheduling first.

2- Normal activity duration. This is the activity normal duration defined by the user based on normal production rate.

Criterion 2: *Activities with the same minimum late start value will be ranked according to their normal duration values. The activity with the least normal duration value will be considered for scheduling first.*

3- Maximum activity duration. This is the activity maximum duration also defined by the user based on the minimum possible production rate to execute the activity.

Criterion 3: *Activities with the same minimum late start value and minimum normal duration value will be ordered according to their maximum duration value. The activity with the least maximum duration value will be considered for scheduling first.*

B.4.2 Execution of the Sequence Generation Module

Once an activity is selected for scheduling and placed in the ordered scheduled set (OSS), the scheduling cycle processor executes the different processors of the sequence generation module to verify the different constraints for the activity. The execution of the sequence generation module's processors is accomplished using an object-oriented approach. As explained earlier in chapter 8, the sequence generation module consists of eight processors defined as ART-IM functions in the knowledge-base. There are eight primary functions defining the processors of the sequence generation module. These functions are:

1- *Verify-horizontal-logic-constraints-A* and *Verify-horizontal-logic -constraints-B*
(Horizontal logic processor)

2- *Verify-vertical-logic-constraints*
(Vertical logic processor)

3- *Verify-resource-constraints*
(Resource allocation processor)

4- *Verify-work-space-constraints*
(Space allocation processor)

5- *Modify-resource-demand-pool*
(Resource modification processor)

6- *Schedule-A* and *Schedule-B*
(Schedule processor)

7- *Get-total-dur*
(Duration processor)

Except for the resource modification and the duration processors, the execution of the other processor are controlled by the controller module. Each activity schema contains a *'scheduling-method'* slot. The attribute value of the slot will define the name of any one of these eight functions. As an activity is selected for scheduling, the scheduling cycle control processor sends a message to the 'scheduling-method' slot of the activity schema. The function defined by the attribute value of the slot is activated and executed. If the constraint is found satisfactory, the function modifies the attribute value of the 'scheduling-method' slot of the activity to define the name of the next function in order. A new message is sent to the slot by the scheduling cycle control processor and the new function is activated and executed. This process is continued until all functions are executed and the activity is scheduled, or until the activity is delayed and dropped from the scheduling cycle. The order in which the functions are executed are shown in Figure B.6. With reference to Figure B.6, the first function to be executed is that of the horizontal logic processor. The attribute value of the 'scheduling-method' slot defines the name of that function,

For continuity class A activities

```
(defschema activity-i
       .
       .
   (scheduling-method      verify-horizontal-logic-constraints-A)
       .
       .
)
```

or, for continuity class B activities

```
(defschema activity-i
       .
       .
   (scheduling-method      verify-horizontal-logic-constraints-B)
       .
       .
)
```

The function name of the horizontal logic processor is set as the initial attribute value for the 'scheduling-method' slot for each activity. This is defined by the scheduling cycle control processor before the start of all the scheduling cycles. As a message is sent to the 'scheduling-method' slot, the horizontal logic processor is activated and the horizontal logic is verified for the activity. If the constraints are satisfactory, the function of the horizontal logic processor modifies the attribute of the 'scheduling-method' slot to the

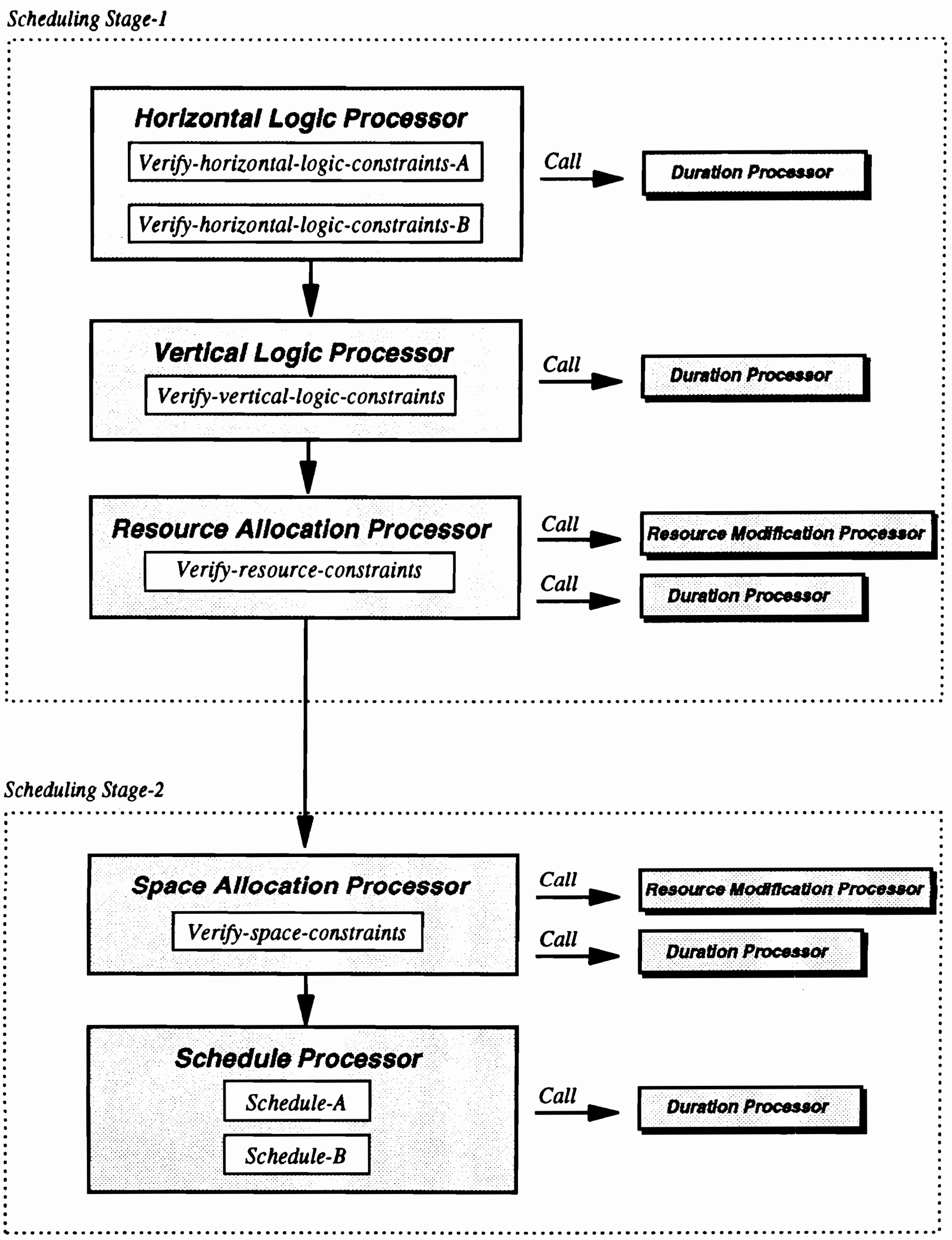

Scheduling Stage-1
Horizontal Logic Processor
Verify-horizontal-logic-constraints-A
Verify-horizontal-logic-constraints-B
Call
Duration Processor
Vertical Logic Processor
Verify-vertical-logic-constraints
Call
Duration Processor
Resource Allocation Processor
Verify-resource-constraints
Call
Resource Modification Processor
Call
Duration Processor
Scheduling Stage-2
Space Allocation Processor
Verify-space-constraints
Call
Resource Modification Processor
Call
Duration Processor
Schedule Processor
Schedule-A
Schedule-B
Call
Duration Processor

Figure B.6 - Execution Order of Processors of the Sequence Generation Module

function name of the vertical logic processor,

```
(defschema activity-i
        .
        .
   (scheduling-method      verify-vertical-logic-constraints)
        .
        .
)
```

A new message is sent by the scheduling cycle control processor to activate the new function defined by the attribute value of the slot. As a result, the vertical logic processor is activated to verify the vertical logic constraints for the activity. Similarly, if the processor is successfully executed and the constraints are found satisfactory, the attribute value of the 'scheduling-method' slot is modified to define the next function in order (i.e function: verify-resource-constraints). A new message is sent to activate the function and execute the resource allocation processor.

This process of sending messages and modifying the slot attribute is continued until all processors are executed and the activity is scheduled. As the last function (i.e. schedule-A or schedule-B) is executed and the activity is scheduled, the attribute value of the slot is modified,

```
(defschema activity-i
        .
        .
   (scheduling-method      activity-scheduled)
        .
        .
)
```

At this point, the activity is dropped from the OSS, EAS, and the IAS. The next activity in order is determined and placed in the OSS and the entire process is repeated. This object-oriented approach is illustrated in Figure B.7.

B.4.3 Execution Messages

Messages are sent to each activity by the scheduling cycle control processor using the ART-IM function *Send*. The general format of this function is

(send slot-name activity-schema-name arguments)

The slot name defines the name of the slot in the activity schema to which a message is to be sent. In this case, the slot name is "scheduling-method". The activity-schema-name defines the name of the activity schema to which the message is sent. In addition to the slot-name and activity-schema-name, the function will take three other arguments:

1- Start time (t_i).
2- From-floor number
3- To-floor number

The start time (t_i) defines the time at which the activity will be scheduled to start. This time will be equal to the time of the scheduling cycle executed. The from-floor number defines the start floor number from which the activity will be considered for scheduling. The to-floor number defines the last floor number to which the activity will be considered for scheduling.

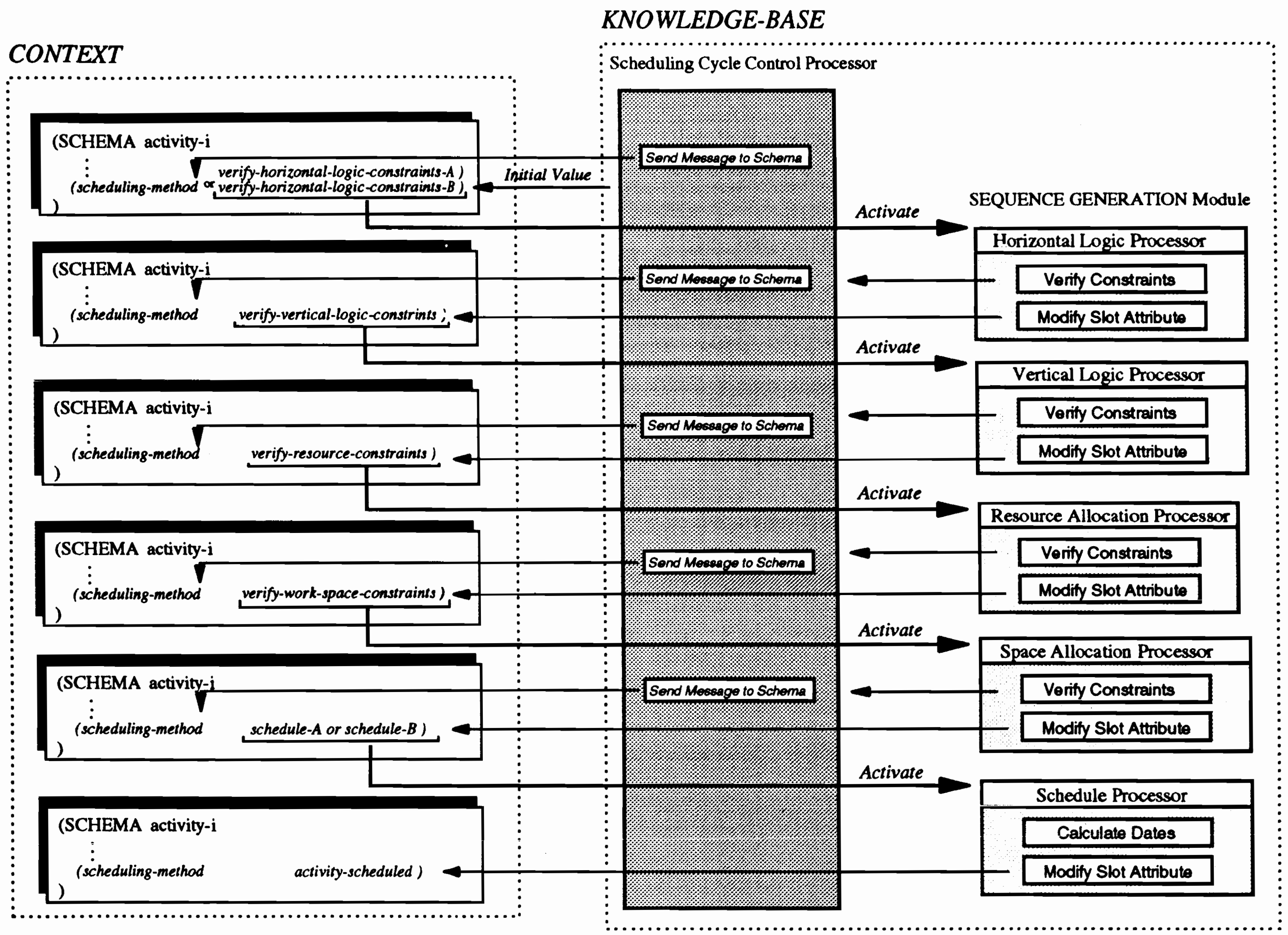

KNOWLEDGE-BASE
CONTEXT
Scheduling Cycle Control Processor
(SCHEMA activity-i
(scheduling-method
verify-horizontal-logic-constraints-A)
or verify-horizontal-logic-constraints-B)
Initial Value
Send Message to Schema
Activate
SEQUENCE GENERATION Module
Horizontal Logic Processor
Verify Constraints
Modify Slot Attribute
verify-vertical-logic-constrints)
Vertical Logic Processor
verify-resource-constraints)
Resource Allocation Processor
verify-work-space-constraints)
Space Allocation Processor
schedule-A or schedule-B)
Schedule Processor
Calculate Dates
activity-scheduled)

Figure B.7 - Activating the Sequence Generation Module Processors Using an Object-Oriented Programming Approach

An example execution message sent to an activity A0010 to be considered for scheduling along floors 1 to 20 at time $t_i = 5$ would be:

(send scheduling-method A0010 5 1 20)

The from-floor and to-floor numbers of each activity considered for scheduling will be based on the continuity class parameter of the activity. For activities with continuity class A, splitting is not allowed and the activity must be scheduled continuously as one segment along all typical floors. The from-floor and to-floor numbers in the execution message will correspond to the first and last typical floors of the project.

For continuity class B activities, each activity will be divided into several segments along the typical floors. The activity will be scheduled one segment at a time. Segments will be scheduled sequentially in order of floors. The from-floor and to-floor numbers in the execution message will correspond to the first and last floor of each segment.

As each message is sent, the activity segment will be checked for the scheduling constraints. If the segment is scheduled, a new from-floor and to-floor numbers will be computed for the next segment and a new message is sent to schedule the activity segment along the computed floors. This process is continued for all activity segments.

The number of floors for each activity segment will be computed by the scheduling cycle control processor. The total floors for each segment will be computed based on the maximum number of splits parameter of the activity. The total number of activity segments will be equal to the maximum number of splits.

Total floor for each segment (except last segment) will be equal and computed as

$$\text{Total floors of segment (except last segment)} = \frac{\text{Total number of typical floors}}{\text{Max number of splits}}$$

for the last activity segment, the total number of floors will be computed as

$$\text{Total floors (last segment)} = \text{Total number of typical floors} - \sum \text{floors of previous segments}$$

The computed values for each activity, with continuity class B, will be stored in the specific activity slots,

```
(defschema activity-i
        •
        •
   (continuity-class B)
        •
        •
   (floors-per-segment  ?)
   (floors-last-segment  ?)
        •
        •
)
```

This will be explained by the following example. Consider the following data for an activity A0010:

Activity schema number = A0010
Continuity class = B
Workflow direction = up
maximum splits = 3

Assuming that the total number of typical floors is 20, then

Total floors of segment = 20/3 = 6 floors
(except last segment)

Total floors for last segment = 20 - (2 x 6) = 8 floors

Based on these calculations, the activity will be considered for scheduling in three segments. The first two segments will each be scheduled along 6 typical floors, while the last segment will be scheduled along 8 floors.

The execution messages sent to schedule each segment will be as follows:

for first segment 1,
```
(send  scheduling-method  A0010  5  1  6)
```

for first segment 2,
```
(send  scheduling-method  A0010  5  7  12)
```

for first segment 3,
```
(send  scheduling-method  A0010  5  13  20)
```

If the three activity segments satisfies all the scheduling constraints verified during the same scheduling cycle, the segments will be scheduled continuously one after the other. In this case, there will be no delay lag between the segments and the activity will appear to be scheduled continuously along the typical floors. This is illustrated by case-1 of

Figure B.8. On the other hand, if one or more segments do not satisfy any scheduling constraint and is delayed, this will result in a lag between the segments. The activity will be split along the typical floors. This is illustrated by cases 2 , 3, and 4 of Figure B.8.

B.4.4 Execution Rules

The scheduling cycle control processor consists of two ART-IM execution rules: *Initialize-Activities* rule, and *Process-Scheduling-Cycles rule.*

The first rule, *Initialize-Activities,* is responsible for defining the horizontal logic processor function name as the initial attribute value of the 'scheduling-method' slot of each activity. During the scheduling cycles, as each activity is selected and an execution message is sent, the horizontal logic processor will be the first to be executed.

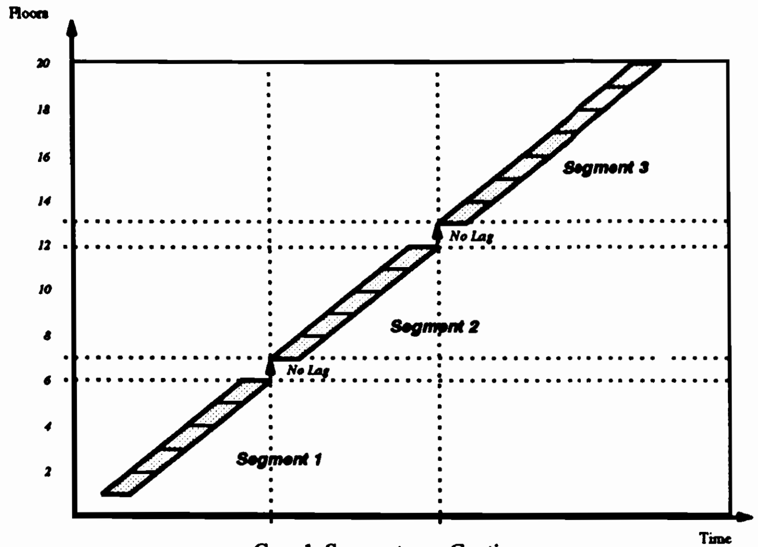
The rule is also responsible for computing the number of floors for each segment of continuity class B activities. Figure B.9 depicts a flow chart of the rule. The general format of the rule is:

Rule: **Initialize-Activities**

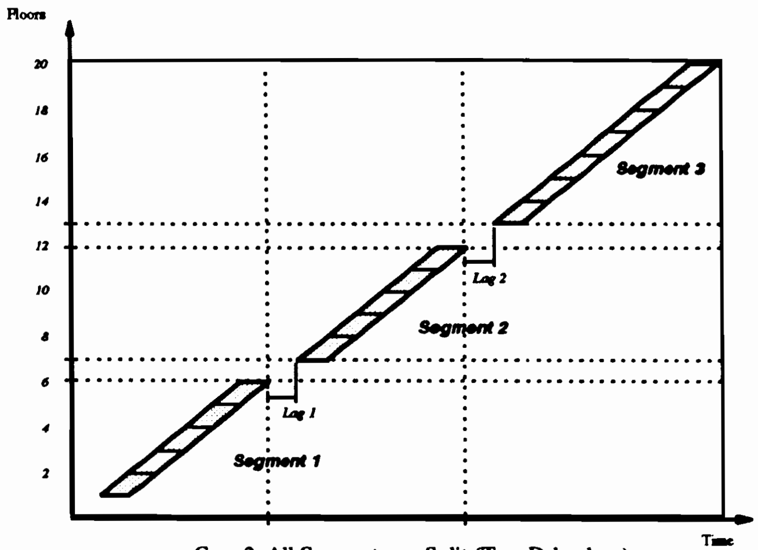
IF: *(all input data is extracted)*

THEN: For each activity schema defined in the context, Do

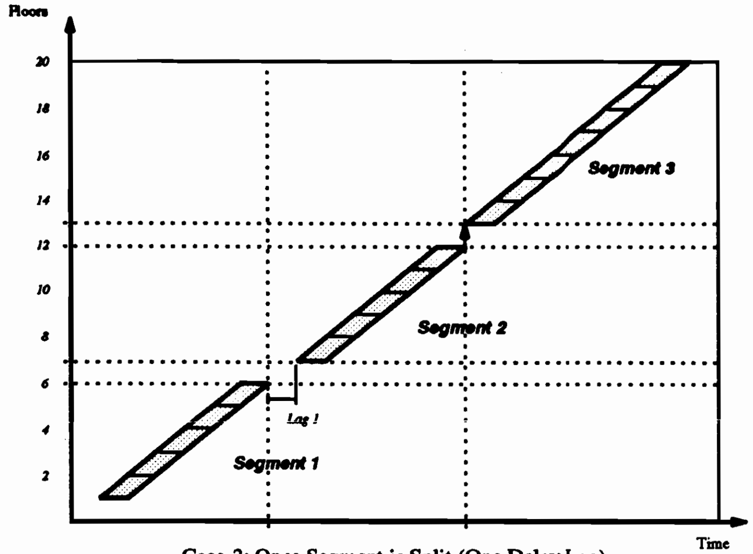
1- Set attribute value of act-dur slot equal to the normal activity duration value defined by user.



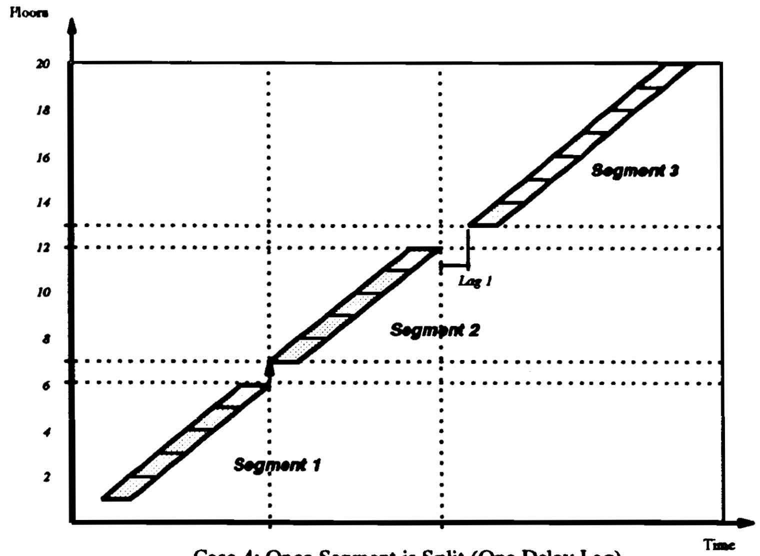
Case-1: Segments are Continous



Case-2: All Segments are Split (Two Delay lags)



Case-3: Once Segment is Split (One Delay Lag)



Case-4: Once Segment is Split (One Delay Lag)

Figure B.8 - Scheduling of Activity Segments

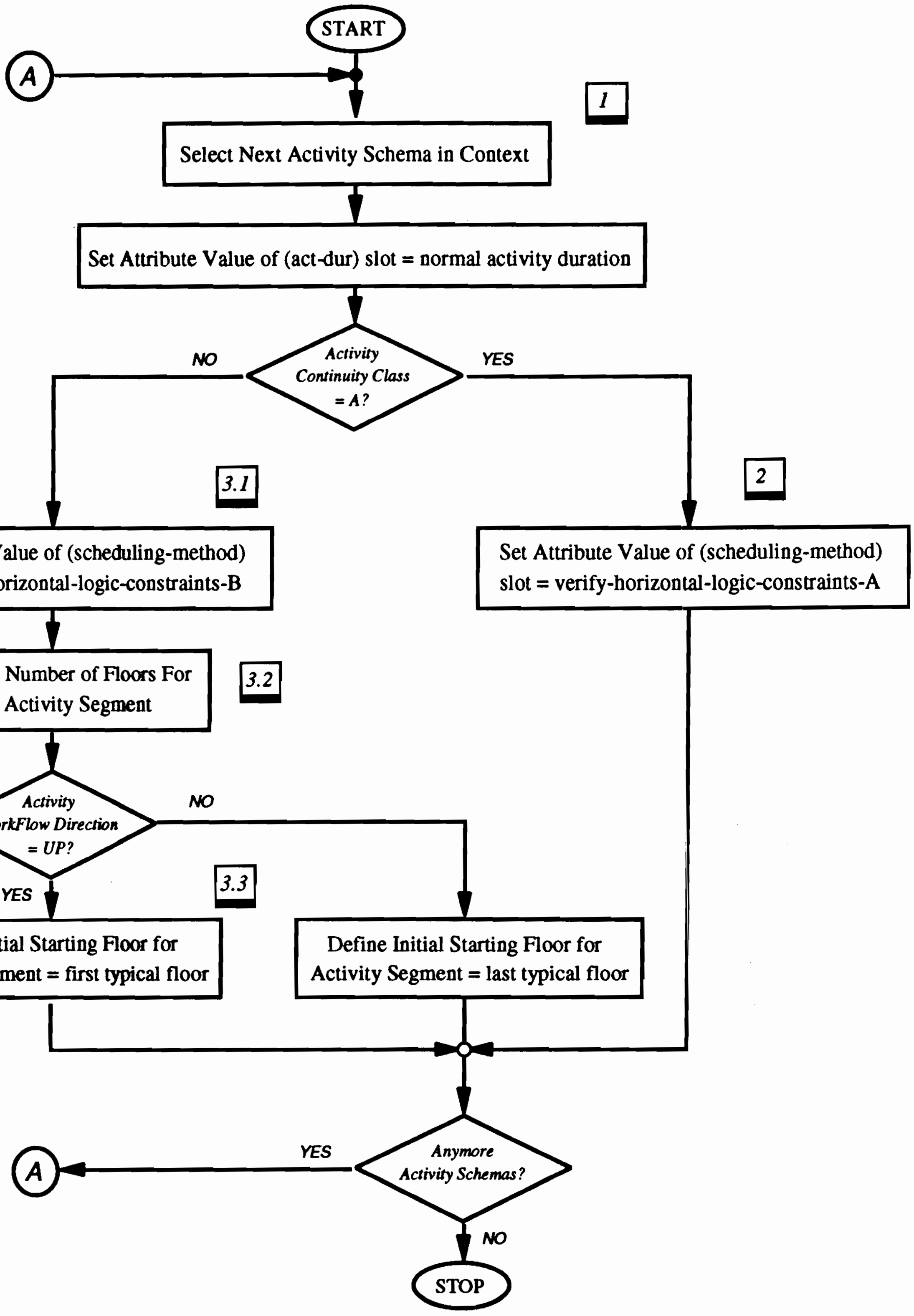
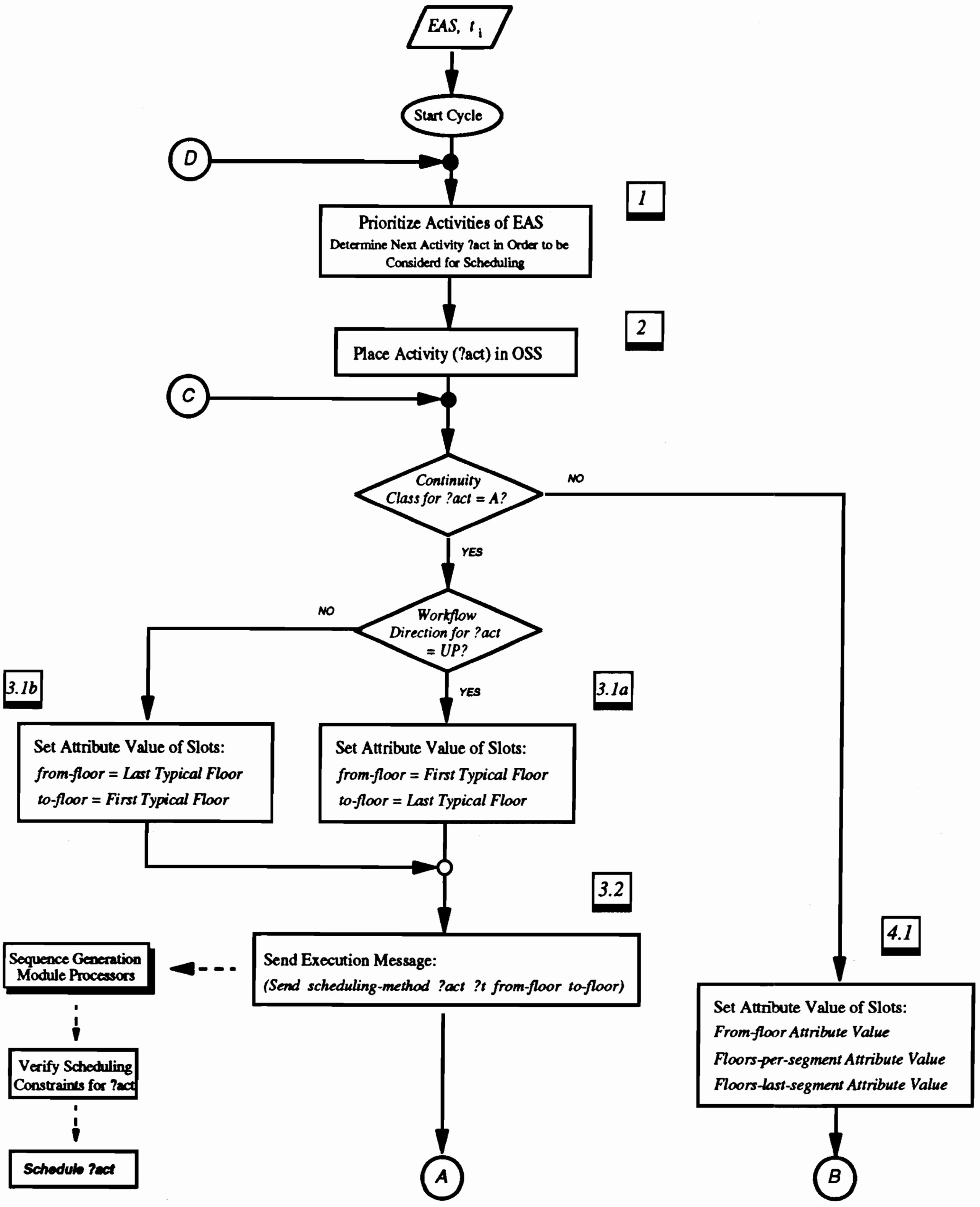

START
A
1
Select Next Activity Schema in Context
Set Attribute Value of (act-dur) slot = normal activity duration
NO
Activity Continuity Class = A?
YES
3.1
2
Set Attribute Value of (scheduling-method) slot = verify-horizontal-logic-constraints-B
Set Attribute Value of (scheduling-method) slot = verify-horizontal-logic-constraints-A
Compute Number of Floors For Each Activity Segment
3.2
Activity WorkFlow Direction = UP?
NO
YES
3.3
Define Initial Starting Floor for Activity Segment = first typical floor
Define Initial Starting Floor for Activity Segment = last typical floor
A
YES
Anymore Activity Schemas?
NO
STOP

Figure B.9 - Flowchart of ART-IM Rule: Initialize-Activities

```
2- IF: (activity continuity class = A)

   THEN: set attribute value of the 'scheduling-method' slot
         to (verify-horizontal-logic-constraints-A)

3- IF: (activity continuity class = B)

   THEN:
        3.1 Set attribute value of the 'scheduling-method'
            slot to (verify-horizontal-logic-constraints-B)

        3.2 Compute number of floors for each activity
            segment.

        3.3 Define starting floor number of first segment of
            activity based on workflow direction parameter.
```

The second rule, *Process-Scheduling-Cycles*, is responsible for the execution of each cycle and sending execution messages to activate the sequence generation module's processors. Figure B.10 depicts a flow chart of the rule. The general format of the rule is:

```
Rule: Process-Scheduling-Cycles

IF: (scheduling cycle time is defined) and
    (eligible activity set is defined)

THEN: For each activity schema defined in the EAS, Do

      1- Prioritize activities of the EAS and determine the next activity
      (?act) in order to be considered for scheduling.

      For all activities of the EAS,

      (IF: (any activity has the earliest late start value)
      THEN: (select activity to be considered for scheduling first)
      )


      ( IF: (two or more activities have the earliest late start value)
```



EAS, t_i
Start Cycle
D
1
Prioritize Activities of EAS
Determine Next Activity ?act in Order to be Considerd for Scheduling
2
Place Activity (?act) in OSS
C
Continuity Class for ?act = A?
NO
YES
Workflow Direction for ?act = UP?
NO
YES
3.1b
Set Attribute Value of Slots:
from-floor = Last Typical Floor
to-floor = First Typical Floor
3.1a
Set Attribute Value of Slots:
from-floor = First Typical Floor
to-floor = Last Typical Floor
3.2
Send Execution Message:
(Send scheduling-method ?act ?t from-floor to-floor)
Sequence Generation Module Processors
Verify Scheduling Constraints for ?act
Schedule ?act
A
4.1
Set Attribute Value of Slots:
From-floor Attribute Value
Floors-per-segment Attribute Value
Floors-last-segment Attribute Value
B

Figure B.10 - Flowchart of ART-IM Rule: Process-Scheduling-Cycles

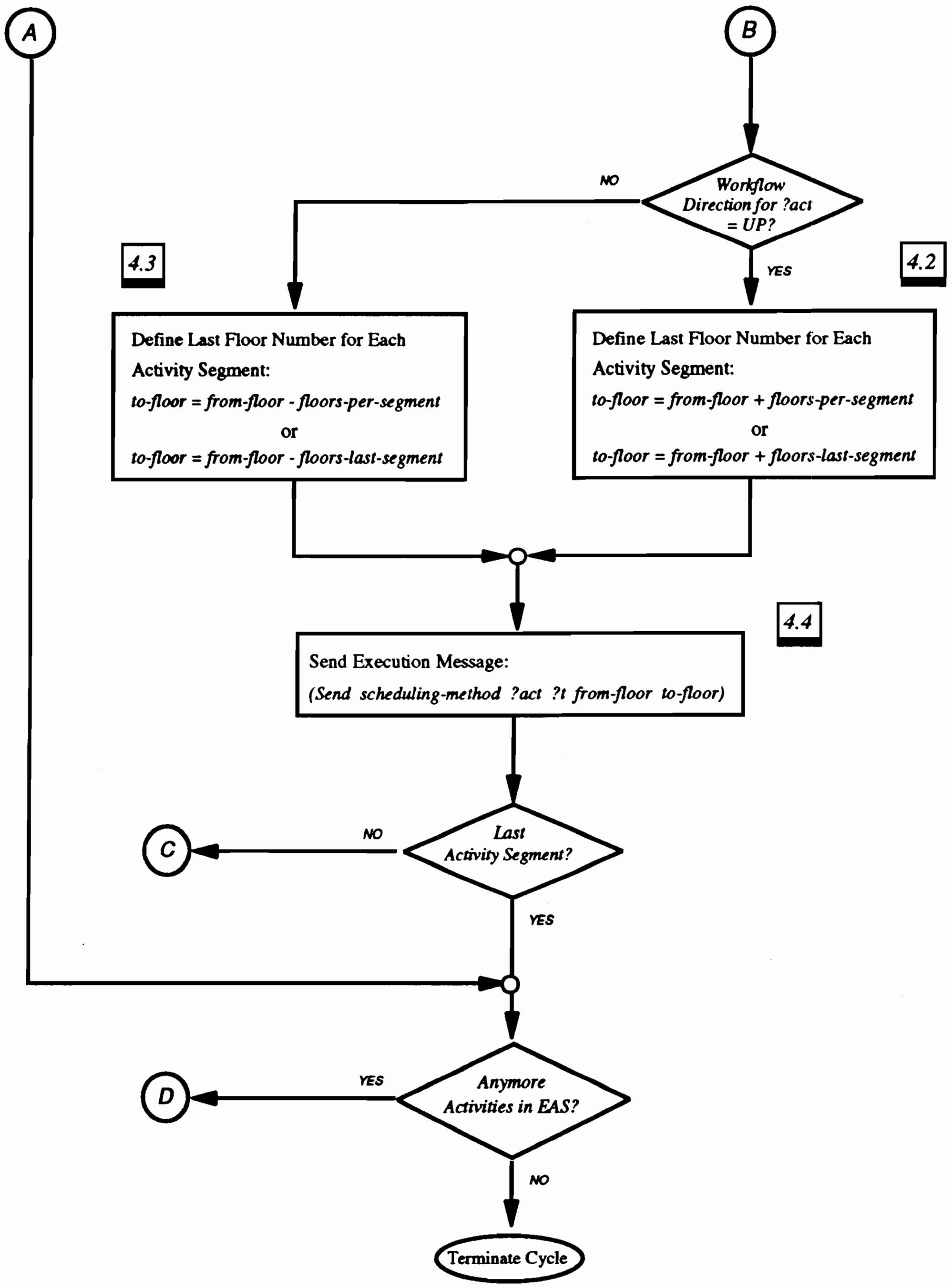

A
B
Workflow Direction for ?act = UP?
NO
YES
4.3
4.2
Define Last Floor Number for Each Activity Segment:
to-floor = from-floor - floors-per-segment
or
to-floor = from-floor - floors-last-segment
Define Last Floor Number for Each Activity Segment:
to-floor = from-floor + floors-per-segment
or
to-floor = from-floor + floors-last-segment
4.4
Send Execution Message:
(Send scheduling-method ?act ?t from-floor to-floor)
Last Activity Segment?
NO
C
YES
Anymore Activities in EAS?
YES
D
NO
Terminate Cycle

Figure B.10 - (Continued)

```
THEN:
        IF: (any activity has the least normal duration)
        THEN: (select activity to be considered for scheduling
        first)
)


( IF: (two or more activities have the earliest late start value and
      minimum normal duration value)
THEN:
        IF: (any activity has the least maximum duration)
        THEN: (select activity to be considered for scheduling
        first)
)

2- Place prioritized activity in the ordered scheduled set (OSS).

3- IF: (?act continuity class = A)
   THEN:
        3.1 IF: (?act workflow direction = up)
            THEN: 3.1a set: from-floor = first typical floor
                       set: to-floor = last typical floor

            ELSE: 3.1b set: from-floor = last typical floor
                       set: to-floor = first typical floor

        3.2 send execution message to schedule activity between
        from floor      and to-floor.


4- IF: (?act continuity class = B)
   THEN: 4.1 Identify computed from-floor number, number of
             floors per activity segment, and number of floors for
             last segment.

         4.2 IF: (?act workflow direction = up)
             THEN: for any activity segment,
                        set: to-floor = from-floor + floors per segment
                   for last activity segment,
                        set: to-floor = from-floor + floors last segment


         4.3 IF: (?act workflow direction = down)
             THEN: For any activity segment,set
```

to-floor = from-floor - floors per segment
For last activity segment, set
to-floor = from-floor - floors last segment

4.4 *Send execution message to schedule activity segment between from-floor and to-floor.*

The source code listing for the scheduling cycle control processor rules is provided in Appendix G of this document.

Appendix C

PRODUCTION of THE SCHEDULE:
The Sequence Generation Module

The *Sequence Generation Module* is the center piece of the overall knowledge-based scheduling system. During each scheduling cycle, the module is responsible for verifying the scheduling constraints for each activity segment and computing its schedule start and finish dates. The execution of the module is controlled by the controller module.

The verification of constraints for each activity is processed in two scheduling stages. In each stage, specific constraints are verified and scheduling decisions are made to schedule or delay the activity. In the first stage, horizontal and vertical constraints are first verified. Activity normal duration may be modified to satisfy these constraints. Consequently, initial activity resource demand pool is modified to account for the new selected duration. Preliminary dates are then computed and the activity is checked for resource constraints. Activity resource demand values, defined by the user or modified by the system, are compared to availability values during the preliminary dates. If resource constraints are satisfactory, the scheduling process moves to the second stage.

In the second scheduling stage space constraints are verified. Activity space demand values are first determined. Space demand for manpower/equipment (SD-1) is computed based on the final resource pool determined for the activity and the unit space demand values defined by the user. Space demand for material (SD-2) is determined based on total values defined by the user.

Demand values (SD-1, SD-2) are then compared to availability in corresponding work blocks. If available space is satisfactory, the activity is scheduled. Otherwise, space scheduling decisions are made to account for lack of space. Based on these decisions, either specific parameters are modified and the activity is scheduled, or the activity is delayed and dropped from the scheduling cycle. Activity duration may be modified for the second time and the resource demand pool is adjusted.

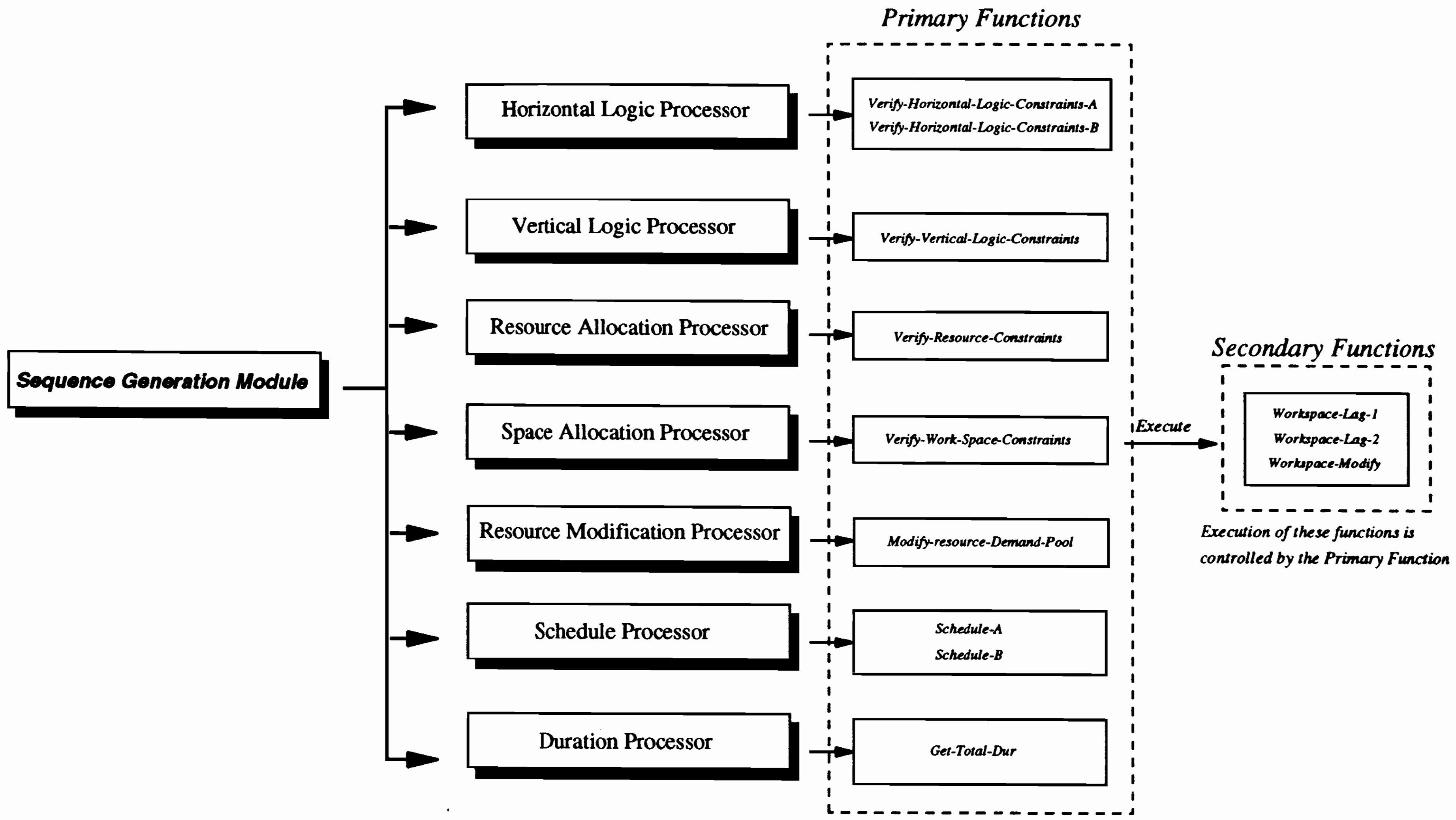
During each scheduling cycle, if any constraint is unsatisfactory and the activity is delayed, the module determines the next possible time the activity should be re-considered for scheduling.

The sequence generation module consists of several processors responsible for verifying constraints and scheduling of each activity. There are seven processors that constitute the sequence generation module:

1- The Horizontal Logic processor
2- The Vertical Logic Processor
3- The Resource Allocation Processor
4- The Space Allocation Processor
5- The Resource Modification Processor
6- The Schedule Processor
7- The Duration Processor

As illustrated in Figure C.1, the sequence generation processors are defined as ART-IM functions in the knowledge-base. Each processor consists mainly of one or more primary functions. The space allocation processor consists of additional secondary functions. The secondary functions are executed from within the primary function of the processor.

Except for the resource modification processor and the duration processors, the execution of the other processors is controlled by the controller module using an object oriented approach. As explained in chapter 11, this is accomplished by sending execution messages to a specific slot of each activity to execute the method defined in that slot. The method defined in each slot will be the name of the primary function of the particular processor. The resource modification processor and the duration processor are executed from within the other five processors by calling the function name defining these processors.



Primary Functions
Sequence Generation Module
Horizontal Logic Processor
Vertical Logic Processor
Resource Allocation Processor
Space Allocation Processor
Resource Modification Processor
Schedule Processor
Duration Processor
Verify-Horizontal-Logic-Constraints-A
Verify-Horizontal-Logic-Constraints-B
Verify-Vertical-Logic-Constraints
Verify-Resource-Constraints
Verify-Work-Space-Constraints
Modify-resource-Demand-Pool
Schedule-A
Schedule-B
Get-Total-Dur
Execute
Secondary Functions
Workspace-Lag-1
Workspace-Lag-2
Workspace-Modify
Execution of these functions is controlled by the Primary Function

Figure C.1 - Processors and Functions of the Sequence Generation Module

The following sections discusses in detail the structure and specific functions of each processor of the sequence generation module.

C.1 The Horizontal Logic Processor:

Horizontal logic constraints define the logical sequence or link between activities of the same floor. Based on such constraints, no activity can be scheduled at any floor unless all its preceding activities in the same floor are scheduled.

The *Horizontal Logic Processor* is responsible for verifying the horizontal logic constraints. During each scheduling cycle, the processor insures that the start date of any activity segment is equal to or greater than the finish date of all its preceding activities at specific critical floors. Critical floors are determined based on the continuity class of the activity. There are ten case scenarios illustrated in Figures C.2 and C.3 defining the different critical floors at which horizontal logic constraints are verified. Figure C.2 depicts five case scenarios of critical floors for activities with continuity class A (i.e. continuous activities). Figure C.3 depicts the other five case scenarios for activities with continuity class B (i.e. intermittent activities).

C.1.1 Horizontal Logic Critical Floors Case Scenarios

Cases 1-5: If the activity has a continuity class A, the activity must be scheduled as one segment along all repetitive floors. Cases 1 through 5 of Figure C.2 depict the critical floors at which the start dates of the activity should be verified with finish dates of

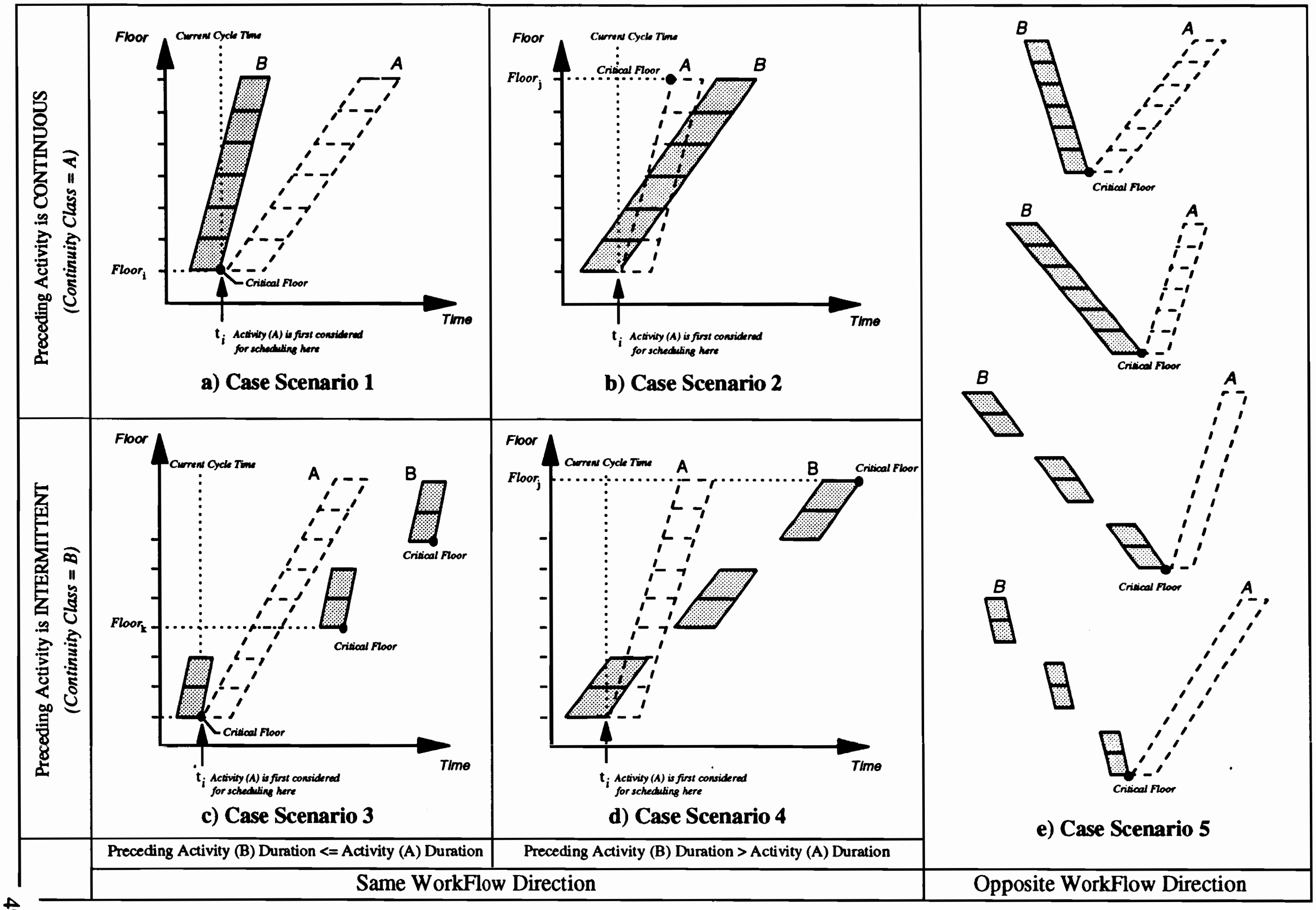


Figure C.2 - Case Scenarios for Continuous Activities (cases 1-5)

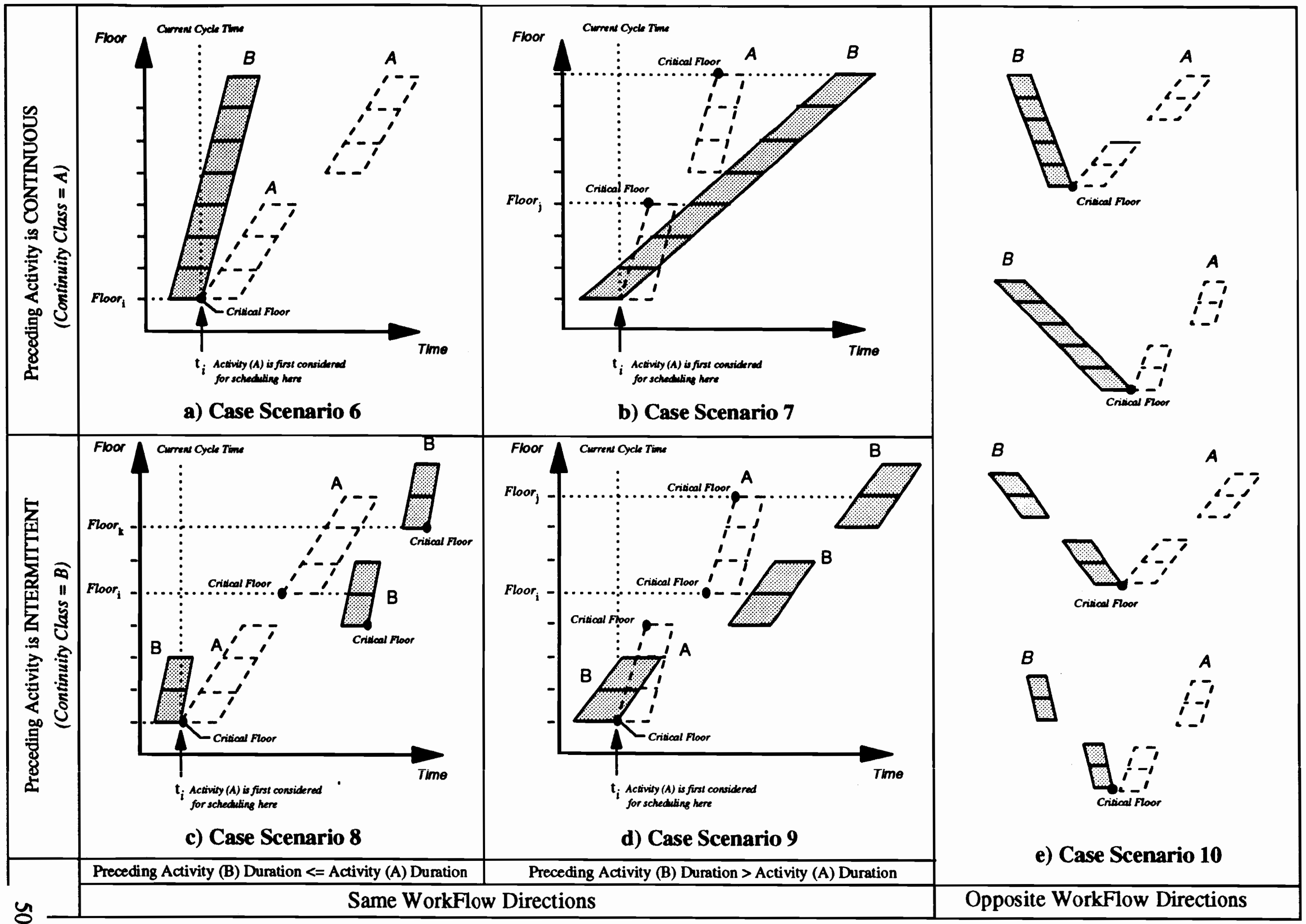

Preceding Activity is CONTINUOUS
(Continuity Class = A)
Preceding Activity is INTERMITTENT
(Continuity Class = B)
Floor
Current Cycle Time
Time
Floor$_i$
Floor$_j$
Floor$_k$
A
B
Critical Floor
t$_i$ Activity (A) is first considered for scheduling here
a) Case Scenario 6
b) Case Scenario 7
c) Case Scenario 8
d) Case Scenario 9
e) Case Scenario 10
Preceding Activity (B) Duration <= Activity (A) Duration
Preceding Activity (B) Duration > Activity (A) Duration
Same WorkFlow Directions
Opposite WorkFlow Directions

Figure C.3 - Case Scenarios for Intermittent Activities (cases 6-10)

preceding activities to insure that horizontal logic constraints are satisfactory.

Case-1

- Workflow direction of preceding activity: *Same as activity*
- Preceding activity continuity status: *Continuous*
- Preceding activity duration: *Less than or equal to activity duration*

This case scenario has only one critical floor. With reference to Figure C.2(a), because preceding activity (B) has a faster pace, the start date of activity (A) is only checked at the first floor (floor-i). This insures that horizontal logic constraints are satisfactory for all other floors.

Case-2

- Workflow direction of preceding activity: *Same as activity*
- Preceding activity continuity status: *Continuous*
- Preceding activity duration: *Greater than activity duration*

Similar to case-1, this case scenario has only one critical floor. As illustrated in Figure C.2(b), because preceding activity (B) has a slower pace than activity (A), start date of (A) is only checked at the last floor (floor-j). This insures that horizontal logic constraints are satisfactory for all other floors.

Case-3

- Workflow direction of preceding activity: *Same as activity*
- Preceding activity continuity status: *Intermittent*
- Preceding activity duration: *Less than or equal to activity duration*

For this case scenario there is a total number of critical floors equal to the number of segments of each preceding activity. As illustrated in Figure C.2(c), because the preceding activity (B) has a faster pace than activity (A), the start date of A is checked at a floor equal to the start floor of each segment of preceding activity (floor-k). This insures that horizontal constraints are satisfactory at all other floors.

Case-4

- Workflow direction of preceding activity: *Same as activity*
- Preceding activity continuity status: *Intermittent*
- Preceding activity duration: *Greater than activity duration*

Similar to case-2, this case scenario has one critical floor. As illustrated in Figure C.2(d), because the preceding activity (B) has a slower pace than activity (A), the start date of A is checked at its last typical floor (floor-j). This insures that horizontal constraints ar satisfactory at all other floors.

Case-5

- Workflow direction of preceding activity: *Opposite to activity*
- Preceding activity continuity status: *Continuous or Intermittent*
- Preceding activity duration: *Less than, equal, or greater than activity duration*

Similar to case scenarios 1 and 2, this case scenario has only one critical floor. The preceding activity continuity class and workflow direction has no influence in this case. As illustrated in Figure C.2(e), because preceding activity (B) has a workflow direction opposite to activity (A), the start date of A is only checked at the first floor. This insures that horizontal logic constraints are satisfactory for all other floors.

Cases 6-10: If the activity has a continuity class B, the activity is scheduled in segments, one segment at a time. Horizontal logic constraints must be checked for each segment. Case 6 through 10 of Figure C.3 depict the critical floors at which each activity segment is verified to insure that the constraints are satisfactory.

Case-6

- Workflow direction of preceding activity: *Same as activity*
- Preceding activity continuity status: *Continuous*
- Preceding activity duration: *Less than or equal to activity duration*

This case scenario has only one critical floor. With reference to Figure C.3(a), because preceding activity (B) has a slower pace than activity (A), the start date of A is only checked at its first floor (floor-i). This insures that horizontal logic constraints are satisfactory for all other floors at subsequent segments.

Case-7

- Workflow direction of preceding activity: *Same as activity*
- Preceding activity continuity status: *Continuous*
- Preceding activity duration: *Greater than activity duration*

This case scenario has one critical floor per activity segment. Because preceding activity (B) has a slower pace then activity (A), the start date of A is checked at the last floor of each segment (floor-j). This insures that horizontal logic constraints ar satisfactory at all other floors of each segment of activity A. This is depicted in Figure C.3(b).

Case-8

- Workflow direction of preceding activity: *Same as activity*
- Preceding activity continuity status: *Intermittent*
- Preceding activity duration: *Greater than activity duration*

This case scenario has a total of two critical floors for each activity segment. With reference to Figure C.3(c), because preceding activity (B) has a faster pace than activity (A), horizontal logic constraints for each segment are verified at the following floors:

1- First floor of activity segment (floor-i)
2- Intermediate floor equal to the first floor of preceding activity (floor-k)

Case-9

- Workflow direction of preceding activity: *Same as activity*
- Preceding activity continuity status: *Intermittent*
- Preceding activity duration: *Equal to or less than activity duration*

Similar to case-8, this case scenario has a total of two critical floors for each activity segment. With reference to Figure C.3(c), because preceding activity (B) has a slower pace than (A), horizontal logic constraints for each activity segment are verified at the following floors:

1- Start floor of activity segment (floor-i)
2- Last floor of activity segment (floor-j)

Case-10

- Workflow direction of preceding activity: *Opposite to activity*
- Preceding activity continuity status: *Continuous or intermittent*
- Preceding activity duration: *Less than, equal, or greater than activity duration*

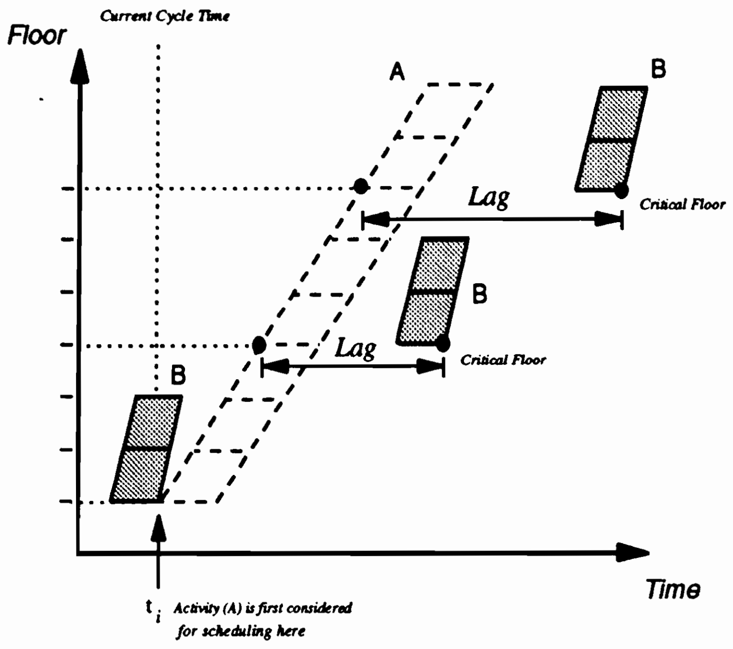
Similar to case-5, this case scenario has only one critical floor. The preceding activity continuity class and workflow direction has no influence in this case. As illustrated in Figure C.3(e), because workflow direction of preceding activity (B) is opposite to that of activity (A), constraints are only verified at the first floor of first activity segment. This will insure that horizontal logic constraints are satisfactory for all other activity segments at all floors. If the workflow direction of activity is up, the critical floor for this case scenario will be the first typical floor.

C.1.2 Verifying Horizontal Logic Constraints for Continuous Activities (Case Scenarios 1, 2, 3, 4, and 5)

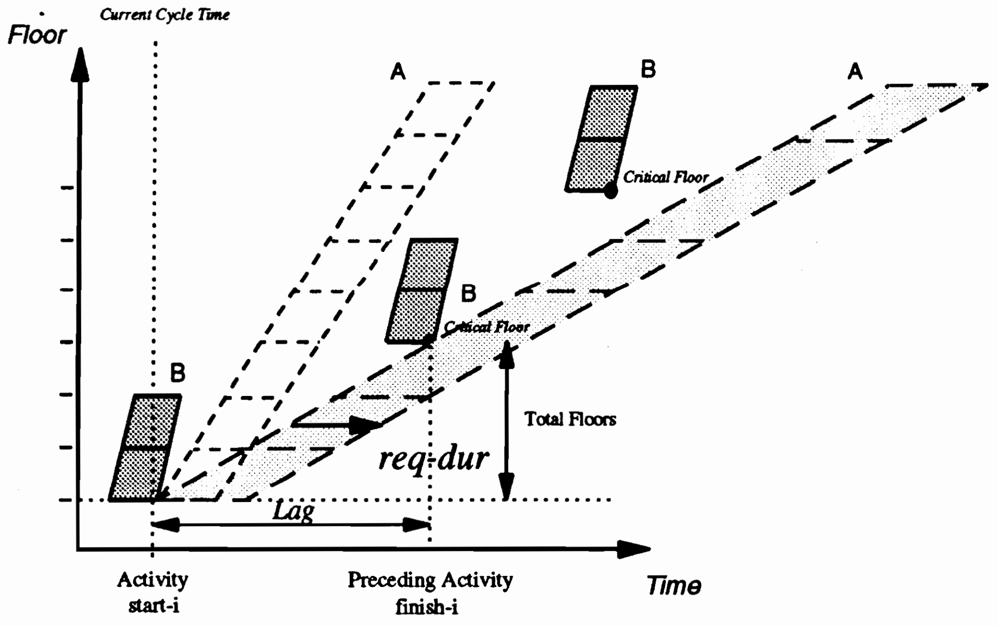
The procedure of verifying the horizontal constraints for continuous activities is based on comparing the start time of activity with the scheduled finish time of all preceding activities at the specific critical floors defined by scenarios 1, 2, 3, 4, and 5. For each critical floor, the procedure determines a time lag value between the preliminary start date of activity and scheduled finish date of each preceding activity.

With reference to Figure C.4(a), the time lag value for any case scenario is expressed as follows:

lag = Preceding activity (B) scheduled finish time at critical floor
- Start time of activity (A) at critical floor



a) Time Lag Values



b) Required Duration Value

Figure C.4 - Time Lag and Duration Values for Continuous Activities

Time lag values are computed for activity at each critical defined in the different case scenarios.

From all time lag values computed, the maximum value is determined,

$$lag_{max} = \max \left[(lag)_1, (lag)_2, \ldots, (lag)_n \right]$$

With reference to previous Figure C.2, it should be noted that computed time lag values at critical floors for case scenarios 1 and 5 will always be equal to or less than zero. This is because any activity is only considered for scheduling at a time equal to or greater than its preceding activity finish at the start floor of activity. Because these critical floors coincides with the start floor of activity, the activity times at these floors will always yield a zero or negative lag value. This condition is determined by the fact that early start value of any activity is always modified once its preceding activity is scheduled, to a value equal to the finish date of this preceding activity at the first or last typical floors.

For example, for case scenario 1 depicted in Figure C.2(a), once activity B is scheduled, the early start value for succeeding activity A is modified to equal the finish date of B at floor i (i.e. t_i). Because any activity becomes eligible for scheduling when the cycle time is equal or greater than its early start value, A will only be considered for scheduling when the scheduling cycle time reaches t_i. As a result, the time lag value for activity A will always be zero or negative for this case scenario. The same condition applies for case scenario 5.

Based on the above, the scheduling procedure of the horizontal logic processor only determines the maximum time lag value for case scenarios 2, 3 and 4 depicted in Figure C.2 (b), (c) and (d) respectively.

At each critical floor of these case scenarios, the procedure also computes a duration value required to reduce the activity pace to eliminate the time lag. This is illustrated in Figure C.4(b). The required duration is computed as follows:

$$req\text{-}dur = \frac{[\text{ preceding activity finish at critical floor-activity start time at its start floor }]}{\text{Total number of floors between first floor and critical floor}}$$

Required duration values are computed for all critical floors in case scenarios 2, 3 and 4. From all duration values computed, a maximum value is determined,

$$req\text{-}dur_{\mathrm{max}} = \max\ [\ (req\text{-}dur)_1,\ (req\text{-}dur)_2,\ \ldots,\ (req\text{-}dur)_{\mathrm{n}}\ ]$$

Scheduling Decisions

If the maximum computed time lag value (lag_{max}) is less than or equal to zero, this indicates that the horizontal logic constraints are satisfactory for the activity. The activity remains in the scheduling cycle as other constraints are continued to be verified.

However, if the maximum lag value (lag_{max}) is greater than zero, horizontal constraints are not satisfactory. Scheduling decisions of the horizontal logic processor attempt to satisfy these constraints by considering one or both of the following issues:

1- Delay the start of activity by a value less than or equal to the maximum lag time computed by the procedure.

2- Decrease activity pace by increasing its duration to a value less than or equal to the maximum limit defined by the user.

Based on the maximum time lag value (lag_{max}) and maximum required duration value ($req\text{-}dur_{max}$), the following scheduling decisions are considered by the processor for continuous activities:

<u>**Decision-1**</u>

Condition: $lag_{max} <= 0$.

Decision: The horizontal logic constraints are satisfactory for the activity. The activity segment remains in the scheduling cycle as further constraints are verified. The attribute value of the '*Scheduling-method*' slot of activity is modified to the function name of the vertical logic processor (i.e. function: *Verify-Vertical-Logic-Constraints*).

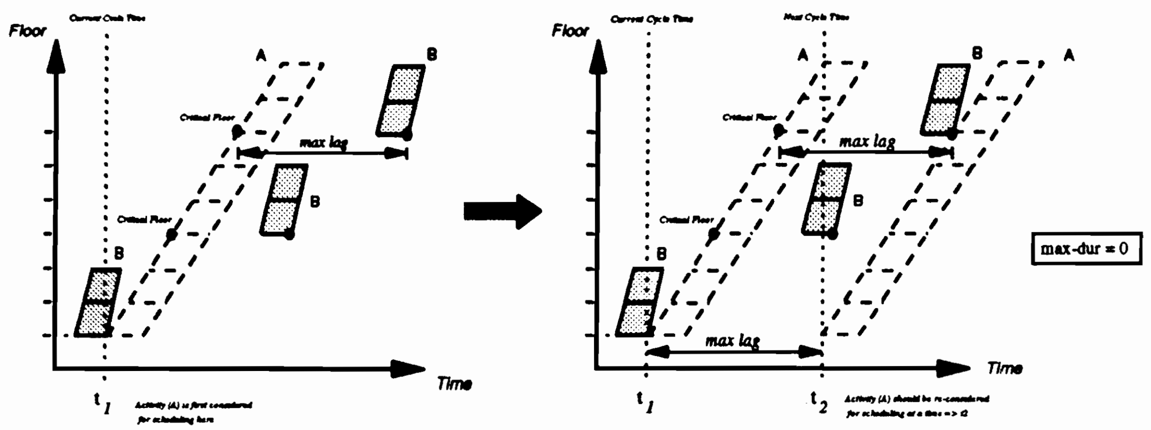
<u>**Decision-2**</u>

Condition: $lag_{max} > 0$, and
activity has no maximum duration value defined by user.

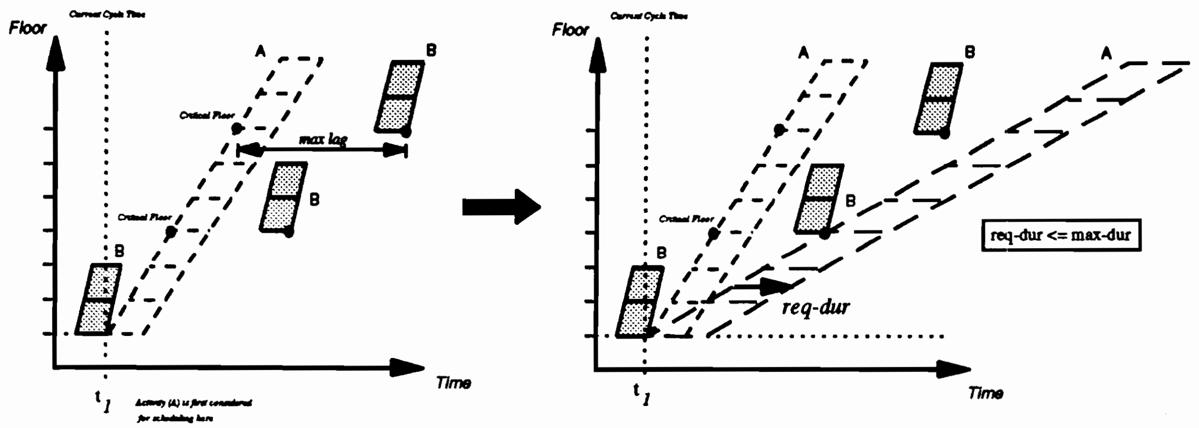
Decision: The activity duration can not be modified beyond its normal value. To satisfy the horizontal constraints, the start date of activity is delayed by the maximum time lag value. This is illustrated in Figure C.5(a).

Based on this decision option, the following actions are made:

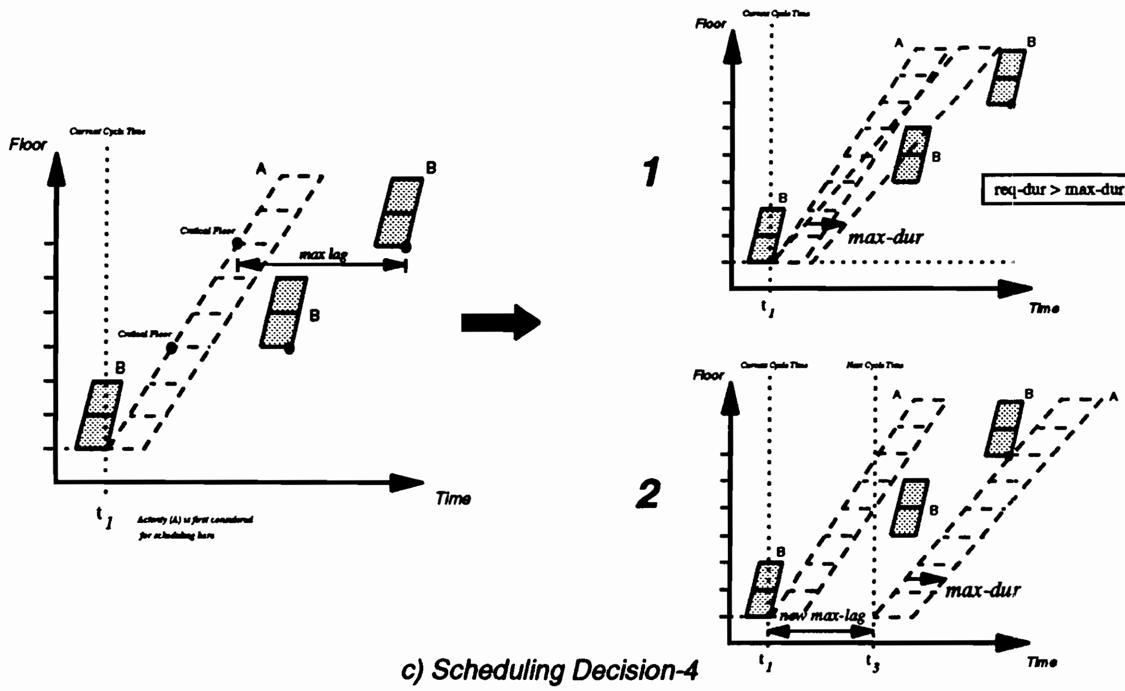
1- Modify early start (ES) value of activity by the maximum time lag value computed,


Floor
Current Cycle Time
Max Cycle Time
A
B
Critical Floor
max lag
max-dur = 0
Time
t1
t2
Activity (A) is first considered for scheduling here
Activity (A) should be re-considered for scheduling at a time => t2

a) Scheduling Decision-2



b) Scheduling Decision-3



c) Scheduling Decision-4

Figure C.5 Decisions for Satisfying Horizontal Constraints

set: $es = t_2 = t_1 + lag_{max}$

2- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time reaches the modified early start value of activity (t_2).

3- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the vertical logic processor (i.e. function: *Verify-Vertical-Logic*). The horizontal logic processor need not be executed when the activity is re-considered for scheduling at t_2. This step will allow to start the vertical logic processor as the first processor to be executed.

__Decision-3__

Condition: $lag_{max} > 0$, and
$req\text{-}dur_{max} <=$ maximum duration value defined by user

Decision: Satisfy horizontal logic constraints by modifying the value of activity duration to the maximum duration value computed by the procedure while maintaining the start time of the activity at the current cycle time (t_1). This is depicted in Figure C.5(b). This option is only considered if the maximum required duration computed by the procedure is less than or equal to the maximum activity duration value defined by the user. The activity remains in the current scheduling cycle as further constraints are verified.

Based on this decision option, the following actions are made:

1- Modify activity duration to the maximum required duration value computed by the procedure,

set: $act\text{-}dur = req\text{-}dur_{max}$

2- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the vertical logic processor (i.e. function: *Verify-Vertical-Logic-Constraints*). This step allows the controller module to execute the vertical logic processor when a new message is sent to the activity.

Decision-4

Condition: $lag_{max} > 0$, and
$req\text{-}dur_{max} >$ maximum duration value defined by user.

Decision: Satisfy the horizontal constraints by increasing the activity duration to the maximum value defined by the user and delay the start of the activity by a value less than the maximum lag computed. This is depicted in Figure C.5(c).

This decision is taken because the maximum required duration computed by the procedure is greater than the maximum duration defined by the user. Increasing the activity duration to the user defined maximum value will not by itself suffice to satisfy the horizontal logic constraints. As a result, the activity start date must be further delayed by a value less than the maximum time lag. A new time lag value is computed using the activity maximum duration.

Based on this decision option, the following actions are taken:

1- Modify activity duration to the maximum duration value defined by the user,

set: act-dur = maximum duration defined by user

2- Determine new maximum time lag value for activity using new activity duration.

3- Modify early start (ES) value of activity by the new maximum time lag value computed,

set: $es = t_3 = t_1 + new\ (lag_{max})$

4- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time reaches the modified early start value of activity (t_2).

5- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the vertical logic processor (i.e. function: *Verify-Vertical-Logic*). The horizontal logic processor need not be executed when the activity is re-considered for scheduling at t_2. This step will allow to start the vertical logic processor as the first processor to be executed.

Execution Function

The ART-IM user defined function:

Function: *Verify-Horizontal-Logic-Constraints-A*

is responsible for verifying horizontal logic constraints for continuous activities. The function is one of two functions that constitute the horizontal logic processor. Figure C.6 depicts a flowchart of the function logic. The general format of the function is as follows:

Function: *Verify-Horizontal-Logic-Constraints-A*

```
1. Set variables:      ?max-lag = 0, and
                       ?max-req-dur = 0


2. For each preceding activity (?prec-act) Do
     2.1 IF: preceding activity has a continuity class A and its workflow direction is
             the same as activity,
         THEN: Verify case scenario 2
                  • Determine time lag (?lag_i) between preceding activity and
```

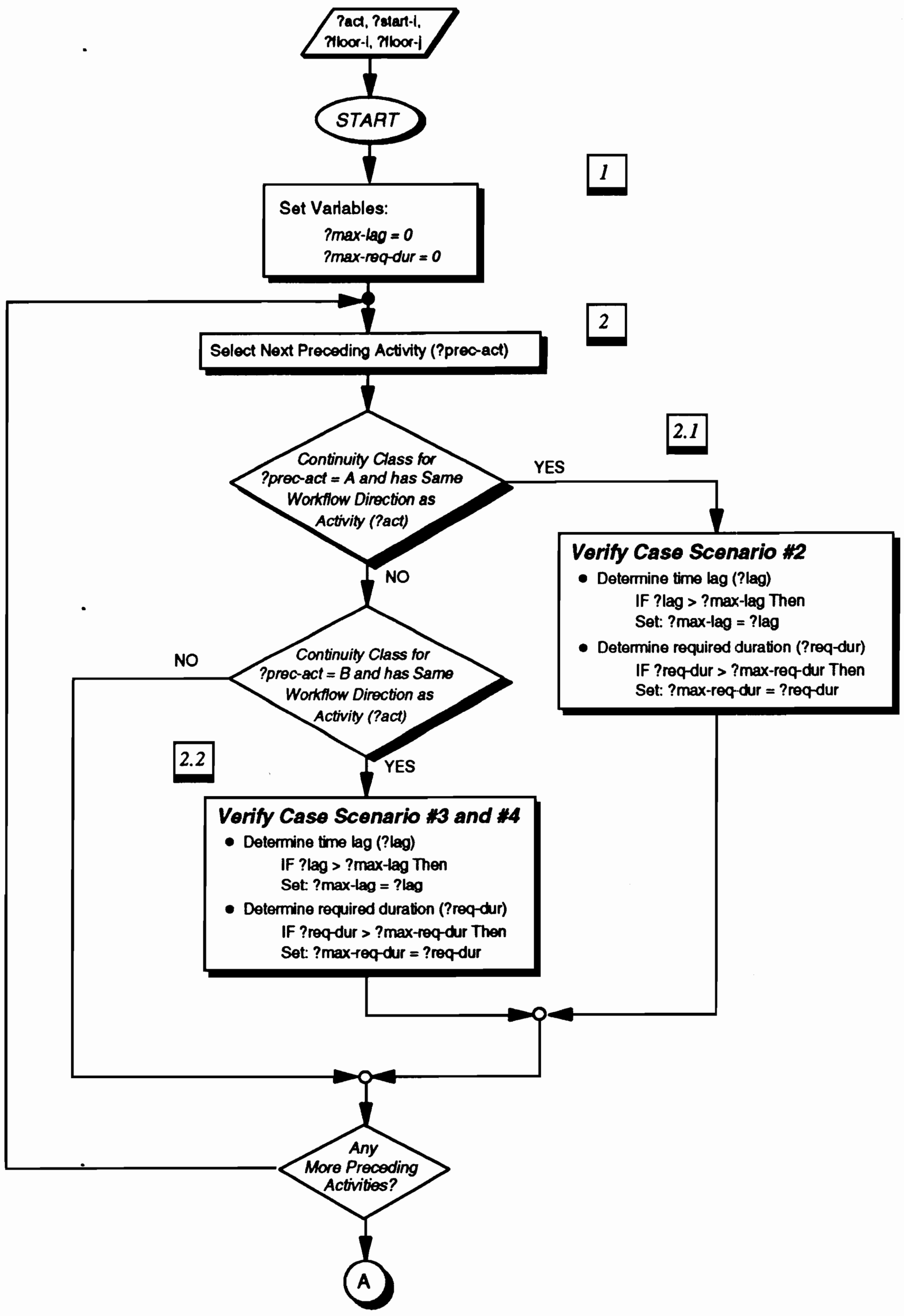

?act, ?start-i, ?floor-i, ?floor-j
START
1
Set Variables:
?max-lag = 0
?max-req-dur = 0
2
Select Next Preceding Activity (?prec-act)
2.1
Continuity Class for ?prec-act = A and has Same Workflow Direction as Activity (?act)
YES
NO
Verify Case Scenario #2
• Determine time lag (?lag)
IF ?lag > ?max-lag Then
Set: ?max-lag = ?lag
• Determine required duration (?req-dur)
IF ?req-dur > ?max-req-dur Then
Set: ?max-req-dur = ?req-dur
NO
Continuity Class for ?prec-act = B and has Same Workflow Direction as Activity (?act)
2.2
YES
Verify Case Scenario #3 and #4
• Determine time lag (?lag)
IF ?lag > ?max-lag Then
Set: ?max-lag = ?lag
• Determine required duration (?req-dur)
IF ?req-dur > ?max-req-dur Then
Set: ?max-req-dur = ?req-dur
Any More Preceding Activities?
A

Figure C.6 - Flowchart for Function: Verify-Horizontal-Logic-Constraints-A

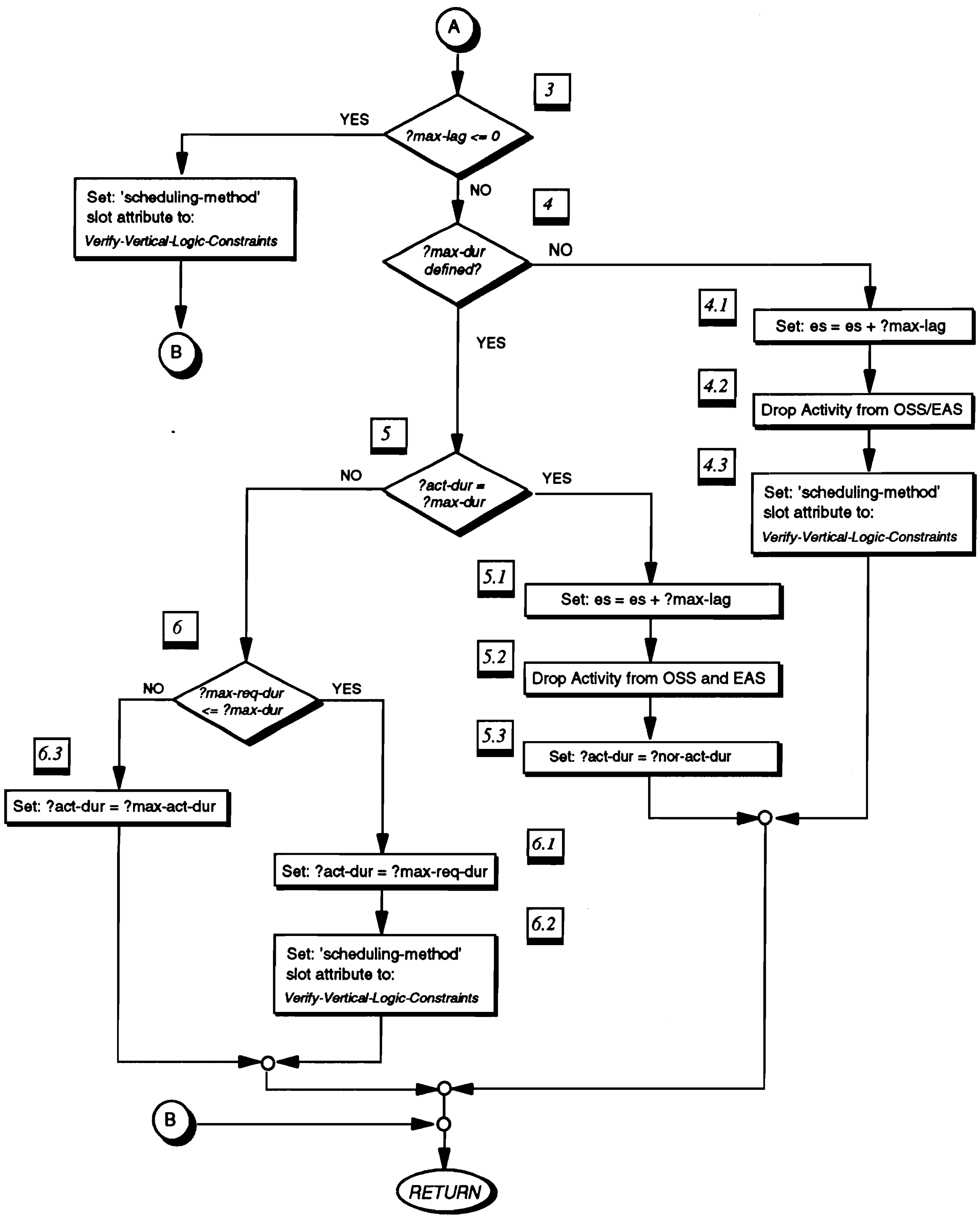

A
3
?max-lag <= 0
YES
NO
Set: 'scheduling-method' slot attribute to: Verify-Vertical-Logic-Constraints
B
4
?max-dur defined?
NO
YES
4.1
Set: es = es + ?max-lag
4.2
Drop Activity from OSS/EAS
4.3
Set: 'scheduling-method' slot attribute to: Verify-Vertical-Logic-Constraints
5
?act-dur = ?max-dur
NO
YES
5.1
Set: es = es + ?max-lag
5.2
Drop Activity from OSS and EAS
5.3
Set: ?act-dur = ?nor-act-dur
6
?max-req-dur <= ?max-dur
NO
YES
6.3
Set: ?act-dur = ?max-act-dur
6.1
Set: ?act-dur = ?max-req-dur
6.2
Set: 'scheduling-method' slot attribute to: Verify-Vertical-Logic-Constraints
B
RETURN

Figure C.6 - Continued

```
activity at all critical floors
        IF: ?lag$_i$ > ?max-lag
        THEN: set: ?max-lag = ?lag$_i$

    • Determine required duration (?req-dur$_i$) for activity at each
    critical floor,
            IF: ?req-dur$_i$ > ?max-req-dur
            THEN: set: ?max-req-dur = ?req-dur$_i$

2.2 IF: preceding activity has a continuity class B and its workflow direction is
        the same as activity,
    THEN: Verify case scenario 3, and 4

    • Determine time lag (?lag$_i$) between preceding activity and
    activity at all critical floors
            IF: ?lag$_i$ > ?max-lag
            THEN: set: ?max-lag = ?lag$_i$

    • Determine required duration (?req-dur$_i$) for activity at each
    critical floor,
            IF: ?req-dur$_i$ > ?max-req-dur
            THEN: set: ?max-req-dur = ?req-dur$_i$

3. IF: ?max-lag <= 0
   THEN: Modify the attribute value of the 'Scheduling-method' slot of activity to the
        function name of the vertical logic processor (i.e. function: Verify-Vertical-Logic
        -Constraints).

4. IF: ?max-lag > 0 and the activity has no maximum duration value defined by user,
   THEN:
        4.1 Modify early start (ES) value of activity by the maximum lag value
            computed,

                set: es = es + ?max-lag

        4.2 Remove activity from scheduling cycle and drop activity number from
            the ordered activity set (OSS) and the eligible activity set (EAS).

        4.3 Modify the attribute value of the 'Scheduling-method' slot of activity
            to the function name of the vertical logic processor (i.e. function:
            Verify-Vertical-Logic).
```

```
5. IF:  ?max-lag > 0, and
        the activity has a maximum duration value defined by the user, and
        the current activity duration value (?act-dur) is already modified to the maximum
        value,

   THEN:
           5.1 Modify early start (ES) value of activity by the maximum lag value
               computed,

                       set: es = es + ?max-lag

           5.2 Drop activity from ordered activity set (OSS) and eligible activity set
               (EAS).

           5.3 Modify activity duration (?act-dur) to the normal duration value,

                       set: ?act-dur = ?nor-act-dur

6. IF:  ?max-lag > 0, and
        the activity has a maximum duration value defined by the user, and
        the current activity duration value (?act-dur) is less than the maximum value,
   THEN:
           IF: ?max-req-dur <= ?max-act-dur
           THEN:
                   6.1 Modify activity duration to the maximum required duration
                       computed by the procedure,

                       set: ?act-dur = ?max-req-dur

                   6.2 Modify the attribute value of the 'Scheduling-method' slot of
                       activity to the function name of the vertical logic processor (i.e.
                       function: Verify-Vertical-Logic-Constraints).

           ELSE:   6.3 Modify activity duration to the maximum activity duration
                       defined by the user,

                               set: ?act-dur = ?max-act-dur
```

The complete source code of the function is provided in Appendix G of this document.

C.1.3 Verifying Horizontal Logic Constraints for Intermittent Activities (Case Scenarios 6, 7, 8, 9 and 10)

The procedure of verifying horizontal constraints for intermittent activities is similar to that for continuous activities. For each activity, the procedure determines all possible time lag values between the activity and preceding activities at all critical floors defined by the different case scenarios. However, because intermittent activities are scheduled in segments, two types of lag values are determined as illustrated in Figure C.7(a). The first type of lag values (lag-1) is determined at the critical floors coinciding with the start floor of the activity segment. The second type of lag values (lag-2) is determined at other critical floors. The distinction between the two types of lags is made because of the different scheduling decisions considered to account for each type.

With reference to Figure C.7(a), for any case scenario time lag values are expressed as follows:

lag-1 = Preceding activity (B) scheduled finish time at critical floor (floor_i) - Start time of activity (A) at that critical floor.

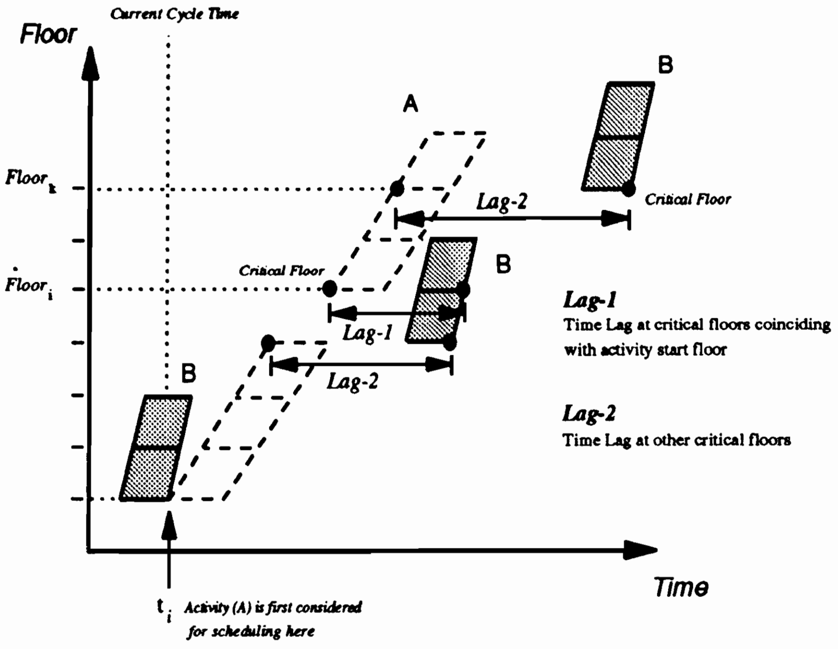
lag-2 = Preceding activity (B) scheduled finish time at other critical floors (floor_j or floor_k) - Start time of activity (A) at these floors.

From all time lag values computed, two maximum values are determined,

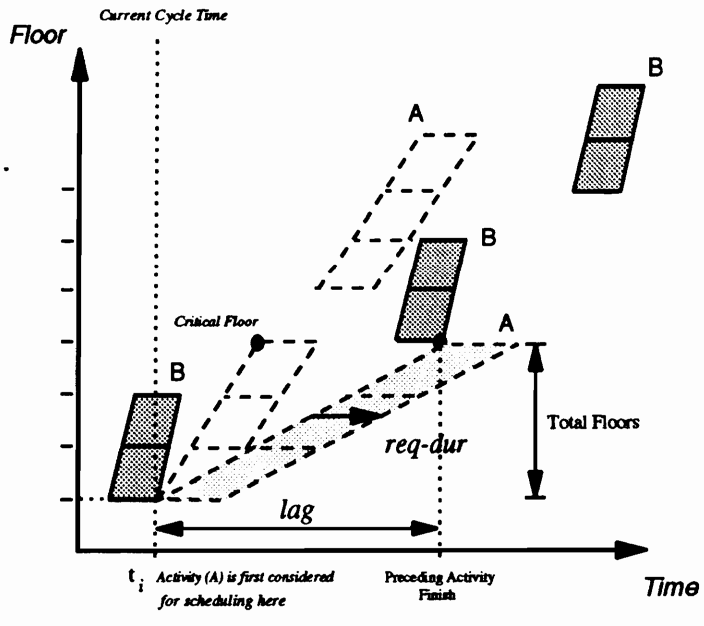
$$\textit{lag-1}_{\max} = \max\ [\ (\textit{lag-1})_1, (\textit{lag-1})_2, ..., (\textit{lag-1})_n\ ], \text{ and}$$

$$\textit{lag-2}_{\max} = \max\ [\ (\textit{lag-2})_1, (\textit{lag-2})_2, ..., (\textit{lag-2})_n\ ]$$

It should be noted that similar to case scenarios 1 and 5, case scenarios 6 and 10 are not


Current Cycle Time
Floor
Floor k
Floor i
A
B
B
B
Critical Floor
Critical Floor
Lag-2
Lag-1
Lag-2
Lag-1
Time Lag at critical floors coinciding with activity start floor
Lag-2
Time Lag at other critical floors
Time
t i Activity (A) is first considered for scheduling here

a) Time Lag Values


Current Cycle Time
Floor
A
B
B
B
A
Critical Floor
req-dur
Total Floors
lag
t i Activity (A) is first considered for scheduling here
Preceding Activity Finish
Time

b) Required Duration Value

Figure C.7 - Time Lag and Duration Values for Intremittent Activities

considered in the verification process. This is because lag values determined for these two scenarios will always be less than or equal to zero. As a result, time lag values are determined for case 7, 8 and 9 only.

At each critical floor of case scenarios 6 and 7, the procedure also computes the duration value required to reduce the activity pace to eliminate the time lag (lag-2 only). As illustrated in Figure C.7(b) the required duration is computed as follows:

$$\textit{req-dur} = \frac{\left[\text{preceding activity finish time at critical floors (floor}_j \text{ or floor}_k\text{)} - \text{activity start time at its first floor (floor}_i\text{)}\right]}{\text{Total number of floors between first floor (i) and critical floor (j or k)}}$$

Required duration values are computed for all critical floors in case scenarios 7, 8, and 9 where lag-2 values are computed. From all duration values computed, a maximum value is determined,

$$\textit{req-dur}_{\max} = \max\left[\,(\textit{req-dur})_1, (\textit{req-dur})_2, \ldots, (\textit{req-dur})_n\,\right]$$

Scheduling Decisions

If both maximum time lag values ($\textit{lag-1}_{max}$, $\textit{lag}_{max}$) computed by the procedure are less than or equal to zero, this will indicate that the horizontal logic constraints are satisfactory for the activity. The activity remains in the scheduling cycle as other constraints are verified.

However, if one or both of these maximum values are greater than zero, this will indicate that horizontal constraints are not satisfactory. Scheduling decisions of the horizontal logic processor attempt to satisfy these constraints by considering one or both of the following issues:

1- Delay the start of activity by a value less than or equal to the maximum lag time value computed by the procedure.

2- Decrease activity pace by increasing its duration to the maximum limit defined by the user.

Based on the computed maximum time lag values (*lag-1*$_{max}$, *lag-2*$_{max}$) and maximum required duration value (*req-dur*$_{max}$), the following decisions are considered by the processor:

Decision-1

Condition: lag-1$_{max}$ <= 0 , and
lag-2$_{max}$ <= 0

Decision: The horizontal logic constraints are satisfactory for the activity. The activity segment remains in the scheduling cycle as further constraints are verified. The attribute value of the '*Scheduling-method*' slot of activity is modified to the function name of the vertical logic processor (i.e. function: *Verify-Vertical-Logic-Constraints*).

Decision-2

Condition: lag-1$_{max}$ > 0

Decision: The start date of activity is delayed by the maximum time lag value (lag-1$_{max}$).

This decision is made regardless of the value of lag-2_{max} computed.

As illustrated in Figure C.8(a), the start date of the activity segment is shifted by a value equal to lag-1_{max} from time t_1 to t_2. This is the only possible decision way to satisfy horizontal constraints at the first floor of activity segment.

Based on this decision option, the following actions are made:

1- Modify early start (ES) value of activity by the maximum time lag-1 value computed,

set: $es = t_2 = t_1 + \text{lag-}1_{max}$

2- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time equals to the modified early start value of activity (t_2).

Decision-3

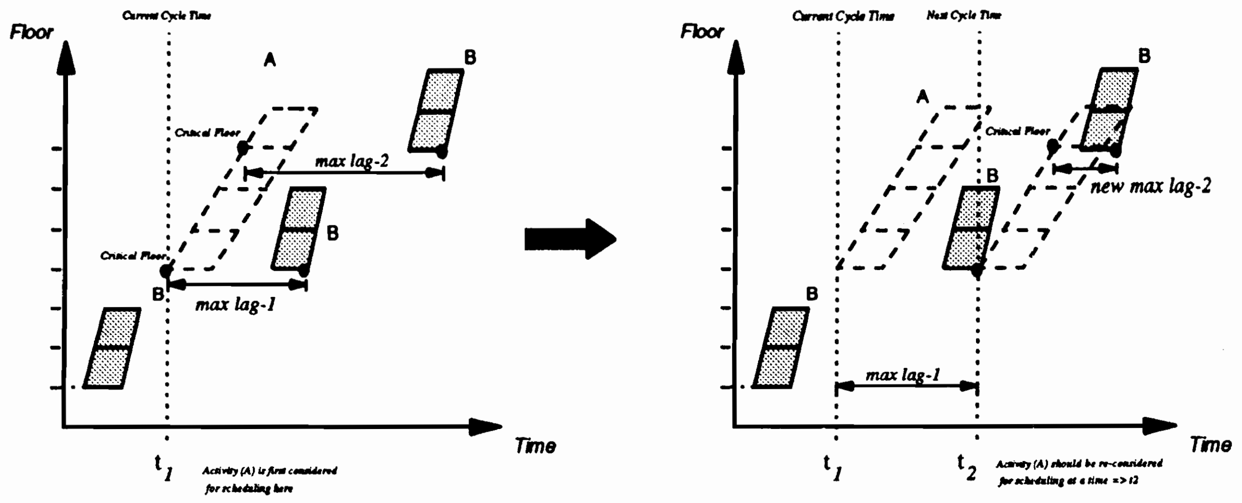
Condition: lag-1_{max} <= 0, and
lag-2_{max} >0, and
activity has no maximum duration value defined by user.

Decision: Because the activity has no maximum duration value, the activity duration can not be modified beyond the normal value. The start date of activity segment must be delayed by the value of lag-2_{max} computed. This is illustrated in Figure C.8(b).

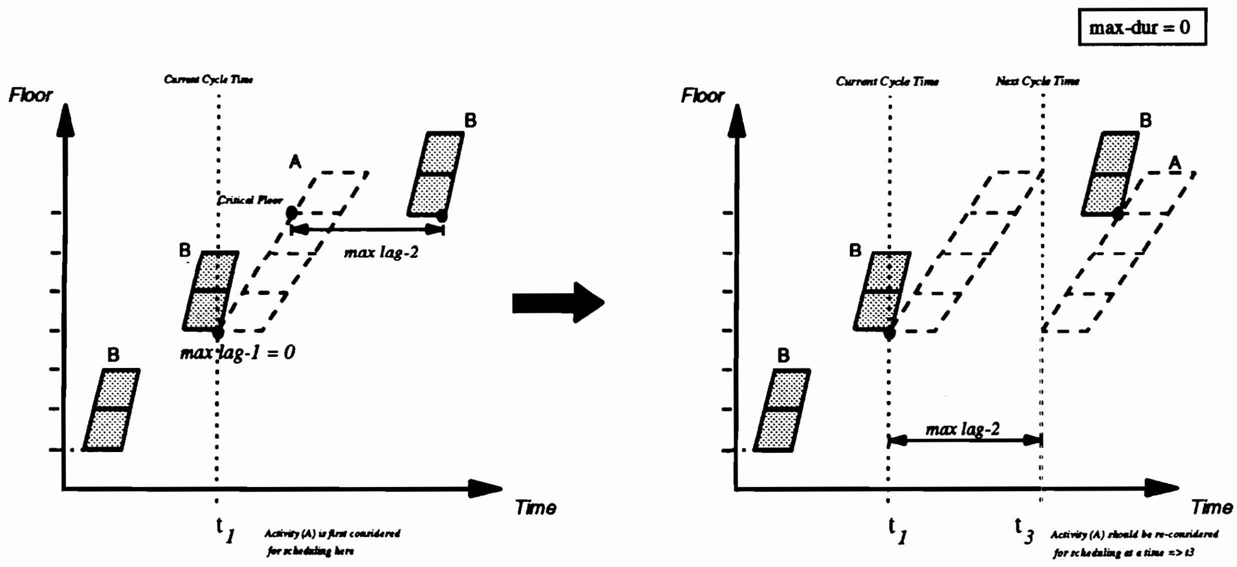
Based on this decision option, the following actions are considered:

1- Modify early start (ES) value of activity by the maximum time lag-2 value computed,

set: $es = t_3 = t_1 + \text{lag-}2_{max}$

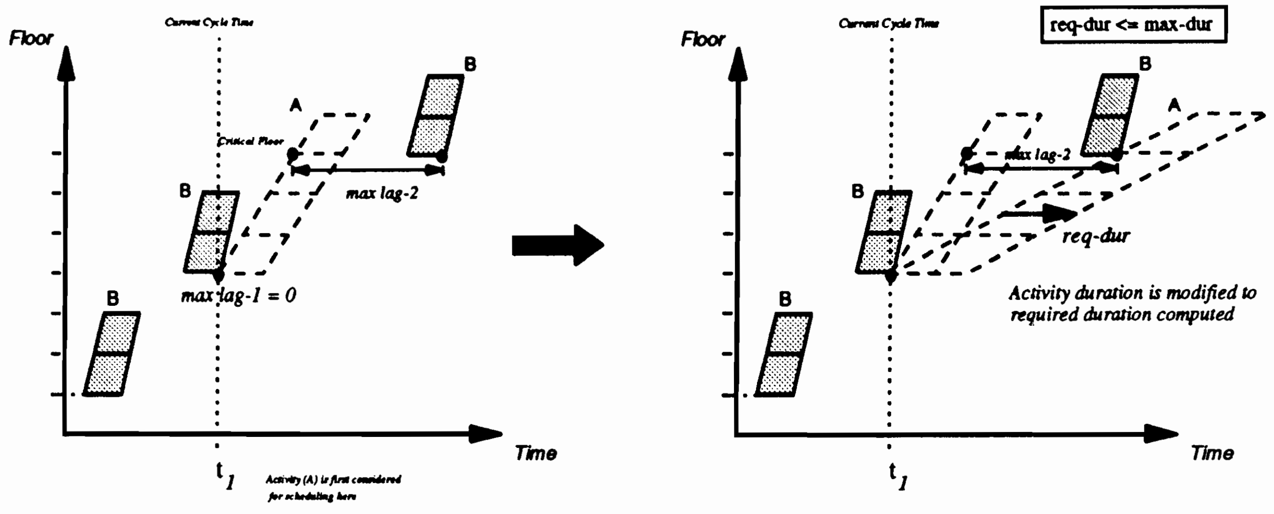


a) Scheduling Decision-2

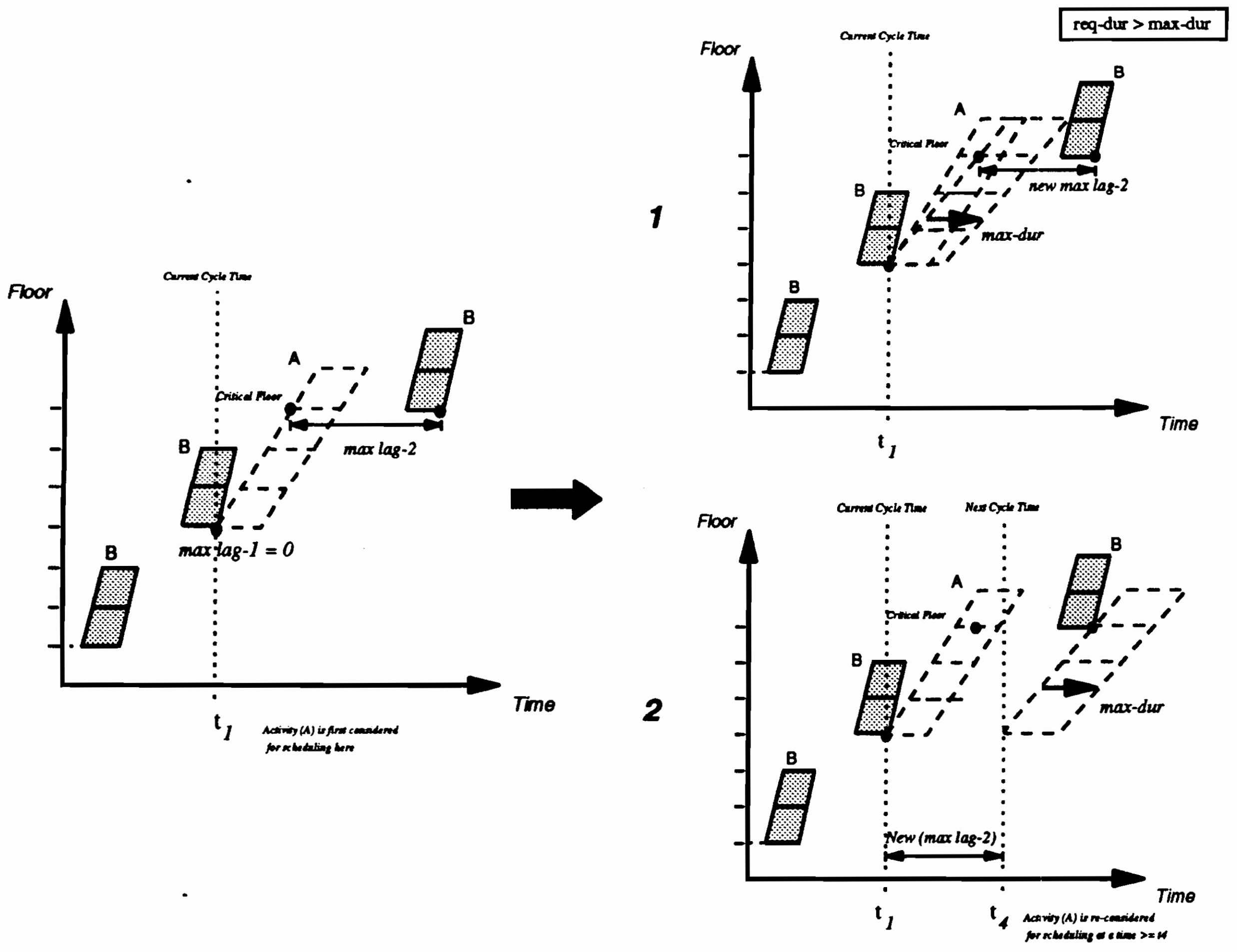


b) Scheduling Decision-3

Figure C.8 - Decions for Satisfying Horizontal Logic Constraints



c) Scheduling Decision-4



d) Scheduling Decision-5

Figure C.8 - Continued

2- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time equals to the modified early start value of activity (t_3).

3- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the vertical logic processor (i.e. function: *Verify-Vertical-Logic-Constraints*). The horizontal logic processor need not be executed when the activity is re-considered for scheduling at t_3. This step will allow to start the vertical logic processor as the first processor to be executed.

Decision-4

Condition: lag-1_{max} <= 0, and
lag-2_{max} > 0, and
req-dur_{max} <= maximum activity duration defined by user

Decision: To satisfy the horizontal logic constraints, activity duration is modified to the maximum required value determined by the procedure while maintaining the start time of activity at the current cycle time t_1. This decision will only be considered if the maximum required duration (req-dur_{max}) computed by the procedure is less than or equal to the activity maximum duration value defined by user. This is illustrated in Figure C.8c

Based on this decision the following actions are taken:

1- Modify activity duration to the maximum required duration computed,

set: act-dur = req-dur_{max}

2- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the vertical logic processor (i.e. function: *Verify-Vertical-Logic -Constraints*).

Decision-5

Condition: lag-1_{max} <= 0, and

$lag\text{-}2_{max} > 0$, and
$req\text{-}dur_{max} >$ maximum activity duration

Decision: Satisfy the horizontal constraints by increasing the activity duration to the maximum value defined by the user and delay the start of activity by a value less than the maximum lag computed. This is depicted in Figure C.8(d).

Similar to the situation defined in decision-4 for continuous activities, this decision is taken because increasing the activity duration to the user defined maximum value will not by itself suffice to satisfy the horizontal logic constraints. As a result, the activity start date must be further delayed by a new value less than the maximum time lag computed. The new time lag value is computed using the activity maximum duration.

With reference to Figure C.8(d), based on this decision option the following actions are taken:

1- Modify activity duration to the maximum duration value defined by the user,

set: act-dur = maximum duration defined by user

2- Determine new maximum time lag-2 value for activity using new activity duration.

3- Modify early start (ES) value of activity by the new maximum time lag value computed,

set: $es = t_4 = t_1 + lag\text{-}2_{max}$

4- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time reaches the modified early start value of activity (t_4).

5- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the vertical logic processor (i.e. function: *Verify-Vertical-Logic-Constraints*). The horizontal logic processor need not be executed when the activity is re-considered for scheduling at t_4. This step will allow to start the vertical logic processor as the first processor to be executed.

Execution Function

The ART-IM user defined function:

Function: *Verify-Horizontal-Logic-Constraints-B*

is responsible for verifying horizontal logic constraints for intermittent activities. This is the second function of the horizontal logic processor. Figure C.9 depicts a flowchart of the function logic. The general format of the function is as follows:

Function: *Verify-Horizontal-Logic-Constraints-B*

```
1. Set variables :    ?max-lag-1 = 0,
                      ?max-lag-2 = 0,
                      ?max-req-dur = 0

2. For each preceding activity (?prec-act) Do
      2.1 IF: preceding activity has a continuity class A and its workflow direction is
              the same as activity,
         THEN: Verify case scenario 7
                  • Determine time lag values ?lag-1_i and ?lag-2_i between preceding
                    activity and activity at all critical floors

                        IF: ?lag-1_i > ?max-lag-1
                        THEN: set: ?max-lag-1 = ?lag-1_i

                        IF: ?lag-2_i > ?max-lag-2
                        THEN: set: ?max-lag-2 = ?lag-2_i

                  • Determine required duration (?req-dur_i) for activity at each critical
                    floor,
                        IF: ?req-dur_i > ?max-req-dur
                        THEN: set: ?max-req-dur = ?req-dur_i
```

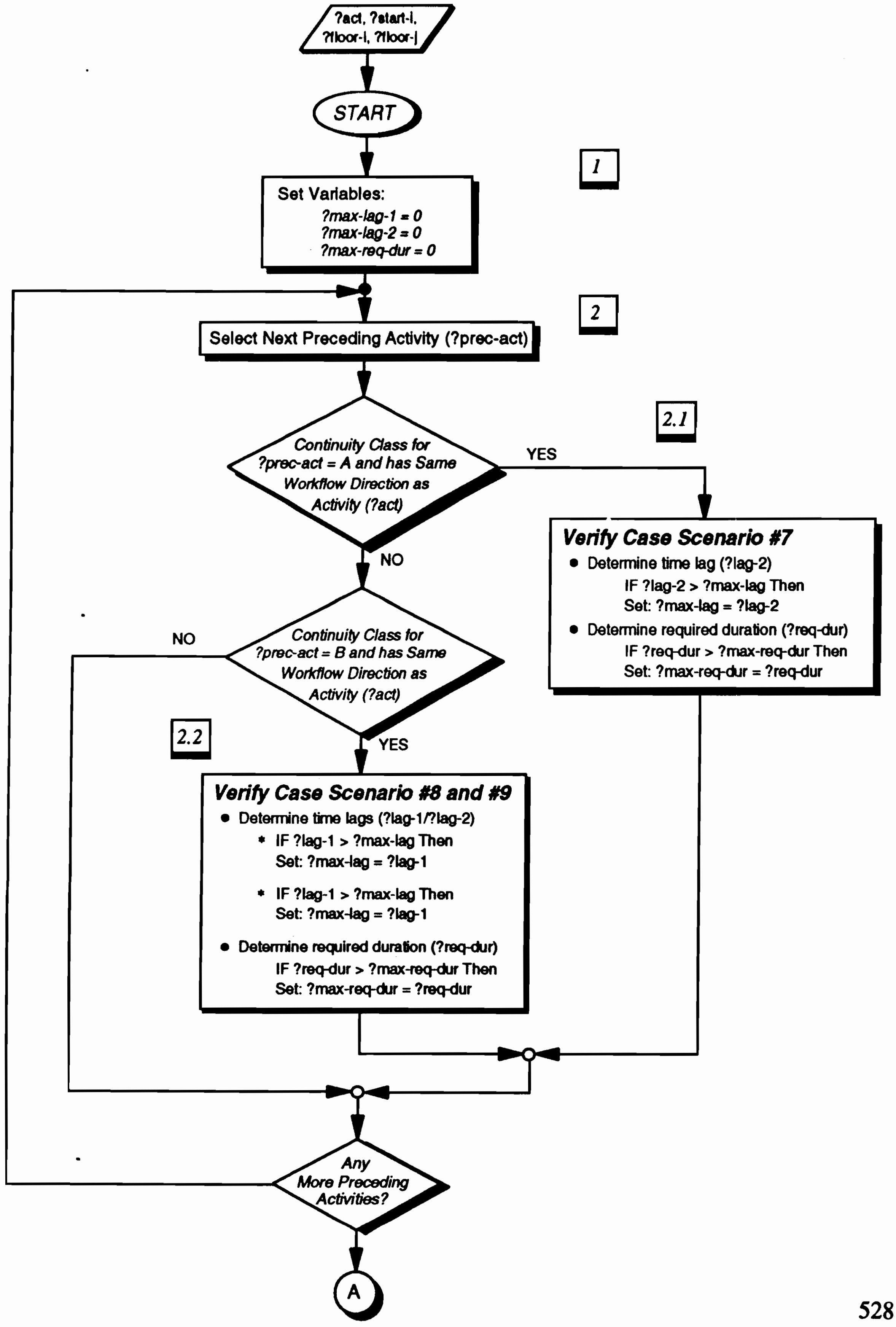
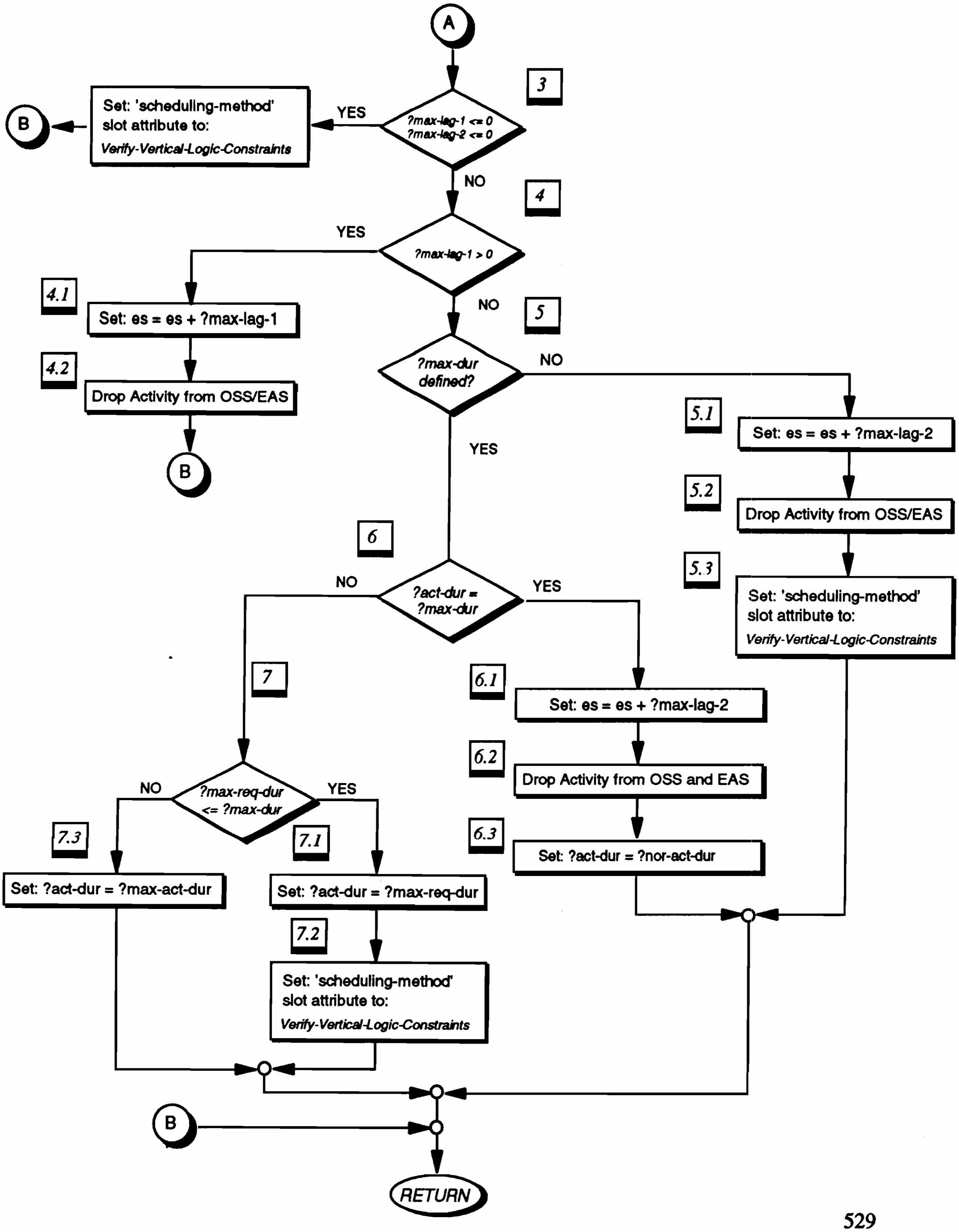

?act, ?start-l, ?floor-l, ?floor-j
START
1
Set Variables:
?max-lag-1 = 0
?max-lag-2 = 0
?max-req-dur = 0
2
Select Next Preceding Activity (?prec-act)
2.1
Continuity Class for ?prec-act = A and has Same Workflow Direction as Activity (?act)
YES
NO
Verify Case Scenario #7
● Determine time lag (?lag-2)
IF ?lag-2 > ?max-lag Then
Set: ?max-lag = ?lag-2
● Determine required duration (?req-dur)
IF ?req-dur > ?max-req-dur Then
Set: ?max-req-dur = ?req-dur
NO
Continuity Class for ?prec-act = B and has Same Workflow Direction as Activity (?act)
2.2
YES
Verify Case Scenario #8 and #9
● Determine time lags (?lag-1/?lag-2)
* IF ?lag-1 > ?max-lag Then
Set: ?max-lag = ?lag-1
* IF ?lag-1 > ?max-lag Then
Set: ?max-lag = ?lag-1
● Determine required duration (?req-dur)
IF ?req-dur > ?max-req-dur Then
Set: ?max-req-dur = ?req-dur
Any More Preceding Activities?
A

Figure C.9 - Flowchart for Function: Verify-Horizontal-Logic-Constraints-B



A
3
?max-lag-1 <= 0
?max-lag-2 <= 0
YES
Set: 'scheduling-method' slot attribute to: *Verify-Vertical-Logic-Constraints*
B
NO
4
?max-lag-1 > 0
YES
4.1
Set: es = es + ?max-lag-1
4.2
Drop Activity from OSS/EAS
B
NO
5
?max-dur defined?
NO
5.1
Set: es = es + ?max-lag-2
5.2
Drop Activity from OSS/EAS
5.3
Set: 'scheduling-method' slot attribute to: *Verify-Vertical-Logic-Constraints*
YES
6
?act-dur = ?max-dur
YES
6.1
Set: es = es + ?max-lag-2
6.2
Drop Activity from OSS and EAS
6.3
Set: ?act-dur = ?nor-act-dur
NO
7
?max-req-dur <= ?max-dur
YES
7.1
Set: ?act-dur = ?max-req-dur
7.2
Set: 'scheduling-method' slot attribute to: *Verify-Vertical-Logic-Constraints*
NO
7.3
Set: ?act-dur = ?max-act-dur
B
RETURN

Figure C.9 - Continued

```
2.2 IF: preceding activity has a continuity class B and its workflow direction is
        the same as activity,
    THEN: Verify case scenario 8, and 9
          • Determine time lag values ?lag-1_i and ?lag-2_i between preceding
            activity and activity at all critical floors

                  IF: ?lag-1_i > ?max-lag-1
                  THEN: set: ?max-lag-1 = ?lag-1_i

                  IF: ?lag-2_i > ?max-lag-2
                  THEN: set: ?max-lag-2 = ?lag-2_i

          • Determine required duration (?req-dur_i) for activity at each critical
            floor,
                  IF: ?req-dur_i > ?max-req-dur
                  THEN: set ?max-req-dur = ?req-dur_i


3. IF: ?max-lag-1 <= 0,
       and ?max-lag-2 <= 0
   THEN: Modify the attribute value of the 'Scheduling-method' slot of activity to the
         function name of the vertical logic processor (i.e. function: Verify-Vertical-Logic
         -Constraints).

4. IF: ?max-lag-1 > 0,
   THEN:
      4.1 Modify early start (ES) value of activity by the maximum lag value computed,

                set:  es = es + ?max-lag-1

      4.2 Drop activity from ordered activity set (OSS) and eligible activity set (EAS).


5. IF:  ?max-lag-1 <= 0, and
        ?max-lag-2 > 0, and
        the activity has no maximum duration value defined by the user,
   THEN:
      5.1 Modify early start (ES) value of activity by the maximum lag value computed,

                set: es = es + ?max-lag-2

      5.2 Drop activity from ordered activity set (OSS) and eligible activity set (EAS).

      5.3 Modify attribute value of the 'Scheduling-method' slot of activity to the
      function name of the vertical logic processor (i.e. function: Verify-Vertical
      -Logic-Constraints).


6. IF:  ?max-lag-1 <= 0, and
        ?max-lag-2 > 0, and
        the activity has a maximum duration value defined by the user, and
        the current activity duration (?act-dur) is already modified to the maximum limit,

   THEN:
        6.1 Modify early start (ES) value of activity by the maximum lag value computed,

                      set:  es = es + ?max-lag-2

        6.2 Drop activity from ordered activity set (OSS) and eligible activity set (EAS).

        6.3 Modify activity duration (?act-dur) to the normal duration value,

                      set:  ?act-dur = ?nor-act-dur

7.  IF: ?max-lag-1 <= 0, and
        ?max-lag-2 > 0, and
        the activity has a maximum duration value defined by the user, and
        the current activity duration value (?act-dur) is less than the maximum duration
        value,

   THEN:
        IF: ?max-req-dur <= ?max-act-dur
        THEN:
              7.1 Modify activity duration to the maximum required duration computed
                  by the procedure,

                      set: ?act-dur = ?max-req-dur

              7.2 Modify the attribute value of the 'Scheduling-method' slot of activity
                  to the function name of the vertical logic processor (i.e. function: Verify
                  -Vertical-Logic).

        ELSE: 7.3 Modify activity duration to the maximum activity duration defined
                  by the user,

                      set: ?act-dur = ?max-act-dur
```

The complete source code of the function is provided in Appendix G of this document.

C.2 The Vertical Logic Processor:

Vertical logic constraints are based on logical links between activities at different floors. Because of such constraints, the execution of an activity at a specific floor affects or controls the start or finish of another activity on a lower floor. This type of constraints establishes a vertical logic or sequence among activities of the different floors. This is opposed to the horizontal logic constraints that establish logical sequence among activities of the same floor.

Vertical logic constraints are defined for each activity using two parameters; vertical logic activity (ver-log-act) and the vertical logic lag (ver-log-lag). As illustrated in Figure C.10, the first parameter specifies the activity number forcing the vertical logic constraint on the activity. The second parameter specifies the floor buffer (i.e. vertical lag) imposed on the activity.

During each scheduling cycle, the *Vertical Logic Processor* is responsible for verifying the vertical logic constraints for each activity. The procedure adopted by the processor for verifying vertical constraints is similar to that for verifying horizontal constraints. The start date of the activity is checked with the finish date of the activity forcing the

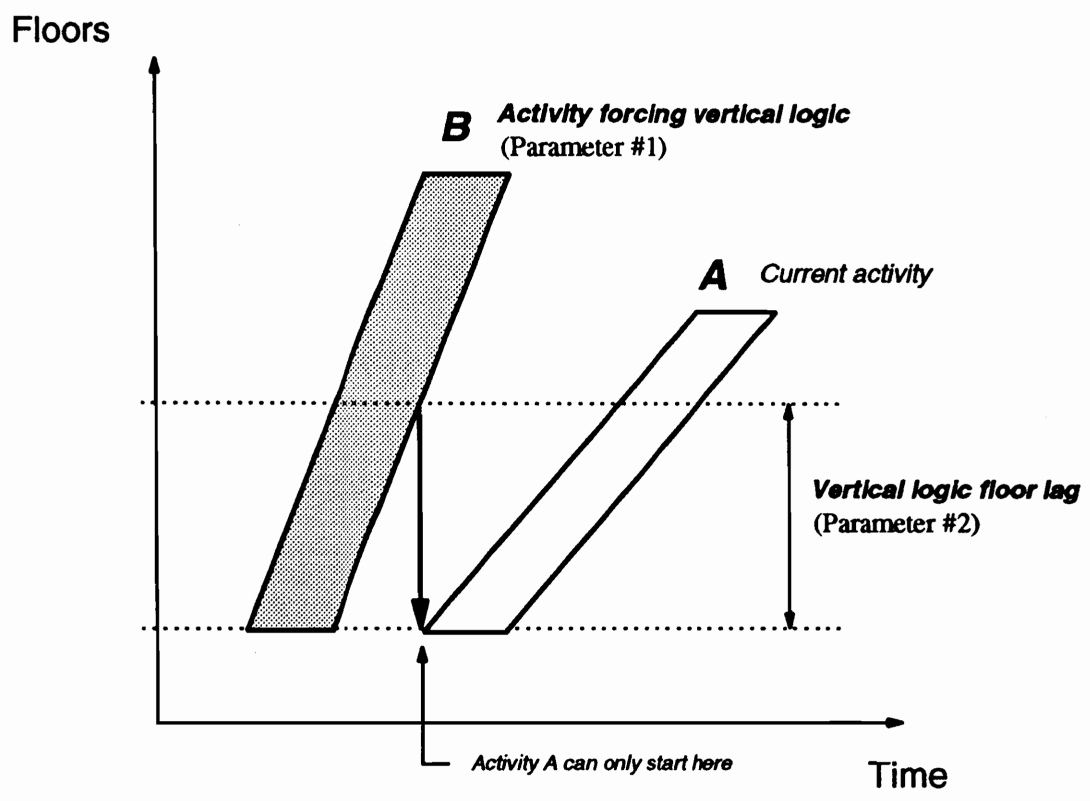


Figure C.10 - Vertical Logic Parameters

constraint at specific critical floors. As illustrated in Figure C.11, critical floors are determined based on the end floors of activity segment and possible end floors of activity segments forcing the constraint. Critical floors depicted in Figure C.11 are:

1- First floor of activity A plus vertical lag (*floor-i* + *ver-lag*).

2- Last floor of activity A plus the vertical lag (*floor-j* + *ver-lag*).

3- End floor of activity B (segment$_1$) less the vertical lag (*floor-x* - *ver-lag*).

4- Start floor of activity B (segment$_2$) less the vertical lag (*floor-y* - *ver-lag*).

Based on the critical floor for any activity segment, the procedure determines a time lag value between the activity and the activity forcing the vertical constraint at each possible critical floor. Two types of lag values are determined; lag-1 and lag-2. The distinction between the two types is made because of the different scheduling decisions considered.

As depicted in Figure C.11, the first type of lag values (lag-1) is determined at the critical floor based on the start floor of activity segment (i.e. floor-i + ver-lag) and is computed as follows:

Lag-1 = Scheduled finish date of B at (floor-i + ver-lag)
- preliminary start date of A (floor-i + ver-lag)

Because there is only one activity forcing the constraint, only one lag-1 value is computed.

The second type of lag values (lag-2) is determined at all other critical floors. For

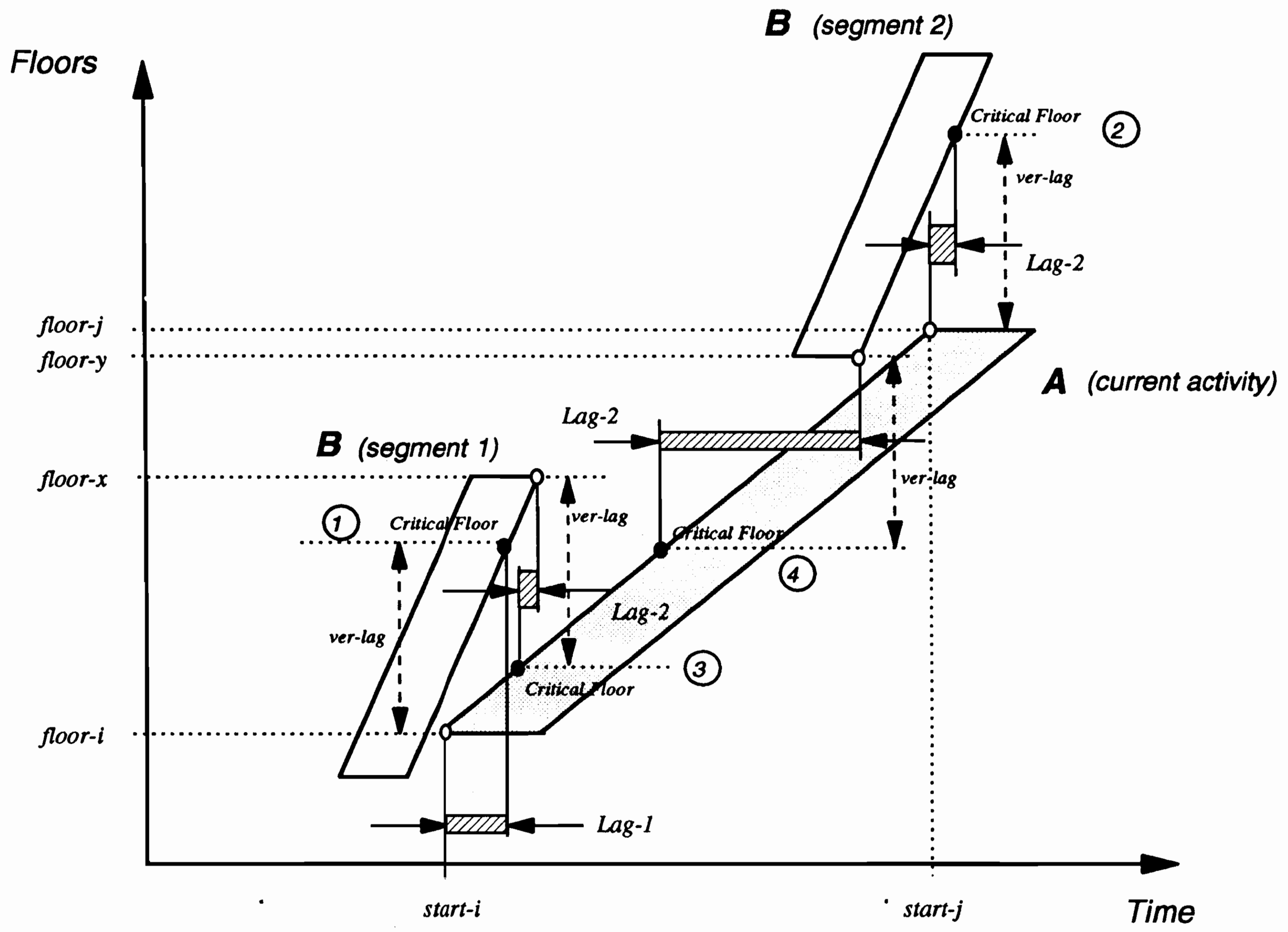


Figure C.11 - Vertical Logic Constraint Time Lags

example, with reference to figure C.11,

based on floor-x,
$(\text{Lag-2})_1$ = Scheduled finish date of B at (*floor-x*)
- preliminary start date of A at (*floor-x - ver-lag*)

based on floor-y,
$(\text{Lag-2})_2$ = Scheduled finish date of B at (*floor-y*)
- preliminary start date of A at (*floor-y - ver-lag*)

based on floor-j,
$(\text{Lag-2})_3$ = Scheduled finish date of B at (*floor-j + ver-lag*)
- preliminary start date of A at (*floor-j*)

From all lag-2 values computed, the procedure determines the maximum value,

$$lag\text{-}2_{\max} = \max\left[\, (lag\text{-}2)_1, (lag\text{-}2)_2, \ldots, (lag\text{-}2)_n \,\right]$$

Similar to horizontal constraints, the procedure also computes the duration value required to reduce the activity pace to eliminate the time lag (lag-2 values only). The required duration is computed as follows:

$$req\text{-}dur = \frac{[\text{preceding activity finish at critical floors - activity start time at first floor } (\text{floor}_i)]}{\text{Total number of floors between first floor (i) and critical floor}}$$

From all required duration values computed, the procedure determines the maximum value,

$$req\text{-}dur_{\max} = \max\left[\, (req\text{-}dur)_1, (req\text{-}dur)_2, \ldots, (req\text{-}dur)_n \,\right]$$

Scheduling Decisions

Based on the computed lag values, decisions are made to schedule or delay the activity segment. If both values (*lag-1, lag-2_{max}*) are less than or equal to zero, then the vertical constraints are satisfactory. The activity is continued to be considered for scheduling and the next processor in order (i.e. Resource Allocation Processor) is executed.

However, if one or both of these maximum values are found greater than zero, this will indicate that horizontal constraints are not satisfactory. Similar to horizontal logic constraints decisions, scheduling decisions of the vertical logic processor attempt to satisfy these constraints by considering one or both of the following issues:

1- Delay the start of activity by a value less than or equal to the lag time values (*lag-1, lag-2_{max}*) computed by the procedure.

2- Decrease activity pace by increasing its duration to the maximum required duration computed by the procedure.

Based on the computed maximum time lag values (lag-1 and lag-2_{max}) and required duration value (req-dur_{max}), the following decisions are considered by the processor:

Decision-1

Condition: lag-1 <= 0 , and lag-2_{max} <= 0

Decision: The horizontal logic constraints are satisfactory for the activity. The activity segment remains in the scheduling cycle as further constraints are verified. The attribute value of the '*Scheduling-method*' slot of activity is modified to the function name of the resource allocation processor (i.e. function: *Verify-Resource-Constraints*).

Decision-2

Condition: lag-1 > 0

Decision: The start date of activity must be delayed by the first lag value (lag-1). This decision is made regardless to the value of the second time lag computed (lag-2_{max}). The start date of the activity segment is delayed by the value of lag-1. This is the only possible way to satisfy vertical constraints at the activity start floor.

Based on this decision option, the following actions are made:

1- Modify early start (ES) value of activity by the lag-1 value computed,

set: $es = t_2 = t_1 + \text{lag-1}$

2- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time equals to the modified early start value of activity.

Decision-3

Condition: lag-1 <= 0, lag-2_{max} >0, and activity has no maximum duration value defined by user,

Decision: Because the activity has no maximum duration value, the activity duration can not be modified beyond the normal value. The start date of activity segment must be delayed by the maximum time lag-2 value computed (lag-2_{max}).

Based on this decision option, the following actions are considered:

1- Modify early start (ES) value of activity by the maximum time lag-2 value computed,

$$\text{set: } es = t_3 = t_1 + \text{lag-}2_{max}$$

2- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time equals to the modified early start value of activity.

3- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the resource allocation processor (i.e. function: *Verify-Resource -Constraints*). This action is taken because vertical logic constraints need not be re -verified when the activity is re-considered for scheduling at time t_3.

Decision-4

Condition: lag-1 <= 0, lag-2_{max} > 0, and maximum activity duration >= req-dur_{max}

Decision: To satisfy the vertical logic constraints, activity duration is modified to the maximum required value determined by the procedure while maintaining the start time of activity at the current cycle time. This decision will only be considered if the activity maximum duration value defined by user is equal to or greater than the maximum required duration (req-dur_{max}) computed by the procedure.

Based on this decision the following actions are taken:

1- Modify activity duration to the maximum required duration computed,

$$\text{set: act-dur} = \text{req-dur}_{max}$$

2- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the resource allocation processor (i.e. function: *Verify -Resource-Constraints*).

Decision-5

Condition: lag-1 <= 0, lag-2_{max} > 0, and maximum activity duration < req-dur_{max}

Decision: Satisfy the horizontal constraints by increasing the activity duration to the maximum value defined by the user and delay the start of activity by a value less than the maximum lag computed.

This decision is taken because increasing the activity duration to the user defined maximum value will not by itself suffice to satisfy the horizontal logic constraints. As a result, the activity start date must be further delayed by a value less than the maximum time lag. A new time lag value is computed using the activity maximum duration.

Based on this decision option the following actions are taken:

1- Modify activity duration to the maximum duration value defined by the user,

set: act-dur = maximum duration defined by user

2- Determine new maximum time lag-2 value for activity using new activity duration.

3- Modify early start (ES) value of activity by the new maximum time lag value computed,

set: es = es + lag-2_{max}

4- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time reaches the modified early start value of activity.

5- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the resource allocation processor (i.e. function: *Verify-Resource -Constraints*). This action is taken because vertical logic constraints need not be re

-verified when the activity is re-considered for scheduling.

Execution Function

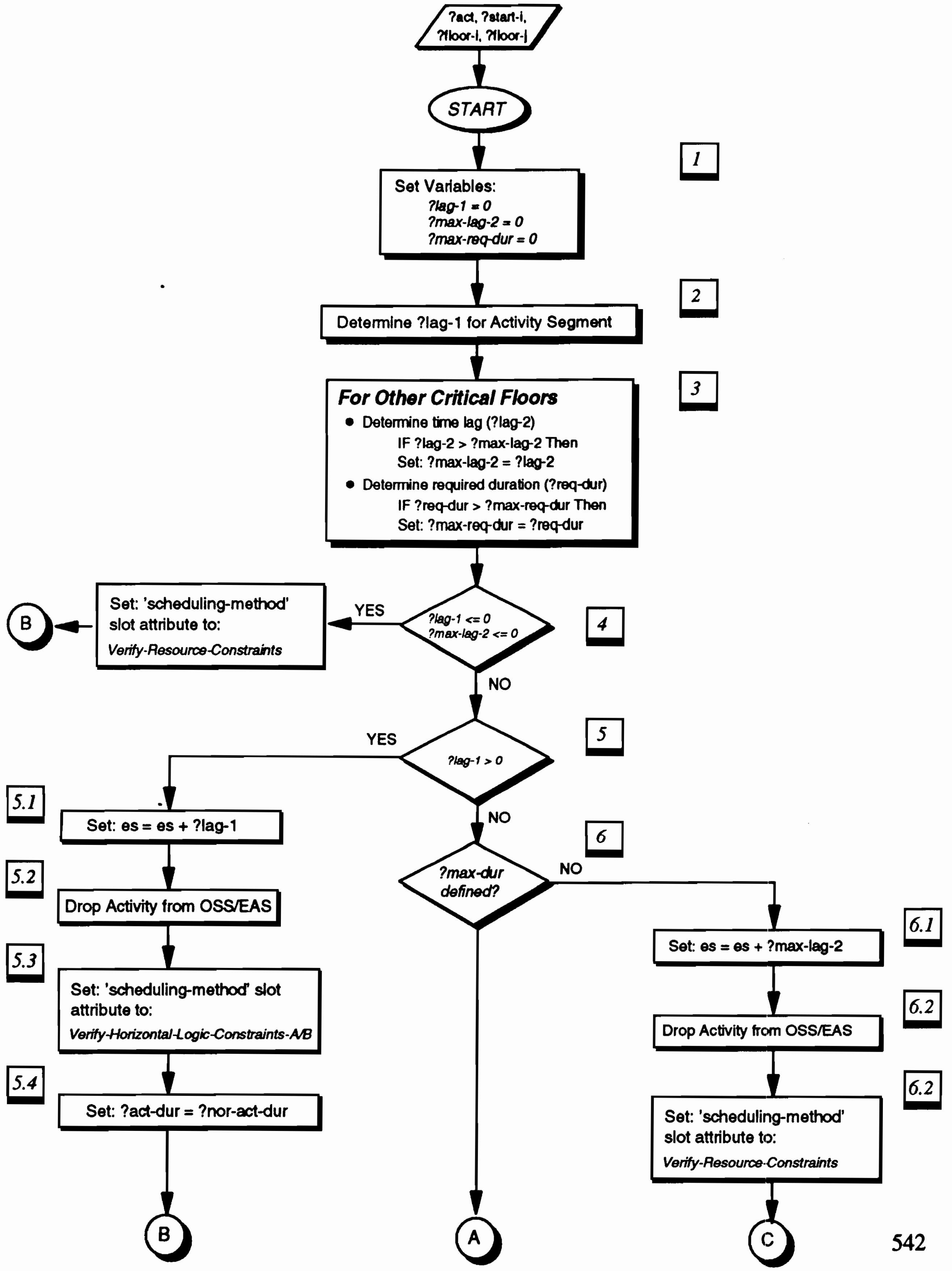
The ART-IM user defined function:

Function: *Verify-Vertical-Logic-Constraints*

is responsible for verifying the vertical logic constraints for activities. Figure C.12 depicts a flowchart of the function logic. The general format of the function is as follows:

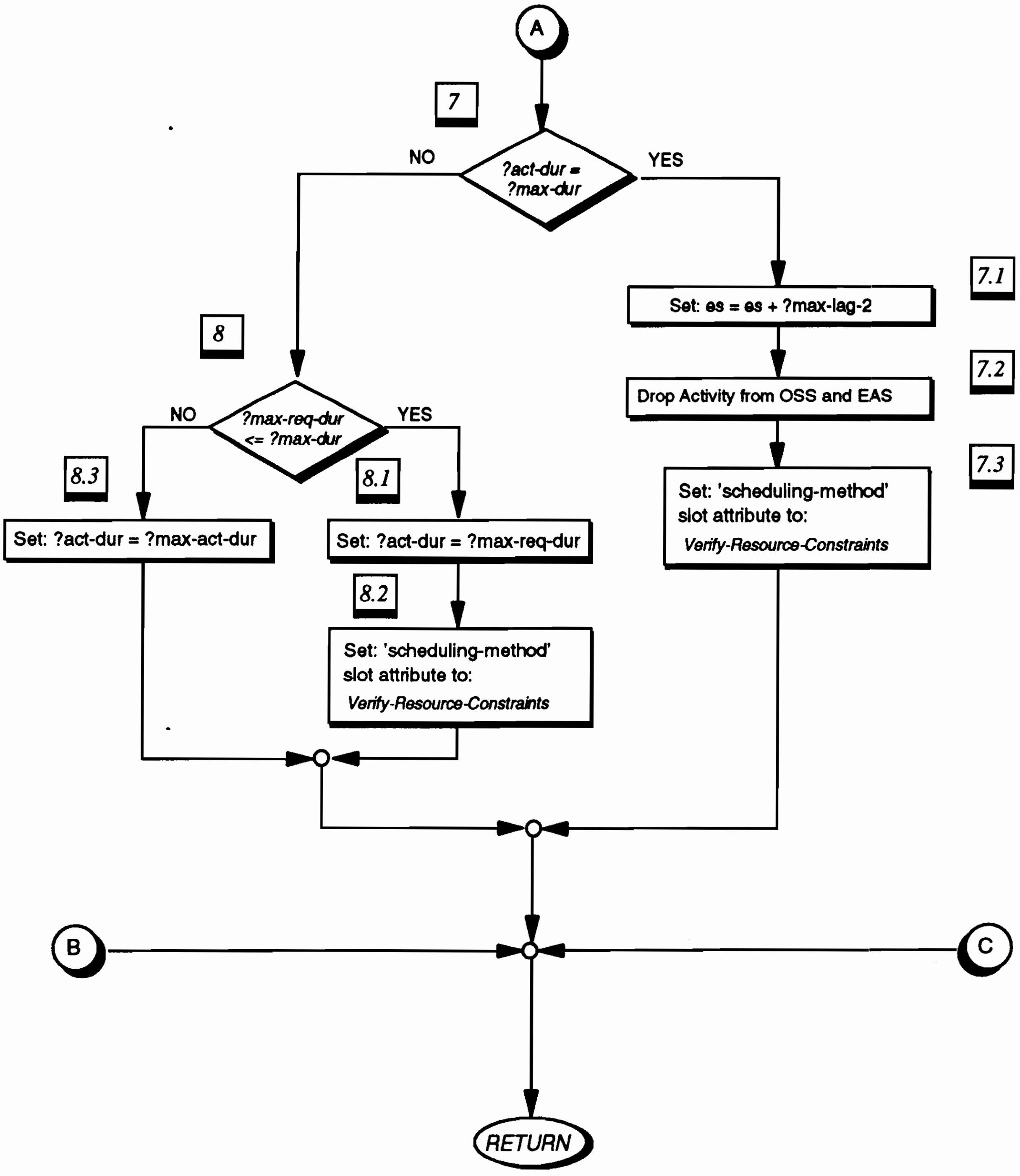
Function: *Verify-Vertical-Logic-Constraints*

```
1. Set variables :    ?lag-1 = 0,
                      ?max-lag-2 = 0,
                      ?max-req-dur = 0

2. Determine time lag value ?lag-1 for activity segment at critical floor

3. For other critical floors

    3.1 Determine time lag value ?lag-2$_i$

            IF: ?lag-2$_i$ > ?max-lag-2
            THEN: set: ?max-lag-2 = ?lag-2$_i$

    3.2 Determine required duration (?req-dur$_i$) for activity at the critical
        floor,
            IF: ?req-dur$_i$ > ?max-req-dur
            THEN: set ?max-req-dur = ?req-dur$_i$

4. IF: ?lag-1 <= 0, and ?max-lag-2 <= 0
   THEN: Modify the attribute value of the 'Scheduling-method' slot of activity to the
        function name of the vertical logic processor (i.e. function: Verify-Vertical
        -Logic).
```



?act, ?start-i, ?floor-i, ?floor-j
START
1
Set Variables:
?lag-1 = 0
?max-lag-2 = 0
?max-req-dur = 0
2
Determine ?lag-1 for Activity Segment
3
For Other Critical Floors
● Determine time lag (?lag-2)
IF ?lag-2 > ?max-lag-2 Then
Set: ?max-lag-2 = ?lag-2
● Determine required duration (?req-dur)
IF ?req-dur > ?max-req-dur Then
Set: ?max-req-dur = ?req-dur
4
?lag-1 <= 0
?max-lag-2 <= 0
YES
Set: 'scheduling-method' slot attribute to: Verify-Resource-Constraints
B
NO
5
?lag-1 > 0
YES
5.1
Set: es = es + ?lag-1
5.2
Drop Activity from OSS/EAS
5.3
Set: 'scheduling-method' slot attribute to: Verify-Horizontal-Logic-Constraints-A/B
5.4
Set: ?act-dur = ?nor-act-dur
B
NO
6
?max-dur defined?
NO
A
6.1
Set: es = es + ?max-lag-2
6.2
Drop Activity from OSS/EAS
6.2
Set: 'scheduling-method' slot attribute to: Verify-Resource-Constraints
C

Figure C.12 - Flowchart for Function: Verify-Vertical-Logic-Constraints



A
7
NO
?act-dur = ?max-dur
YES
7.1
Set: es = es + ?max-lag-2
7.2
Drop Activity from OSS and EAS
7.3
Set: 'scheduling-method' slot attribute to: Verify-Resource-Constraints
8
NO
?max-req-dur <= ?max-dur
YES
8.3
Set: ?act-dur = ?max-act-dur
8.1
Set: ?act-dur = ?max-req-dur
8.2
Set: 'scheduling-method' slot attribute to: Verify-Resource-Constraints
B
C
RETURN

Figure C.12 - Continued

5. IF: ?lag-1 > 0,
   THEN:
   5.1 Modify early start (ES) value of activity by the maximum lag value computed,

   set: es = es + ?lag-1

   5.2 Drop activity from ordered activity set (OSS) and eligible activity set (EAS).

   5.3 Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the horizontal logic processor (i.e. function: *Verify -Horizontal-Logic-Constraints-A/B*).

   5.4 Modify activity duration (?act-dur) to the normal duration value,

   set: ?act-dur = ?nor-act-dur

6. IF: ?lag-1 <= 0,
   ?max-lag-2 > 0, and
   the activity has no maximum duration value defined by the user,

   THEN:
   6.1 Modify early start (ES) value of activity by the maximum lag value computed,

   set: es = es + ?max-lag-2

   6.2 Drop activity from ordered activity set (OSS) and eligible activity set (EAS).

   6.3 Modify attribute value of the '*Scheduling-method*' slot of activity to the function name of the resource allocation processor (i.e. function: *Verify -Resource-Constraints*).

7. IF: ?lag-1 <= 0, and
   ?max-lag-2 > 0, and
   the activity has a maximum duration value defined by the user, and
   the current activity duration value (?act-dur) is already modified to the maximum value,
   THEN:
   7.1 Modify early start (ES) value of activity by the maximum lag value computed,

```
                set: es = es + ?max-lag-2

      7.2 Drop activity from ordered activity set (OSS) and eligible activity set (EAS).

      7.3 Modify attribute value of the 'Scheduling-method' slot of activity to the
          function name of the resource allocation processor (i.e. function: Verify
          -Resource-Constraints).

8. IF: ?max-lag-1 <= 0,
      ?max-lag-2 > 0,
      the activity has a maximum duration value defined by the user, and
      the current activity duration value (?act-dur) is less than the maximum duration
      value,
   THEN:
      IF: ?max-req-dur <= ?max-act-dur
      THEN:
         8.1 Modify activity duration to the maximum required duration computed
             by the procedure,

                set: ?act-dur = ?max-req-dur

         8.2 Modify the attribute value of the 'Scheduling-method' slot of activity
             to the function name of the resource allocation processor (i.e. function:
             Verify-Resource-Constraints).

      ELSE: 8.3 Modify activity duration to the maximum activity duration defined by
             the user,

                set: ?act-dur = ?max-act-dur
```

The complete source code of the function is provided in Appendix *** of this document.

C.3 The Resource Allocation Processor

Resource constraints results from the conflict between the resources that can be made available for the project and those needed to execute each activity. Every project has specific quantities of each resource that can be made available on a daily bases. This quantity is termed resource availability. On the other hand, any activity requires a certain amount of each resource in order to execute the activity. This quantity of each resource is termed resource demand. The activity can not be scheduled during any time period unless its resource demand is available in that time period.

During each scheduling cycle, the *Resource Allocation Processor* is responsible for verifying resource availability for each activity. The processor verifies that the required resource pool is available for activity to start at current time period. If available resources are satisfactory, the scheduling procedure moves to the second stage to verify space availability. However, if availability is less than the demand, the activity is delayed and dropped from the scheduling cycle. The processor computes the next possible time to reconsider activity for scheduling by determining the earliest date when the required resource or resources become available.

The general scheduling procedure of the resource allocation processor to verify resource availability for each activity is accomplished as follows:

Step-1: Modify resource pool for activity. Based on the selected duration for activity by the horizontal logic processor and vertical logic processor, the demand quantity of each resource is modified (manpower and equipment resources only).

Step-2: Define a Preliminary Scheduling Period (PSP) for activity. The preliminary period is defined by computing preliminary start and finish dates for the activity segment.

Step-3: For each resource, determine other activities using same resource during PSP. For each resource required for activity execution, the procedure searches the context for other activities using the resource and are already scheduled within the computed preliminary scheduling period (PSP).

Step-4: Determine total usage of resource during PSP. Based on all activities scheduled, the procedure first determines the usage of each resource by other activities. If the usage plus the activity demand exceeds the availability value of the resource, the activity can not be scheduled.

Step-5: Determine next possible time period to re-consider activity. If the activity can not be scheduled, the procedure determines the next possible time the activity can be re-considered for scheduling.

These steps are explained in more detail as follows:

Step-1: Modify resource pool for activity.

The resource demand pool for the activity is first modified by executing the ***Resource Modification Processor***. If initial activity normal duration defined by the user is modified, the resource modification processor determines the new demand values for each resource (manpower and equipment only). Quantities of material are not modified. The verification of resource constraints will be based on the new values determined. The resource modification processor is described in section C.5 below.

Step-2: Define a Preliminary Scheduling Period (PSP) for activity

To define a preliminary scheduling period (PSP), the start date of the first floor (floor-i) and finish date of last floor (floor-j) of activity segment is defined. The start date (start-i) of the first floor of activity segment will be equal to the current cycle time (t_i),

$$\text{Start-i} = \text{Current Cycle Time } (t_i)$$

The finish date (finish-j) of the last floor of activity segment is computed using the duration of activity segment and the total number of floors,

Finish-j = Start-i + (Segment Duration * No. of Floors)

Step-3: For each resource, determine other activities using same resource during PSP. To determine whether any scheduled activity is using the resource during the preliminary scheduling period, activities using the resource are first identified. This is accomplished by searching the attribute value of the '*act-using-res*' slot of the resource schema. As explained in chapter 8, when any activity is scheduled, the activity number is inserted as an attribute of this slot in the resource schema for each resource required by activity.

For example, if activities A and B requires resource Res-1 for their execution, then when these activities are scheduled their numbers are inserted in the 'act-using-res' slot of the res-1 schema,

```
(defschema res-1
      •
      •
  (act-using-res A B)
      •
      •
)
```

Once an activity using the resource is identified, the procedure checks whether the activity is scheduled within the preliminary scheduling period (PSP). As illustrated in Figure C.13, the scheduled activity using resource falls within the preliminary scheduling period if:

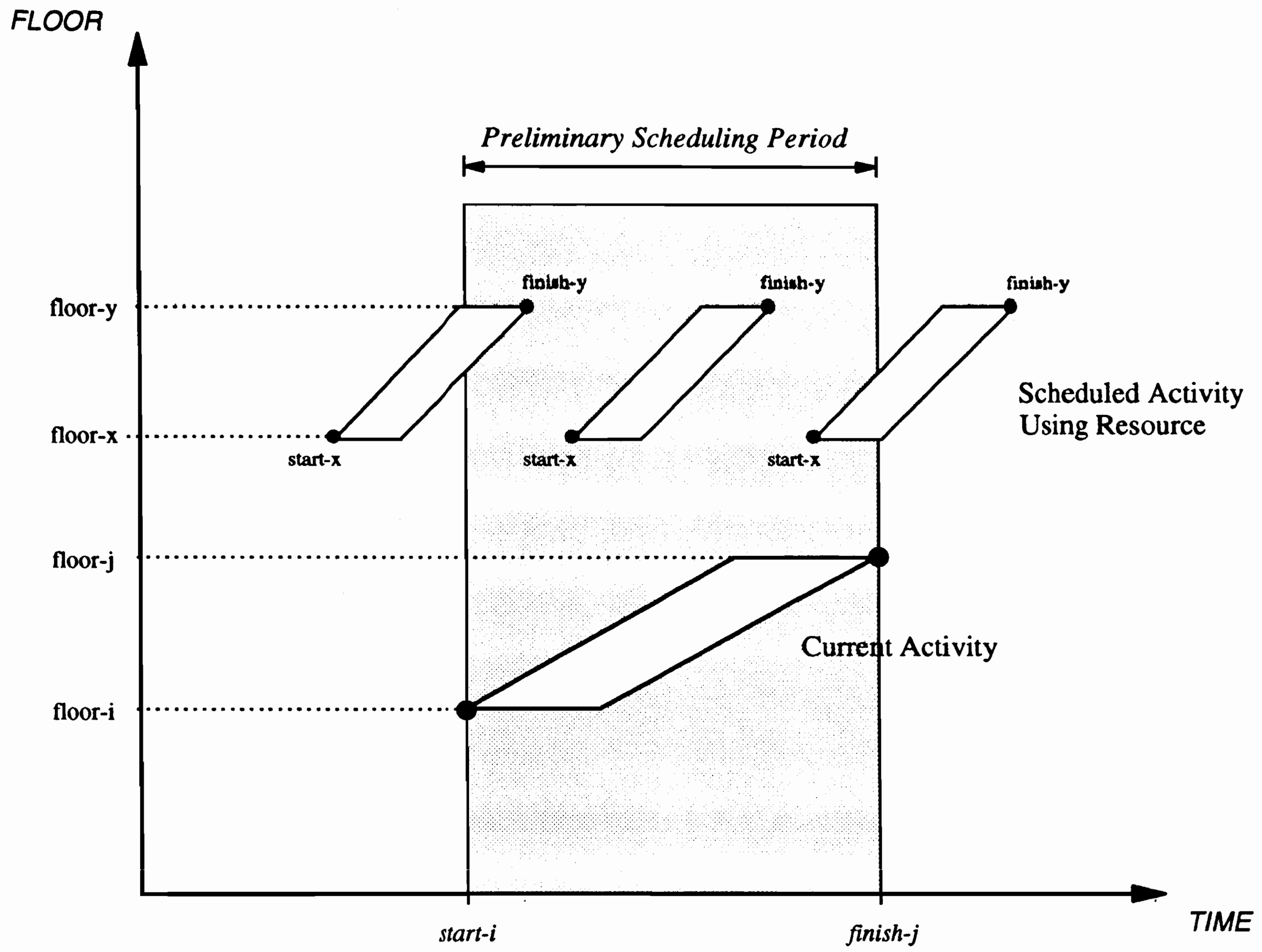


Figure C.13 - Searching Activities in Preliminary Scheduled Period (PSP)

1- The scheduled start date of its first floor (start-x) is less than the preliminary finish date (finish-j) of current activity, and

2- The scheduled finish date of its last floor (finish-y) is greater than the preliminary start date (start-i) of current activity,

start-x < finish-j, and
finish-y > start-i

Step-4: Determine total usage of resource during PSP.

Once an activity segment using the resource is determined to be scheduled during the preliminary scheduling period, its resource demand is first determined based on its scheduled duration. the resource modification processor is executed to compute the demand pool of the activity. This step is necessary because each activity may have a different duration value for each of its segments and the demand pool stored in the attribute of the "new-res-demand" slot of activity schema varies for each segment and will correspond to the last segment considered.

Once the demand pool of activity is determined, its demand quantity for the resource under consideration is added to a usage value,

$$\text{Usage} = \Sigma\ \text{Usage} + \text{Scheduled Activity Demand Value}$$

This is repeated for all activities using the resource and scheduled within the preliminary scheduling period. A total usage value is determined by adding the usage value of all scheduled activities to the demand value of the current activity, Total Usage = Usage + Current Activity Demand Value

For each resource, the total usage value is compared to the availability value of the resource defined by the user. If the total usage value is less than or equal to the availability, the resource is satisfactory. However, if the total usage value is greater than the availability the resource is not satisfactory. This is repeated for each resource of activity.

Step-5: Determine next possible time period to re-consider activity.

If one or more resources are not satisfactory, the activity can not be scheduled. The resource allocation processor determines the next earliest time the activity is re-considered for scheduling. This procedure is accomplished as follows:

1- For each resource found unsatisfactory (res_i), the procedure determines the earliest finish date ($finish\text{-}res_i$) of all activities using the resource and scheduled during the preliminary scheduling period,

$$finish\text{-}res_i = \min\left[(finish_1), (finish_2), \ldots, (finish_n)\right]$$

This value is determined for each resource (res_i) of activity.

2- From all finish values determined for all resources, the procedure determines the earliest possible time (min-es) the activity can be re-considered for scheduling. This time is equal to the maximum value of all ($finish\text{-}res_i$) values computed,

$$min\text{-}es = \max\left[(finish\text{-}res_1), (finish\text{-}res_2), \ldots, (finish\text{-}res_n)\right]$$

Determining the next possible time to re-consider an activity for scheduling results in considerable savings in memory usage and overall processing time.

Scheduling Decisions

As the total usage value is compared to the availability value for each resource, the processor makes the following decisions:

Decision-1

Condition: Total Usage <= Availability (for all resources defined for activity)

Decision: All resources defined for activity are satisfactory. The activity remains in the scheduling cycle and the scheduling procedure moves to the second stage to verify space constraints for the activity. The attribute value of the "*Scheduling-Method*" slot is modified to the function name of the space allocation processor (function: Verify-Space-Constraints).

Decision-2

Condition: Total Usage > Availability (for one or more resources defined for activity)

Decision: Because one or more resources can not be made available to the activity to start at the current period, the activity must be delayed. The activity is dropped from the scheduling cycle and the next possible time to reconsider activity for scheduling is determined.

Based on this decision the following actions are taken:

1- Modify early start (ES) value of activity to the earliest possible time to re-consider activity for scheduling,

set: es = min-es

2- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time reaches the modified early start value of activity (min-es).

3- Based on the continuity class of activity, modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the horizontal logic processor (i.e. function: *Verify-Horizontal-Logic-Constraints-A* or *Verify-Horizontal-Logic-Constraints-B*). This allows to re-verify the horizontal logic constraints when the activity is re-considered for scheduling.

4- Modify the attribute value of the '*act-dur*' slot of activity to the normal duration value. The purpose is to initialize the activity duration to the normal value defined. This allows the scheduling procedure verify the horizontal logic constraints with the normal duration value when the activity is re-considered for scheduling.

set: act-dur = normal activity duration

Execution Function

The resource allocation processor consists of one ART-IM user defined function:

Function: *Verify-Resource-Constraints*

A flowchart of the function logic is depicted in Figure C.14. The general format of the function is as follows:

Function: *Verify-Resource-Constraints*

1. Determine preliminary scheduling period. Compute start and finish dates of first and last floor of activity segment (start-i, and finish-j).

2. Set variables:

   ?flag = Yes
   ?act-min-es = 0

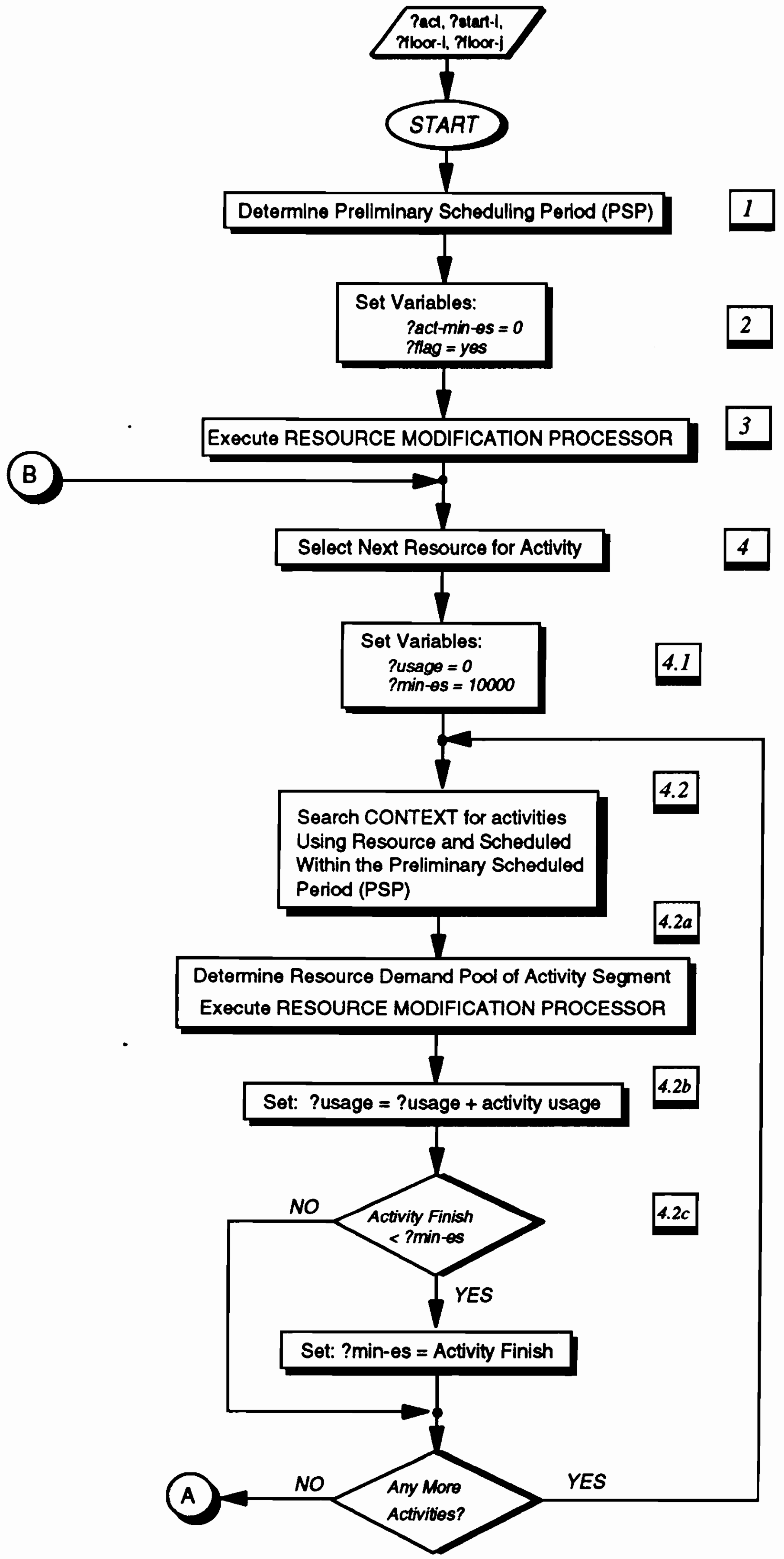

?act, ?start-l, ?floor-l, ?floor-j
START
Determine Preliminary Scheduling Period (PSP)
1
Set Variables: ?act-min-es = 0 ?flag = yes
2
Execute RESOURCE MODIFICATION PROCESSOR
3
B
Select Next Resource for Activity
4
Set Variables: ?usage = 0 ?min-es = 10000
4.1
Search CONTEXT for activities Using Resource and Scheduled Within the Preliminary Scheduled Period (PSP)
4.2
4.2a
Determine Resource Demand Pool of Activity Segment
Execute RESOURCE MODIFICATION PROCESSOR
Set: ?usage = ?usage + activity usage
4.2b
NO
Activity Finish < ?min-es
4.2c
YES
Set: ?min-es = Activity Finish
A
NO
Any More Activities?
YES

Figure C.14 - Flowchart for Function: Verify-Resource-Constraints

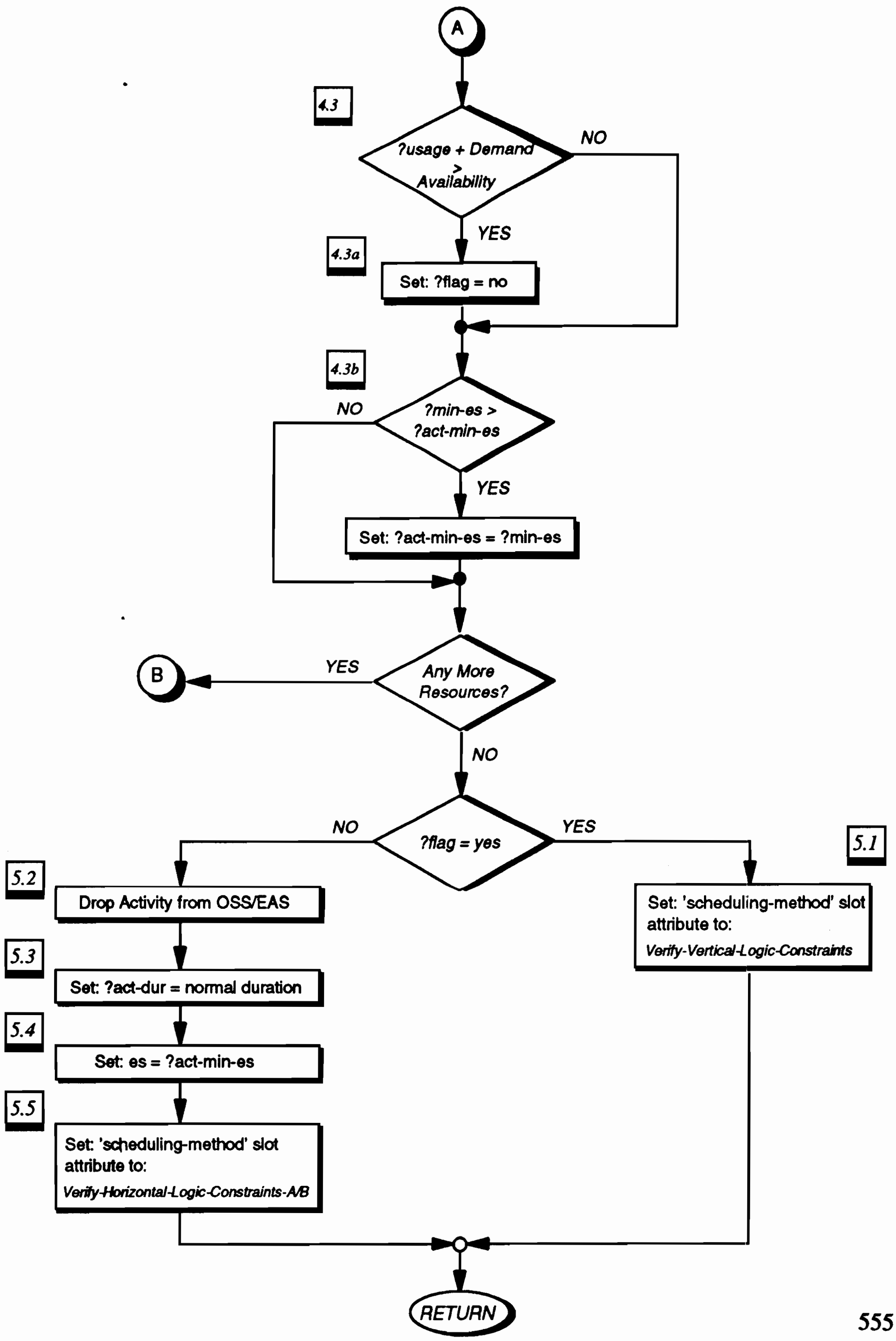

A
4.3
?usage + Demand > Availability
NO
YES
4.3a
Set: ?flag = no
4.3b
NO
?min-es > ?act-min-es
YES
Set: ?act-min-es = ?min-es
B
YES
Any More Resources?
NO
NO
?flag = yes
YES
5.1
Set: 'scheduling-method' slot attribute to: Verify-Vertical-Logic-Constraints
5.2
Drop Activity from OSS/EAS
5.3
Set: ?act-dur = normal duration
5.4
Set: es = ?act-min-es
5.5
Set: 'scheduling-method' slot attribute to: Verify-Horizontal-Logic-Constraints-A/B
RETURN

Figure C.14 - Continued

```
3.  Execute the Resource Modification Processor to modify the activity resource
    demand pool.

4.  For each resource code defined for the activity in the "res-demand' slot, Do:
      4.1 Set variables:
                          ?usage = 0
                          ?min-es = 10000 (big value)

      4.2 Search context for activities using the resource and are already scheduled
          within the preliminary scheduling period. For each activity determined, do

            4.2a Execute resource modification processor to determine activity
                 demand pool based on its segment scheduled duration.
            4.2b Set: ?Usage = ?Usage + ?res-usage
            4.2c IF: activity finish < ?min-es
                   THEN: set: ?min-es = activity finish

      4.3 IF: ?usage + activity demand > availability value of resource
          THEN:
                4.3a    set: ?flag = no
                4.3b    IF: ?min-es > ?act-min-es
                        THEN: set ?act-min-es = ?min-es

5.  IF: ?flag = yes
    THEN:
          5.1 modify attribute value of 'scheduling-method' slot to function name of
          space
             allocation processor.
    ELSE:
          5.2 Drop activity from OSS and EAS
          5.3 Modify attribute value of 'act-dur' slot to the normal duration value.
          5.4 Set: es = ?act-min-es
          5.5 Modify attribute value of 'scheduling-method' slot to the function
          name of the horizontal logic processor.
```

The complete source code of the function is provided in Appendix G.

C.4 The Space Allocation Processor

Space constraints results from the conflict between the space demand of all activities scheduled concurrently in the same work area or work block, and the total available space of that block.

During each scheduling cycle, the *Space Allocation Processor* is responsible for verifying space availability for each activity. The processor verifies that the space demand of the activity plus the space demand of the other concurrent activities, already scheduled in the same work block(s) of the activity, do not exceed the total work space availability of the block.

The general scheduling procedure of the space allocation processor to verify space availability is as follows:

Step-1 Determine other concurrent activities scheduled in the same work block(s). For each floor of activity segment, the procedure checks each block of activity and determines all other activities already scheduled in the same block.

Step-2 Identify space demand class of activity and concurrent activities sharing block. The procedure identifies activity space demand class (A, B, or C) defined by the user. The procedure also identifies the demand class of the other concurrent activities sharing the activity work block(s) as determined from step-1.

Step-3 If class A is identified, delay activity beyond the concurrent time period. If the current activity or one or more concurrent activity sharing the block(s) has a class A space demand, the activity can not be scheduled during the concurrent time period at the specific floor or floors. The procedure attempts to delay the activity beyond this period. This is accomplished by either increasing the duration of activity segment to reduce its pace, or by delaying the start of the entire activity segment to a following time period.

Step-4 If class B or C space demand is identified, verify space availability of activity. If neither the activity nor the concurrent activities have a demand class A, the procedure compares space demand of the activity plus the demand of the concurrent activities with the total available space of work block and attempt to schedule activity utilizing the different space-based scheduling decisions discussed in Chapter7.

Step-5 If no other activities are sharing the work block(s), schedule activity. If no other activities are concurrently scheduled in the activity work block(s), the activity is scheduled.

The scheduling procedure responsible for these steps is defined by the ART-IM user defined primary function: *Verify-Work-Space-Constraints*. The function is executed by the controller module when the first scheduling stage is completed. The five scheduling steps performed by the function are explained in more detail as follows:

STEP-1: Determine other concurrent activities scheduled in the same work block(s)

For each floor of activity segment, the procedure determines all other concurrent activities already scheduled in the same block(s) designated to activity. The procedure verifies the manpower/equipment block and the material block of the activity. If a concurrent scheduled activity is allocated to one or both blocks, the activity number is placed in the attribute of the "*act-sharing-space*" slot of the activity schema.

STEP-2: Identify space demand class of activity and concurrent activities sharing block

The procedure first identifies the space demand of the current activity. If the demand class is A, the procedure moves to step-3. If the current activity has a demand class B or C, he procedure identifies the space demand class of all concurrent activities defined in step 1 and defined in the attribute of the "act-sharing-space" slot of activity schema. If one or more concurrent activities has a space demand class A, the procedure moves to step-3.

If the activity and all concurrent activities sharing the work block(s) do not have a class A space demand, the procedure moves to step-4.

•

STEP-3: If class A is identified, delay activity beyond the concurrent time period

If the activity or one or more concurrent activities sharing work block(s) have an A space demand class, the activity can not be scheduled during the concurrent period. The procedure attempts to delay the activity beyond the current period at the specific floor or floors.

This is accomplished by first computing time lag values between the activity and each concurrent activity. Two types of lag values are computed. The first type (lag-1) is determined between the start date of the first floor of activity segment and the finish date of the corresponding floor of the concurrent activity sharing the work block(s). This is illustrated in Figure C.15.

The second type of lag values (lag-2) are determined at other floors defined by the last floor of the current activity segment and the start/finish floors of each concurrent activity segment. As illustrated in Figure C.16, based on the pace of the current activity relative to the. concurrent activity, three case scenarios define values of lag-2. These case scenarios are as follows:

1- Case-1: The first lag-2 value is determined between the finish date of the last floor (floor-j) of current activity A and the finish date of corresponding floor of the concurrent activity B (Figure C.16a).

2- Case-2: The second lag-2 value is determined between the finish date of the first floor (floor-x) of concurrent activity B and the start date of corresponding floor of the current activity A (Figure C.16b).

3- Case-3: The third lag-2 value is determined between the finish date of the last floor (floor-y) of concurrent activity B and the start date of corresponding floor of the current activity A (Figure C.16c).

The secondary ART-IM user defined functions: *Workspace-Lag-1* and *WorkSpace-Lag-2* are responsible for computing the lag-1 and lag-2 values respectively. The functions are called from within the primary function *Verify-Work-Space-Constraints* by using the function calls:

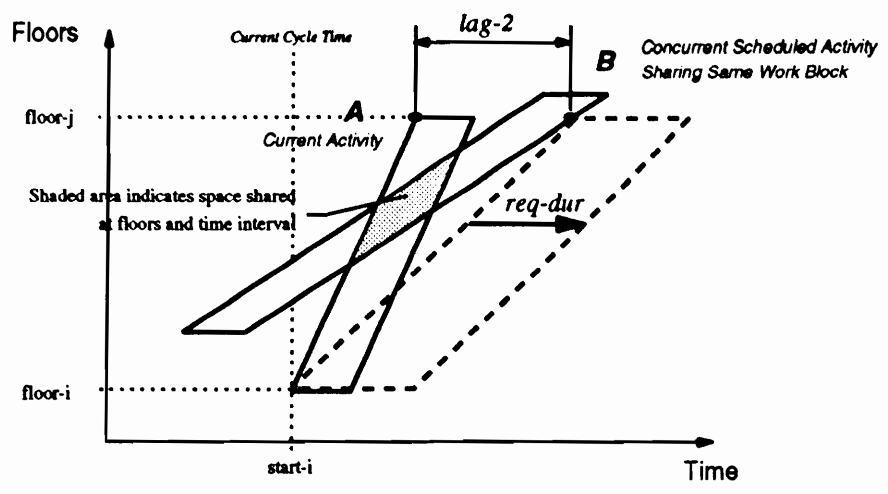
(Workspace-Lag-1 ?act ?act-dur ?floor-i ?floor-j ?act-sharing-space),

and

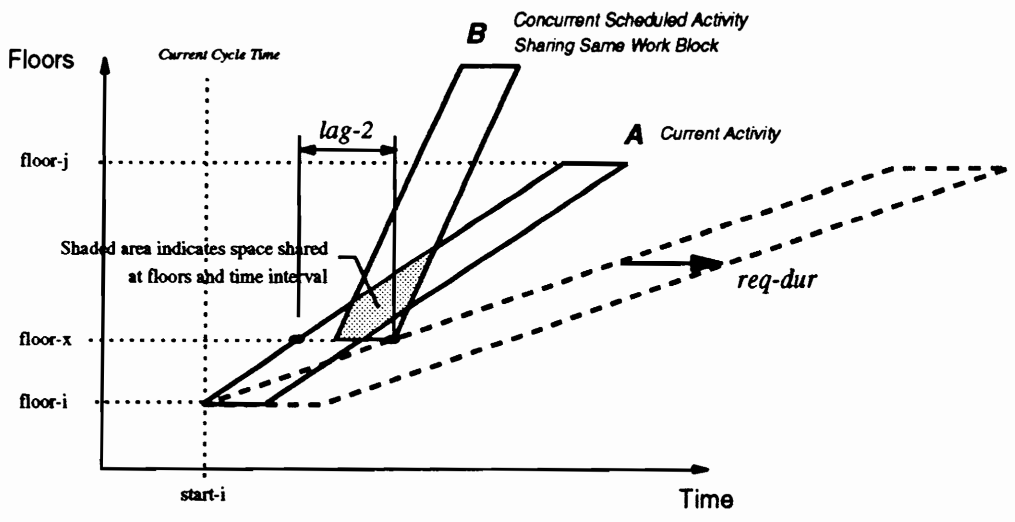
(Workspace-Lag-2 ?act ?act-dur ?floor-i ?floor-j ?act-sharing-space)

where,

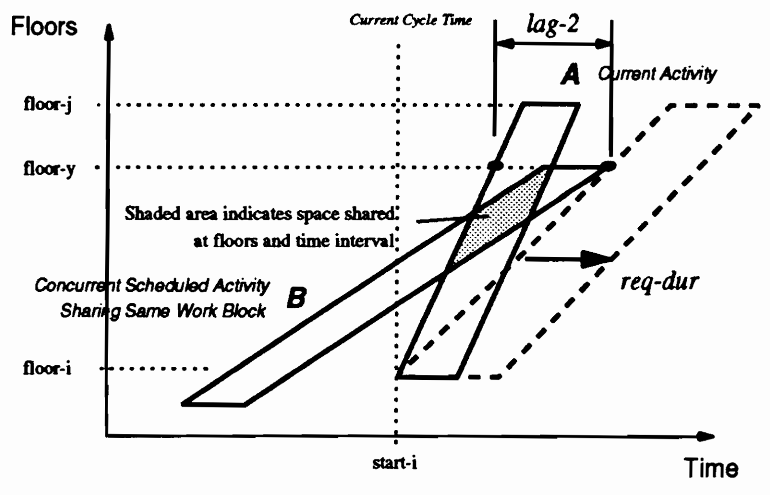
?act	: is a variable that denotes the current activity number,
?act-dur	: is a variable that denotes the selected duration for the activity segment,
?floor-i	: is a variable that denotes the number of the first floor of activity segment,
?floor-j	: is a variable that denotes the number of the last floor of activity segment,
?act-sharing-space	: is a variable that denotes the number of concurrent activity sharing space.



a) Case Scenario #1



b) Case Scenario #2



c) Case Scenario #3

Figure C.16 - Case Scenarios for LAG-2

Once the function is executed, control is returned to the primary function.

From all time lag values computed between the current activity and all concurrent activities sharing the work block(s), two maximum values are determined

$$lag\text{-}1_{max} = \max\left[\ (lag\text{-}1)_1,\ (lag\text{-}1)_2,\ \ldots,\ (lag\text{-}1)_n\ \right], \text{ and}$$

$$lag\text{-}2_{max} = \max\left[\ (lag\text{-}2)_1,\ (lag\text{-}2)_2,\ \ldots,\ (lag\text{-}2)_n\ \right]$$

In addition to computing maximum values of lag-1 and lag-2, the procedure also computes the duration value (req-dur) required to reduce the activity pace in order eliminate the time lags (lag-2 only) and schedule activity beyond the concurrent time period (shown hatched in Figure C.15). As illustrated in Figure C.16, the required duration is computed for each case and is equal to:

$$req\text{-}dur = \frac{[\text{concurrent activity finish at specific floor - activity start time at first floor}_i\ ]}{\text{Total number of floors}}$$

Required duration values are computed for all cases of Figure C.16d. From all duration values computed, a maximum value is determined,

$$req\text{-}dur_{max} = \max\left[\ (req\text{-}dur)_1,\ (req\text{-}dur)_2,\ \ldots,\ (req\text{-}dur)_n\ \right]$$

Based on the determined maximum lag values (lag-1_{max} and lag-2_{max}) and the maximum required duration value (req-max_{max}), the procedure makes the following decisions:

Decision-1
Condition: lag-$1_{max} > 0$

Decision: The start date of activity segment is delayed by the maximum time lag value (lag-1_{max}). This decision is made regardless to the value of lag-2_{max} computed. Based on this decision option, the following actions are made:

1- Modify early start (ES) value of activity by the lag-1_{max} value computed,

set: es = es + lag-1_{max}

2- Modify activity segment duration to the normal duration value defined by user,

set: act-dur = normal duration

3- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time equals to the modified early start value of activity.

4- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the horizontal logic processor (i.e. function: *Verify-Horizontal-Logic -Constraints-A/B*).

Decision-2
Condition: lag-$1_{max} <= 0$, and
lag-$2_{max} >0$, and
activity has no maximum duration value defined by user.

Decision: Because the activity has no maximum duration value, the activity duration can not be modified beyond the normal value. The start date of activity segment must be delayed by the value of lag-2_{max} computed.

Based on this decision, the following actions are considered:

1- Modify early start (ES) value of activity by the maximum time lag-2 value computed,

set: $es = es + lag\text{-}2_{max}$

2- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time equals to the modified early start value of activity (t_3).

3- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the vertical logic processor (i.e. function: *Verify-Vertical-Logic-Constraints*). The horizontal logic processor need not be executed when the activity is re-considered for scheduling. This step allows to start the vertical logic processor as the first processor to be executed.

Decision-3

Condition: $lag\text{-}1_{max} <= 0$, and
$lag\text{-}2_{max} > 0$, and
$req\text{-}dur_{max} <=$ maximum activity duration defined by user

Decision: Modify activity duration to the maximum required value determined by the procedure while maintaining the start time of activity segment at the current cycle time. This decision is only considered if the maximum required duration ($req\text{-}dur_{max}$) computed by the procedure is less than or equal to the activity maximum duration value defined by user.

Based on this decision the procedure modifies the activity duration to the maximum required duration computed,

set: $act\text{-}dur = req\text{-}dur_{max}$

Decision-4

Condition: $\text{lag-1}_{max} <= 0$, and
$\text{lag-2}_{max} > 0$, and
$\text{req-dur}_{max} >$ maximum activity duration

Decision: Modify the activity segment duration to the maximum activity duration value defined by the user and delay the start of activity by a value less than the maximum lag computed. This decision is taken because increasing the activity duration to the user defined maximum value will not by itself suffice to delay the activity beyond the concurrent time period. As a result, the activity start date must be further delayed by a new value less than the maximum value computed. The new time lag value is computed using the activity maximum duration.

Based on this decision option the following actions are taken:

1- Modify activity duration to the maximum duration value defined by the user,

set: act-dur = maximum duration defined by user

2- Determine new maximum time lag-2 value for activity using new activity duration.

3- Modify early start (ES) value of activity by the new maximum time lag value computed,

set: $es = es + \text{lag-2}_{max}$

4- Remove activity from current scheduling cycle by dropping the activity number from the ordered scheduled set (OSS) and the eligible activity set (EAS). The activity is reconsidered for scheduling when the cycle time reaches the modified early start value of activity (t_4).

5- Modify activity duration to the normal duration value defined by the user,

set: act-dur = normal duration defined by user

6- Modify the attribute value of the '*Scheduling-method*' slot of activity to the function name of the horizontal logic processor (i.e. function: *Verify-Horizontal-Logic -Constraints*). The horizontal logic processor need not be executed when the activity is re-considered for scheduling at t_4. This step will allow to start the vertical logic processor as the first processor to be executed.

These decisions are executed by a third secondary function: *Workspace-Modify*. The function is executed from within the primary function *Verify-Work-Space-Constraints* by using the function call:

(Workspace-Modify ?act ?act-dur ?floor-i ?floor-j ?max-lag-1 ?max-lag-2)

where

?act	: is a variable that denotes the activity number under consideration,
?act-dur	: is a variable that denotes the selected duration for the activity segment,
?floor-i	: is a variable that denotes the number of the first floor of activity segment,
?floor-j	: is a variable that denotes the number of the second floor of activity segment,
?max-lag-1	: is a variable that denotes the value of the first lag type,
?max-lag-2	: is a variable that denotes the value of the second lag type

Once the function is executed, control is returned to the primary function.

STEP-4: If class B or C space demand is identified, verify space availability of activity

If neither the current activity nor any of the concurrent activities sharing activity work block(s) has an A space demand class, The procedure verifies space demand requirements for all activities sharing the work block(s). The procedure compares space demand of the current activity plus the total space demand of the concurrent activities with total space availability of the activity's work block(s) to decide whether to schedule the current activity or delay it to a following time period. This is accomplished as follows:

4-1 Determine space demand of activity SD-1 (manpower and equipment) and SD-2 (material). The procedure computes the demand values using equations 5(a) and 5(b) defined in Chapter 7,

$$\text{SD-1 } (\textit{Activity}) = \Sigma \text{ Quantity x } (S_{physical} + S_{surrounding}) \quad \text{------(5a)}$$

$$\text{SD-2 } (\textit{Activity}) = [\, \Sigma \text{ (Quantity x } S_{physical}) \,] + S_{surrounding} \quad \text{------(5b)}$$

4-2 For each floor of activity segment, the procedure determines the space demand values sharing the activity work block(s). Demand values of each concurrent activity is determined using same equations,

$$\text{SD-1 } (\textit{Concurrent Activity}) = \Sigma \text{ Quantity x } (S_{physical} + S_{surrounding}) \quad \text{------(5a)}$$

$$\text{SD-2 } (\textit{Concurrent Activity}) = [\, \Sigma \text{ (Quantity x } S_{physical}) \,] + S_{surrounding} \quad \text{------(5b)}$$

If the concurrent activity has a class C space demand, and the concurrent time period is beyond 2/3 of its duration, then half the value of SD-2 occupied by its material is considered. This is based on the step function proposed in Chapter 7 for this class of activities.

4-3 For each floor, determine total usage value of space in each work block of activity. The space demand values of each concurrent activity are added to usage value for each block. There are two usage values defined, *usage-1* for manpower and equipment block, and *usage-2* for material block. Based on the work block of the concurrent activity, its space demand values are added to the appropriate usage value of the activity. That is,

Usage-1 = Σ Usage-1 + SD-1 and/or SD-2 (Concurrent Activity)

Usage-2 = Σ Usage-2 + SD-1 and/or SD-2 (Concurrent Activity)

4-4 For each concurrent activity, the procedure determines a floor lag value (floor-lag) between the start time of the activity and finish time of concurrent activity at the current floor of activity segment.

floor-lag = Finish time of concurrent activity at the floor - Start time of activity at current floor

The procedure also computes a duration value required to move activity beyond the concurrent time period to eliminate the lag. This value is computed at every floor except the start floor of activity.

$$\text{req-dur} = \frac{[\text{concurrent activity finish at current floor - activity start time at its first floor}]}{\text{Total number of floors (current - first floor)}}$$

Steps 4-2 through 4-4 are repeated for all concurrent activities sharing the work block(s) with the current activity. From all activities considered, the procedure determines the minimum value of all floor lag values computed (floor-lag$_{min}$). The procedure also determines the maximum value of all required durations computed (req-dur_{max}).

Once all concurrent activities are considered, the procedure moves to step 4-5.

4-5 The procedure compares space demand value of activity material (SD-2) plus space usage in material block (Usage-2) with the space availability of material block. If availability is satisfactory, the procedure moves to step 4-6.

If availability is less than total demand (SD-2 + Usage-2), the procedure allocates the minimum floor lag value (floor-lag_{min}) to the material block lag (block2-lag-1 or block2-lag-2),

```
IF:          floor              =     1st floor of activity, and
             floor-lag_min      >     block2-lag-1
THEN: set:   block2-lag-1       =     floor-lag_min

IF:          floor              <>    1st floor of activity, and
             floor-lag_min      >     block2-lag-2 (material), and
THEN: set:   block2-lag-2       =     floor-lag_min
```

4-6 The procedure compares space demand vale of activity manpower and equipment (SD-1) plus total space usage in manpower/equipment block (Usage-1) with space availability of block. If availability is satisfactory, space constraints for the current floor of activity segment are satisfactory.

If availability is less than the total demand (SD-1 + Usage-1), the procedure performs the following:

1- Allocate the minimum floor lag value (floor-lag$_{min}$) to the manpower/equipment block lag (block1-lag-1 or block1-lag-2)

IF:		floor	=	1st floor of activity, and
		floor-lag$_{min}$	>	block1-lag-1
THEN:	set:	block1-lag-1	=	floor-lag$_{min}$

IF:		floor	<>	1st floor of activity, and
		floor-lag$_{min}$	>	block1-lag-2
THEN:	set:	block1-lag-2	=	floor-lag$_{min}$

2- Compute a SCF for the current floor. Based on the SCF value computed, the procedure determines a modified duration value using the Productivity-SCF curve. The procedure determines an average value (ave-mod-dur) for all the floors of the activity segment.

At this point, if the current floor is not the last floor of activity segment, the procedure moves to the next floor and repeats steps 4-2 through 4-6. Otherwise, the procedure moves to step 4-7.

4-7 From all iterative steps of 4-2 through 4-6 along all floors of activity segment, the following values are determined by the procedure,

- Two maximum lag values in manpower/equipment work block determined in step 4-6:

  block1-lag-1$_{max}$, and *block1-lag-1*$_{max}$

- Two maximum lag values in material work block determined in step 4-5:

block2-lag-1$_{\max}$, and *block2-lag-1*$_{\max}$

- An average modified duration value based on an average value of SCF determined in step 4-6:
  mod-act-dur
- A maximum required duration value to reduce the pace of activity determined in step 4-4:
  req-dur$_{\max}$

Based on these values, the procedure makes the following decisions:

Decision-1
Condition: block2-lag-1$_{\max}$ > 0, or
block2-lag-2$_{\max}$ > 0

Decision: The activity must be delayed beyond the concurrent time period. This is accomplished by either decreasing activity duration by the maximum required duration value computed (req-dur$_{\max}$), or by delaying the start of the entire activity segment to a following time period. Decisions 1 through 4 discussed in main step-3 above are applied here.

Using values of **block2-lag-1**$_{\max}$ and **block2-lag-2**$_{\max}$, this is accomplished by executing the secondary function: *Workspace-Modify* using the ART-IM function call:

(Workspace-Modify ?act ?act-dur ?block2-lag-1 ?block2-lag-2)

Decision-2
Condition: block2-lag-1$_{\max}$ <= 0, and
block2-lag-2$_{\max}$ <= 0, and
mod-act-dur <= maximum value defined by Productivity-SCF curve

Decision: The activity is scheduled during the current time period using the modified duration value computed. The procedure modifies the attribute value of the '*Scheduling-method*' slot of activity to the function name of the schedule processor (i.e. function: *Schedule-A or Schedule-B).*

```
.       For continuity class A activities:
                set:  attribute of slot = "Schedule-A"

        For continuity class B activities:
                set:  attribute of slot = "Schedule-B"
```

Decision-3

Condition: block2-lag-1_{max} <= 0, and
block2-lag-2_{max} <= 0, and
mod-act-dur > maximum value defined by Productivity-SCF curve, and
block1-lag-1_{max} <= 0, and
block1-lag-2_{max} <= 0

Decision: The activity must be delayed beyond the concurrent time period. This is accomplished by either decreasing activity duration by the maximum required duration value computed (req-dur_{max}), or by delaying the start of the entire activity segment to a following time period. Decisions 1 through 4 discussed in main step-3 above are applied here.

Using values of **block1-lag-1_{max}** and **block1-lag-2_{max}**, this is accomplished by executing the secondary function: *Workspace-Modify* using the ART-IM function call:

(Workspace-Modify ?act ?act-dur ?floor-i ?floor-j ?block1-lag-1 ?block1-lag-2)

STEP-5: If no other activities are sharing the work block(s), schedule activity

If no other activities are determined to be concurrently sharing the same work block(s) with the current activity, the activity is scheduled regardless to the space demand class of activity. The procedure modifies the attribute value of the '*Scheduling-method*' slot of activity to the function

name of the schedule processor (i.e. function: *Schedule-A or Schedule-B).*

```
For continuity class A activities:
        set:  attribute of slot = "Schedule-A"

For continuity class B activities:
        set:  attribute of slot = "Schedule-B"
```

Execution Functions

The space allocation processor consists of four ART-IM user defined functions:

Function: *Verify-Work-Space-Constraints*	(primary function)
Function: *Workspace-Lag-1*	(secondary function)
Function: *Workspace-lag-2*	(secondary function)
Function: *Workspace-Modify*	(secondary function)

The *Verify-Work-Space-Constraints* function is the main function of the space allocation processor. The function is responsible for the overall scheduling procedure of the processor. The other secondary functions are executed from within this function. Figure C.17 depicts a flowchart of the function logic. The general format of the function is as

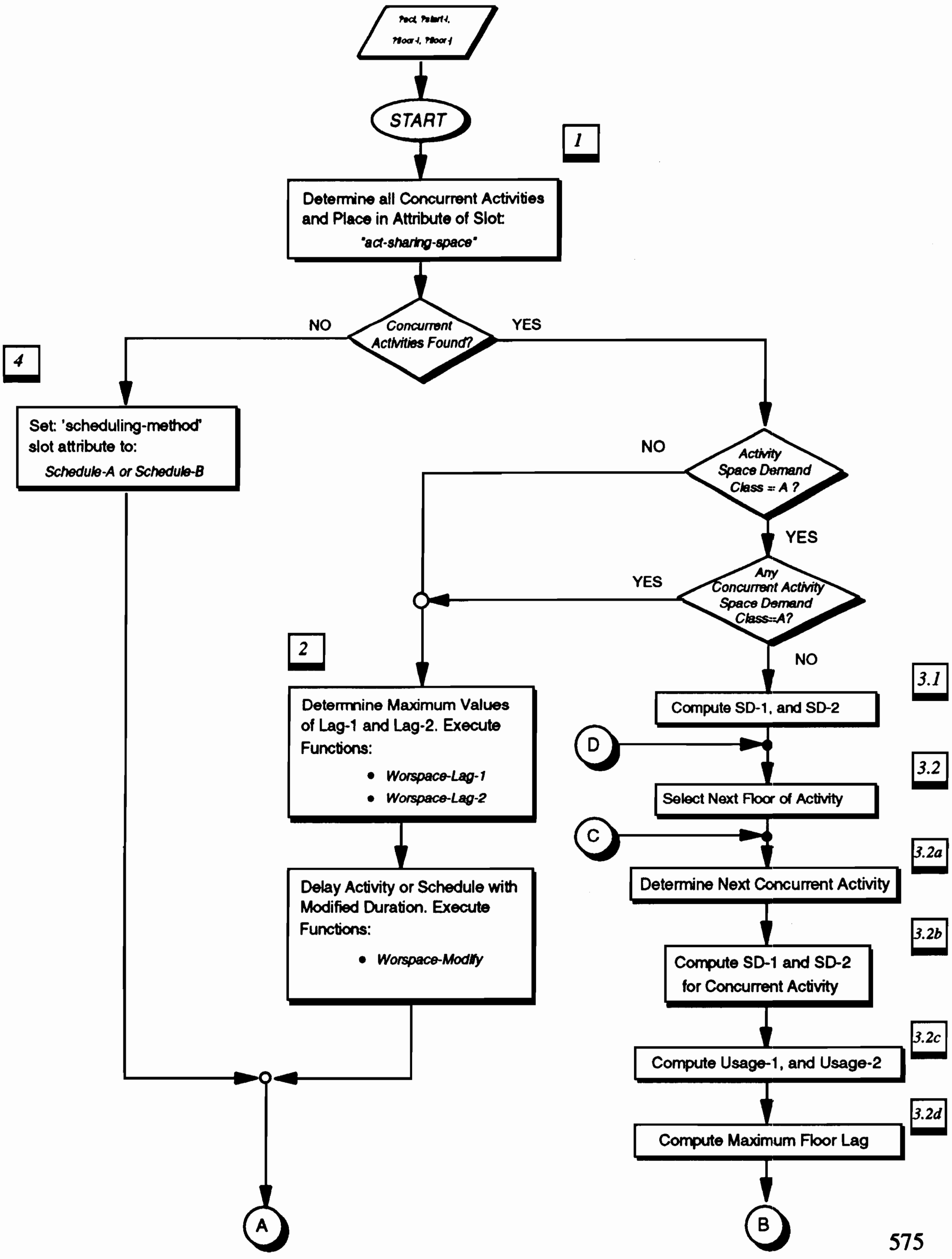

START
1
Determine all Concurrent Activities and Place in Attribute of Slot: 'act-sharing-space'
Concurrent Activities Found?
NO
YES
4
Set: 'scheduling-method' slot attribute to: Schedule-A or Schedule-B
Activity Space Demand Class = A ?
NO
YES
Any Concurrent Activity Space Demand Class=A?
YES
NO
2
Determnine Maximum Values of Lag-1 and Lag-2. Execute Functions:
• Worspace-Lag-1
• Worspace-Lag-2
Delay Activity or Schedule with Modified Duration. Execute Functions:
• Worspace-Modify
3.1
Compute SD-1, and SD-2
D
3.2
Select Next Floor of Activity
C
3.2a
Determine Next Concurrent Activity
3.2b
Compute SD-1 and SD-2 for Concurrent Activity
3.2c
Compute Usage-1, and Usage-2
3.2d
Compute Maximum Floor Lag
A
B

Figure C.17 - Flowchart for Function: Verify-Work-Space-Constraints

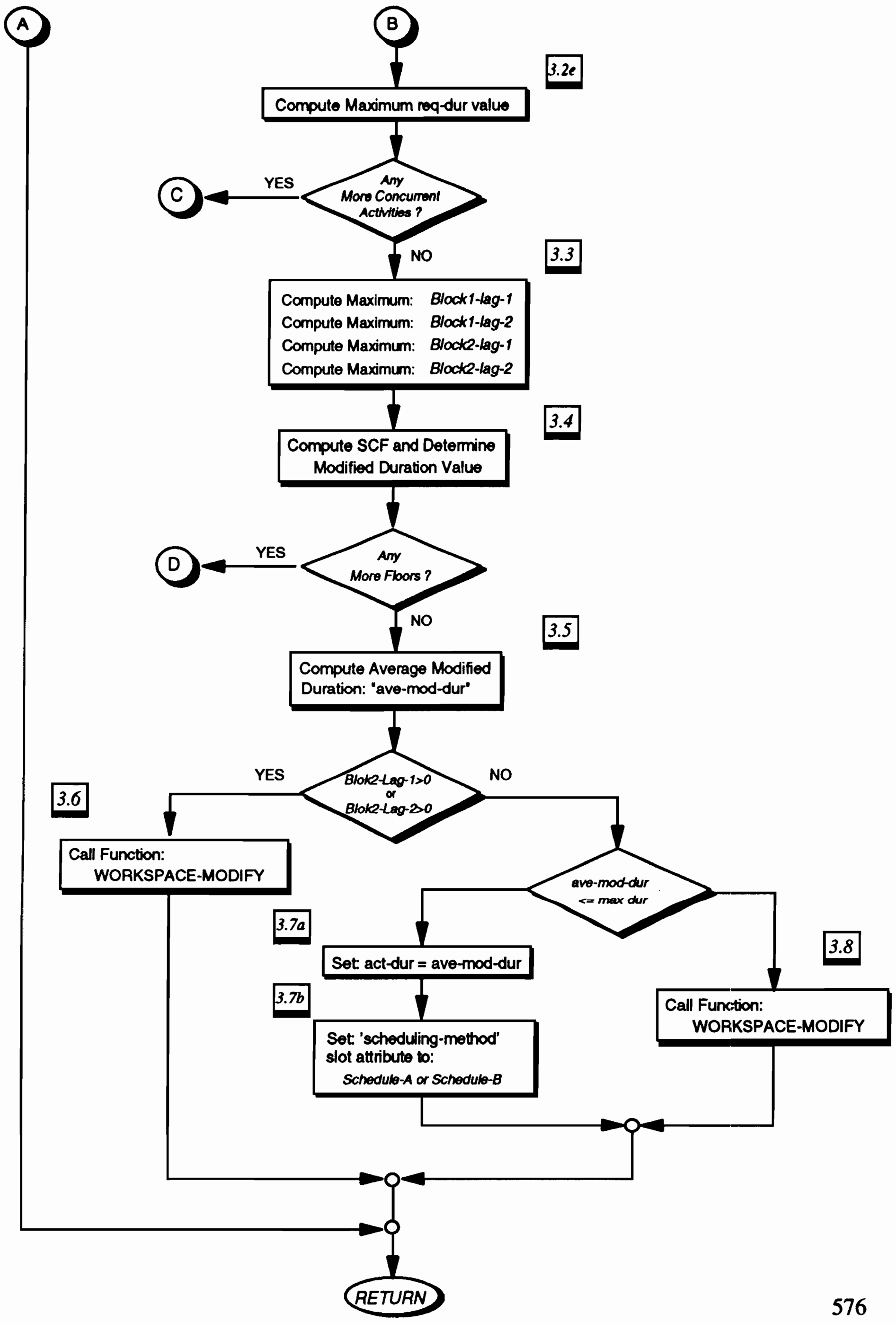

A
B
3.2e
Compute Maximum req-dur value
YES
C
Any More Concurrent Activities ?
NO
3.3
Compute Maximum: Block1-lag-1
Compute Maximum: Block1-lag-2
Compute Maximum: Block2-lag-1
Compute Maximum: Block2-lag-2
3.4
Compute SCF and Determine Modified Duration Value
YES
D
Any More Floors ?
NO
3.5
Compute Average Modified Duration: "ave-mod-dur"
YES
Blok2-Lag-1>0 or Blok2-Lag-2>0
NO
3.6
Call Function: WORKSPACE-MODIFY
ave-mod-dur <= max dur
3.7a
Set act-dur = ave-mod-dur
3.7b
Set 'scheduling-method' slot attribute to: Schedule-A or Schedule-B
3.8
Call Function: WORKSPACE-MODIFY
RETURN

Figure C.17 - Continued

follows:

Function: ***Verify-Work-Space-Constraints***

```
1.  For each floor of activity segment, determine other scheduled concurrent activity
    sharing activity work block(s). Place concurrent activity number in attribute of the
    "act-sharing-space" slot of activity schema.

2.  IF:     Concurrent activities exist, and
            (Activity has a class "A" space demand, or
             any concurrent activity has a class "A" space demand),
    THEN:  Execute secondary functions:

                    Workspace-lag-1
                    Workspace-lag-2
                    Workspace-Modify

3.  IF:     Concurrent activities exist, and
            Activity space demand class <> "A", and
            Concurrent activities space demand class <> "A"
    THEN:
            3.1 Compute activity space demand values SD-1, and SD-2
            3.2 For each floor of activity segment:
                    3.2a Determine concurrent activity sharing work block with activity
                    3.2b Determine space demand values SD-1 and SD-2 for
                    concurrent activity
                    3.2c Determine block usage values (Usage-1, Usage-2)
                    3.2d Compute maximum value of floor-lag
                    3.2e Compute maximum value of req-dur

            3.3 Compute: max block1-lag-1, max block1-lag-2, max block2-lag-2, and
                max
                        block2-lag-2

            3.4 Compute space capacity factor (SCF) and determine modified duration
                for current floor using Productivity-SCF curve.

            3.5 Determine average modified duration (ave-mod-dur)

            3.6 IF: Block2-lag-1 > 0, or Block2-lag-2 > 0,
                THEN: Execute secondary function: Workspace-Modify

            3.7 IF: Block2-lag-1 <= 0, and Block2-lag-2 <= 0, and
            ave-mod-dur <= maximum duration allowed by Productivity-SCF
            curve
        THEN:
                    3.7a  Set: activity duration value = mod-act-dur
                    3.7b  Modify attribute value of "scheduling method" slot of
                          activity to function name of the schedule processor
                          (Schedule-A or Schedule-B)

    3.8  IF: Block2-lag-1 <= 0, and Block2-lag-2 <= 0, and
             ave-mod-dur > maximum duration allowed by Productivity-SCF
             curve
         THEN: Execute secondary function: Workspace-Modify.

IF: No concurrent activities exists,
THEN:
        Modify attribute value of "scheduling method" slot of activity to function
        name of the schedule processor (Schedule-A or Schedule-B).
```

The second function of the space allocation processor is the: *Workspace-Lag-1* secondary function. The function is responsible for computing values of lag-1 for any activity. The function is executed from within the primary function during step-3. Figure C.18 depicts a flowchart of the function logic. The general format of the function is as follows:

Function: *Workspace-Lag-1*

```
1.  Select next segment of scheduled concurrent activity

2.  IF: Current floor falls within the concurrent activity segment
    THEN: Compute value of lag-1
```

The third of the space allocation processor is the: *Workspace-Lag-2* secondary function. The function is responsible for computing values of lag-2 for any activity. The function is executed from within the primary function during step-3. Figure C.19 depicts a

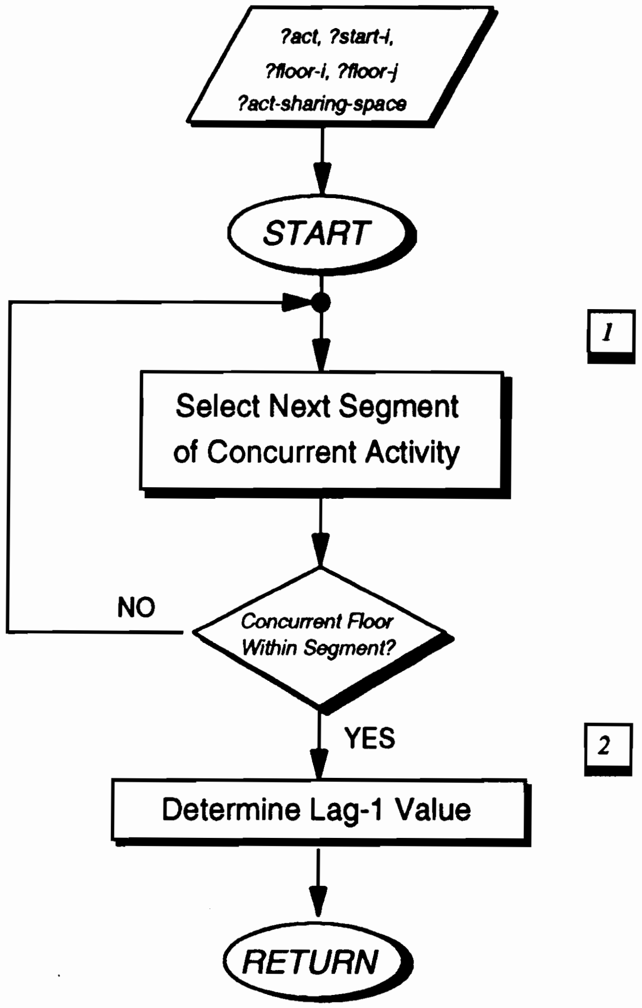


Figure C.18 - Flowchart for Function: Workspace-Lag-1

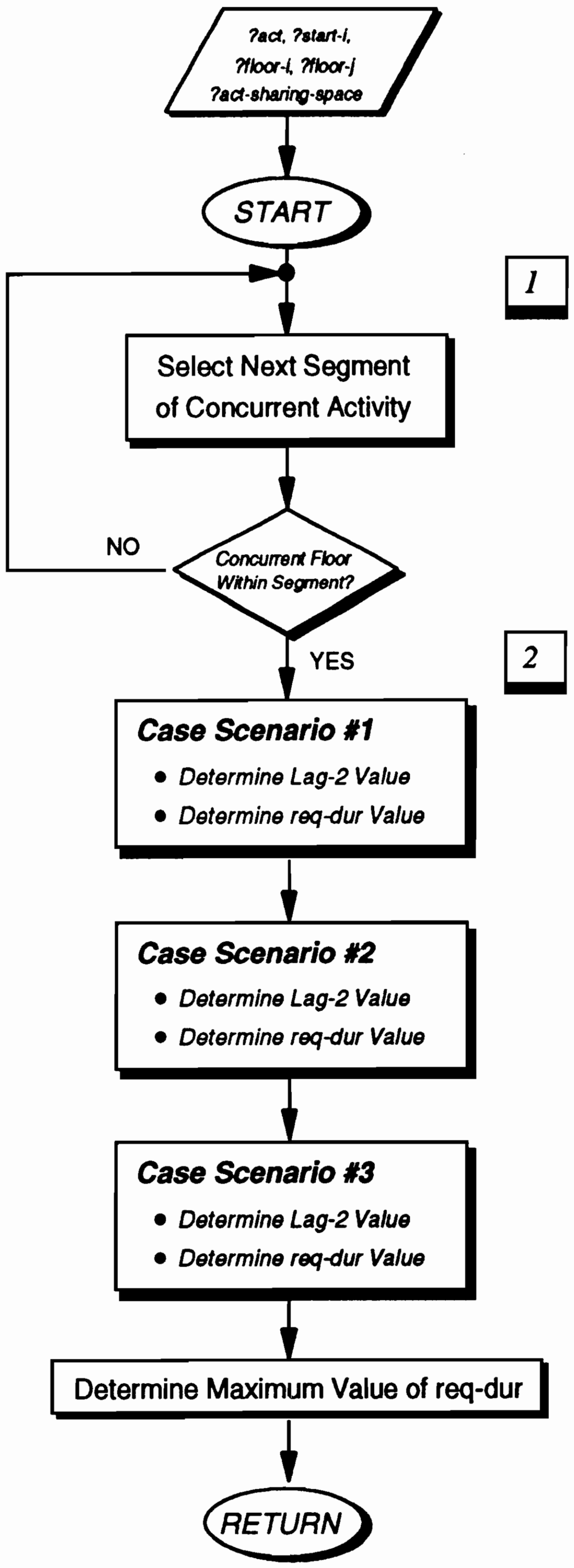


Figure C.19 - Flowchart for Function: Workspace-Lag-2

flowchart of the function logic. The general format of the function is as follows:

Function: *Workspace-Lag-2*

```
1. Select next segment of scheduled concurrent activity

2. IF: current floor falls within the concurrent activity segment
   THEN:
        2.1 Compute value of lag-2 and req-dur for case scenario #1
        2.2 Compute value of lag-2 and req-dur for case scenario #2
        2.3 Compute value of lag-2 and req-dur for case scenario #3
        2.4 Determine maximum value of all req-dur values computed
```

The forth of the space allocation processor is the: *Workspace-Modify* secondary function. The function is responsible for moving scheduled activity beyond concurrent time with other activity sharing the block by either modifying activity duration or delaying activity by lag values defined in function call. The function is executed from within the primary function during step-3 and step-4. Figure C.20 depicts a flowchart of the function logic. The general format of the function is as follows:

Function: *Workspace-Modify*

```
1. IF:     ?max-lag-1 > 0,
   THEN:
        1.1 Modify early start (ES) value of activity by the maximum lag value
            computed,

                    set:  es = es + ?max-lag-1

        1.2 Drop activity from ordered activity set (OSS) and eligible activity set
            (EAS).

        1.3 Modify attribute value of the 'Scheduling-method' slot of activity to the
            function name of the horizontal logic processor (i.e. function: Verify
            -Horizontal-Logic-Constraints-A/B).
```

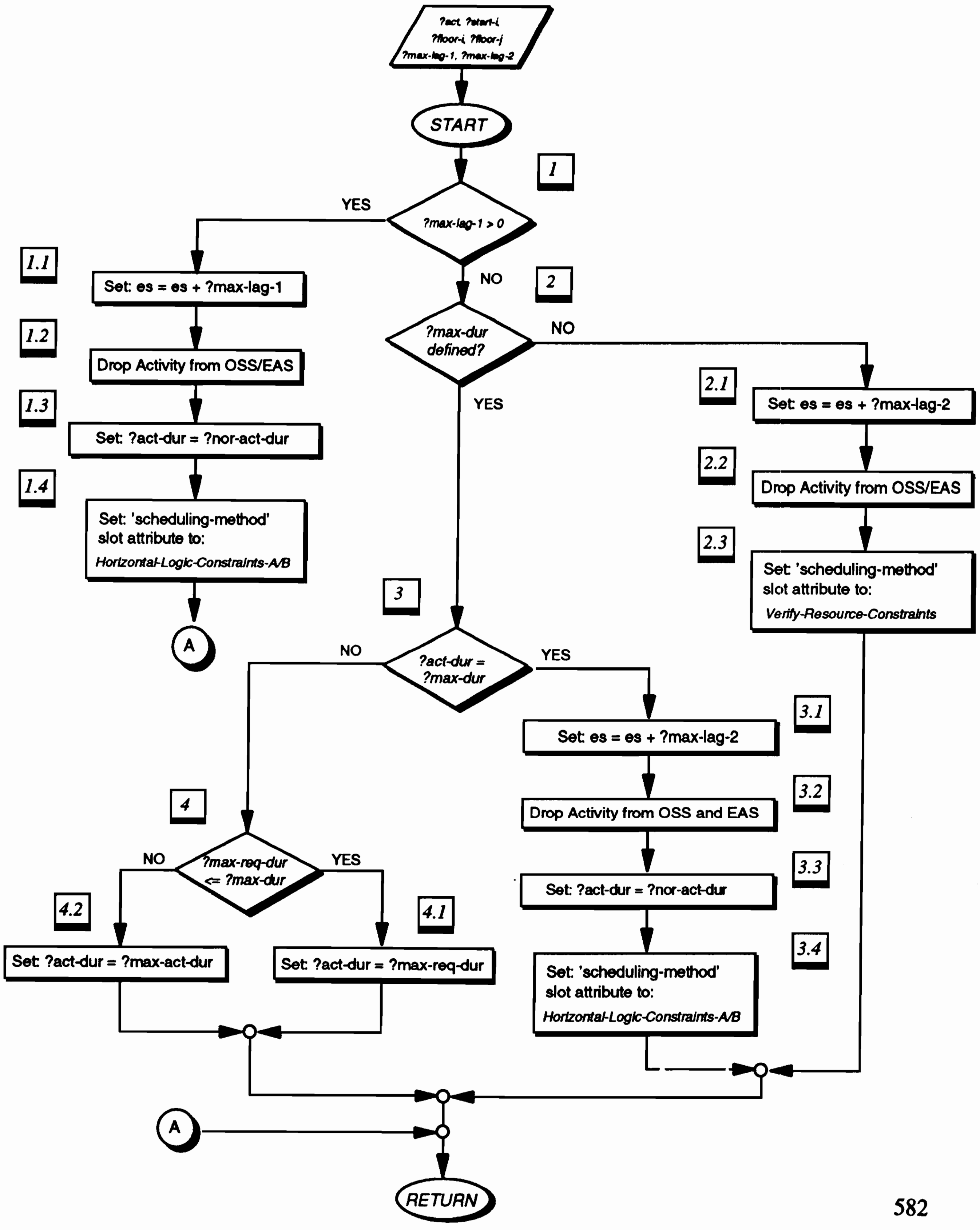

?act, ?start-i,
?floor-i, ?floor-j
?max-lag-1, ?max-lag-2
START
1
?max-lag-1 > 0
YES
1.1
Set: es = es + ?max-lag-1
1.2
Drop Activity from OSS/EAS
1.3
Set: ?act-dur = ?nor-act-dur
1.4
Set: 'scheduling-method' slot attribute to: Horizontal-Logic-Constraints-A/B
A
NO
2
?max-dur defined?
NO
2.1
Set: es = es + ?max-lag-2
2.2
Drop Activity from OSS/EAS
2.3
Set: 'scheduling-method' slot attribute to: Verify-Resource-Constraints
YES
3
?act-dur = ?max-dur
NO
YES
3.1
Set: es = es + ?max-lag-2
3.2
Drop Activity from OSS and EAS
3.3
Set: ?act-dur = ?nor-act-dur
3.4
Set: 'scheduling-method' slot attribute to: Horizontal-Logic-Constraints-A/B
4
?max-req-dur <= ?max-dur
NO
YES
4.2
Set: ?act-dur = ?max-act-dur
4.1
Set: ?act-dur = ?max-req-dur
A
RETURN

Figure C.20 - Flowchart for Function: Workspace-Modify

```
    1.4 Modify activity duration (?act-dur) to the normal duration value,

            set:  ?act-dur = ?nor-act-dur

2.  IF:     ?max-lag-1 <= 0, and
            ?max-lag-2 > 0, and
            the activity has no maximum duration value defined by the user,
    THEN:
            2.1 Modify early start (ES) value of activity by the maximum lag value
            computed,

                    set: es = es + ?max-lag-2

            2.2 Drop activity from ordered activity set (OSS) and eligible activity set
            (EAS).

            2.3 Modify attribute value of the 'Scheduling-method' slot of activity to
            the function name of the resource allocation processor (i.e. function: Verify
            -Resource-Constraints).

3.  IF:     ?max-lag-1 <= 0, and
            ?max-lag-2 > 0, and
            the activity has a maximum duration value defined by the user, and
            the current activity duration (?act-dur) is already modified to the maximum
            limit,
    THEN:
            3.1 Modify early start (ES) value of activity by the maximum lag value
            computed,

                    set:  es = es + ?max-lag-2

            3.2 Drop activity from ordered activity set (OSS) and eligible activity set
            (EAS).

            3.3 Modify activity duration (?act-dur) to the normal duration value,

                    set:  ?act-dur = ?nor-act-dur

4.  IF:     ?max-lag-1 <= 0, and
            ?max-lag-2 > 0, and
            the activity has a maximum duration value defined by the user, and
            the current activity duration value (?act-dur) is less than the maximum
            duration value,
THEN:
        IF: ?max-req-dur <= ?max-act-dur
        THEN:
                4.1 Modify activity duration to the maximum required duration
                computed by the procedure,

                                set: ?act-dur = ?max-req-dur

        ELSE: 4.2 Modify activity duration to the maximum activity duration
                defined by the user,

                                set: ?act-dur = ?max-act-dur
```

The complete source code of all functions of the space allocation processor are provided in Appendix G.

C.5 The Resource Modification Processor:

The resource demand pool for any activity is defined by the user during data input. The resource magnitude values for each resource are defined based on the activity normal duration. However, during the scheduling procedure, scheduling decisions made may modify activity normal duration in an attempt to satisfy specific constraints. As was discussed in sections C.1, C.2 and C.4 above, activity pace may be slowed down by increasing activity duration (up to the maximum value) in order to satisfy the horizontal, vertical and space constraints. Slowing the activity pace and modifying its normal duration requires adjusting, if possible, its initial quantity values defined by the user. Only quantity values of manpower and equipment are modified. Defined quantity values for materials are not affected by the production rate of activity and are not changed.

The procedure adopted to adjust quantity values of manpower and equipment for each activity uses a simple algorithm to compute a resource ratio between the new duration value selected and the initial normal duration defined by the user. The resource ratio is expressed as follows:

$$\text{Resource ratio} = \frac{\text{Selected new duration}}{\text{Defined normal duration}}$$

The modified quantity of each resource is then computed by multiplying the resource ratio by the initial resource quantity,

$$\text{Modified resource quantity} = \text{Resource ratio} * \text{Initial quantity}$$

Modified quantity values are truncated towards positive infinity (e.g. a 3.6 labor is computed as 4.0).

Realistic modification of resource quantities as a result of duration change requires availability of resource demand pools for different ranges of productivity rates. Once a duration is selected by the scheduling procedure, the productivity range can be identified and the new quantity value of the specific resource can be determined. Different activity resource demand pools for different ranges of productivity rates can be defined during data input.

This more exact approach to determine new resource quantities is not considered in this research work and requires future modification to the database program and the scheduling procedure. The objective here is to recognize the realistic necessity of modifying resource pools as activity duration is changed and to allow for a step, during the scheduling process, to perform such modification.

During each scheduling cycle, the *Resource Modification Processor* is responsible for adjusting the manpower and equipment quantity values. The processor is executed from within two other processors during each scheduling cycle in order to modify the resource demand pool for each activity being scheduled. This is depicted in Figure C.21. During the first scheduling stage, the processor is executed from within the resource allocation processor to modify the resource pool during the verification process of resource availability constraints. During the second stage, the processor is executed from within the space allocation processor to modify the resource pool before computing space demand values for concurrent activities.

The processor is executed using the ART-IM function call:

(Modify-Resource-demand-Pool ?act ?act-dur)

where,

modify-resource-demand-pool	: is the function name of the processor,
?act	: is a variable that denotes the activity number under consideration,
?act-dur	: is a variable that denotes the selected duration for the activity segment,

Given a selected duration for any activity segment, the processor modifies quantity values

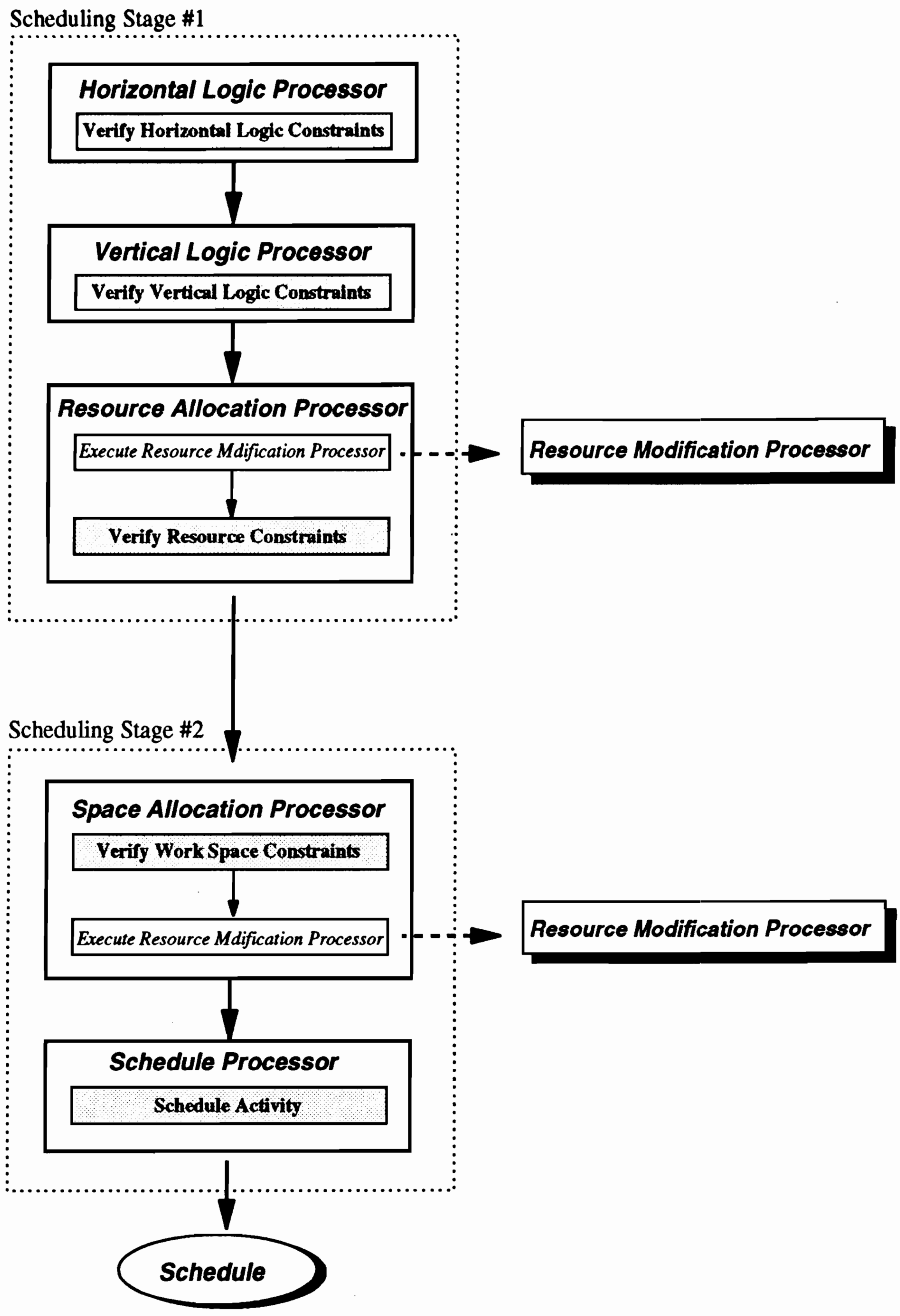

Scheduling Stage #1
Horizontal Logic Processor
Verify Horizontal Logic Constraints
Vertical Logic Processor
Verify Vertical Logic Constraints
Resource Allocation Processor
Execute Resource Mdification Processor
Verify Resource Constraints
Resource Modification Processor
Scheduling Stage #2
Space Allocation Processor
Verify Work Space Constraints
Execute Resource Mdification Processor
Resource Modification Processor
Schedule Processor
Schedule Activity
Schedule

Figure C.21 - Control of Execution of the Resource Moification Processor

of manpower/equipment only. The processor stores the modified quantity values for manpower/equipment and initial quantity values of material are stored in the attribute of the '*new-res-demand*' slot of each activity.

Execution Function

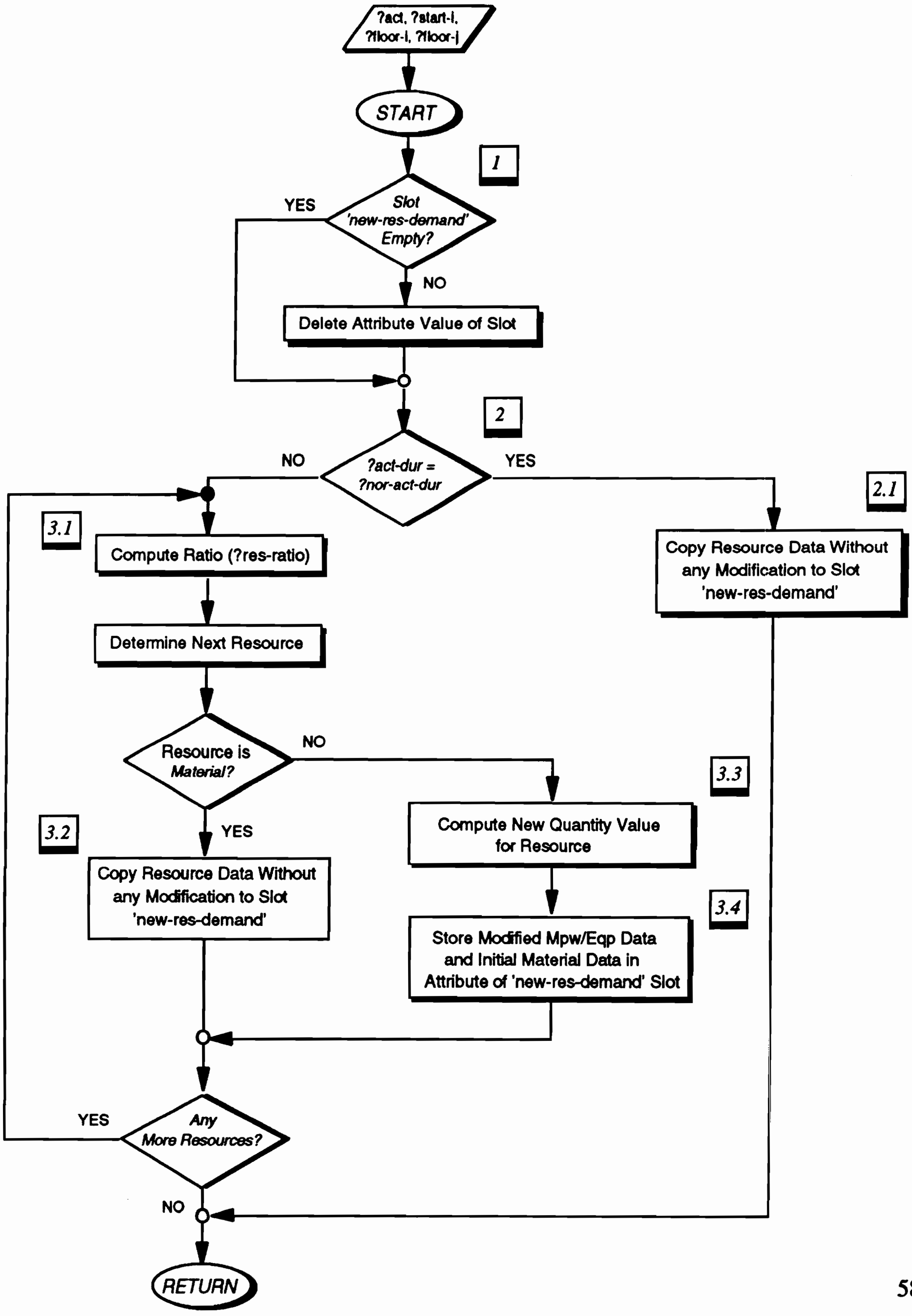
The ART-IM user defined function:

Function: *Modify-Resource-Demand-Pool*

constitutes the resource modification processor. A flowchart of the function logic is depicted in Figure C.22. The general format of the function is as follows:

Function: *Modify-Resource-Demand-Pool*

```
1. Retract any previous values defined in the attribute of the "new-res-demand' that may
   be defined during previous scheduling attempts. If this is the first time to execute this
   function, the slot will be empty and this step will not be considered.

2. IF: Normal activity duration is not modified (i.e. act-dur = nor-act-dur)
   THEN:
         2.1  Initial quantity values of activity manpower and equipment are not
              modified. Copy the attribute value of the 'res-demand' slot to the 'new-res-
              demand' slot without any changes.

3. IF: Normal activity duration is modified,
   THEN:
         3.1  Compute resource ratio, and
              For each resource defined for activity, Do
                     IF: Resource is of type material
                     THEN: 3.2 Copy resource data (code, type, quantity) into attribute
                               of 'new-res-demand' slot without any change.
                     ELSE: 3.3 Determine new quantity value of resource based on
                               computed resource ratio.
                           3.4 Copy resource data (code, type, and new quantity) into
                               attribute of 'new-res-demand' slot.
```



?act, ?start-i, ?floor-i, ?floor-j
START
1
Slot 'new-res-demand' Empty?
YES
NO
Delete Attribute Value of Slot
2
?act-dur = ?nor-act-dur
NO
YES
2.1
Copy Resource Data Without any Modification to Slot 'new-res-demand'
3.1
Compute Ratio (?res-ratio)
Determine Next Resource
Resource is Material?
NO
YES
3.2
Copy Resource Data Without any Modification to Slot 'new-res-demand'
3.3
Compute New Quantity Value for Resource
3.4
Store Modified Mpw/Eqp Data and Initial Material Data in Attribute of 'new-res-demand' Slot
Any More Resources?
YES
NO
RETURN

Figure C.22 - Flowchart for Function: Modify-Resource-Demand-Pool

The complete source code of the function is provided in Appendix G.

C.6 The Schedule Processor

The *Schedule Processor* is the last processor to be executed by the controller module after all other processors are executed and all constraints are verified to be satisfactory. The main purpose of the processor is to determine final start and finish dates for each activity. In addition, the processor carries out the following tasks:

1- Insert activity number in the 'act-using-res' slot of each resource schema allocated to the activity. As explained in section C.3 above, this is necessary during resource constraints verification to determine activities using any resource during a any specific time period.

2- Remove activity from the scheduling cycle by deleting activity number from the initial activity set (IAS), the eligible activity set (EAS), and the ordered scheduled set (OSS).

3- Modify the attribute value of the 'scheduling-method' slot of the activity schema to: activity-scheduled.

4- Insert the activity number in the 'prec-act-scheduled' slot of all succeeding activities to the activity. This is important for determining subsequent activities to be eligible for scheduling by matching the attribute value of the 'prec-act-scheduled' slot with the attribute value of the 'prec-act' slot in these activities.

5- Modify the early start value of all succeeding activities to be equal to the finish time of the first floor of activity.

For activities with a continuity class B, the following additional functions are also performed by the processor:

6- Determine the next segment of activity to be scheduled based on the total activity segments computed.

7- Group each activity segment continuous with the previous scheduled segment into one whole segment. This is based on comparing activity duration and start and finish dates of both segments. For example, given the previous activity segment between scheduled floors 1 and 5 with a 1 day duration

(1 5 1 0 1 5 6)

then, if the next segment is scheduled between floors 6 and 10 as,

(6 10 1 6 7 9 10)

the processor will group both segments as follows:

(1 10 1 0 1 9 10)

Execution Function

The schedule processor comprises of two ART-IM user defined function:

Function: *Schedule-A*, and *Schedule-B*

The first function is executed for activities with a continuity class A, while the second function is executed for activities with a continuity class B. Figures C.23 and C.24 depict a flowchart of function Schedule-A and Schedule-B respectively. The general format of each function is as follows:

Function: *Schedule-A*

1. Given the start date (start-i) of the first typical floor of activity, compute the other start and finish dates (finish-i, start-j, and finish-j).
2. For each resource defined in the attribute value of the 'new-res-demand' slot of activity, insert activity number in the 'act-using-res' slot of the resource schema.

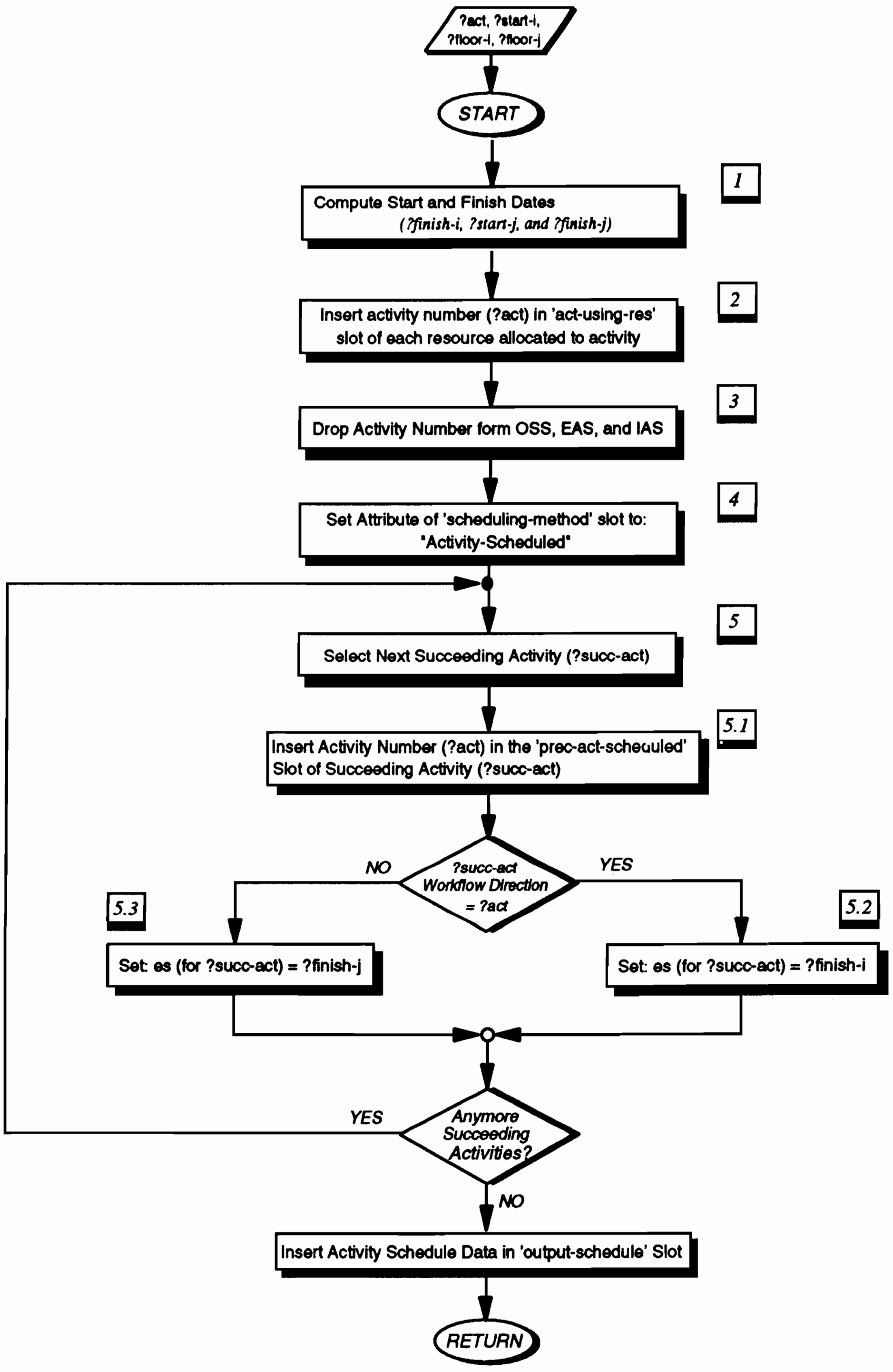
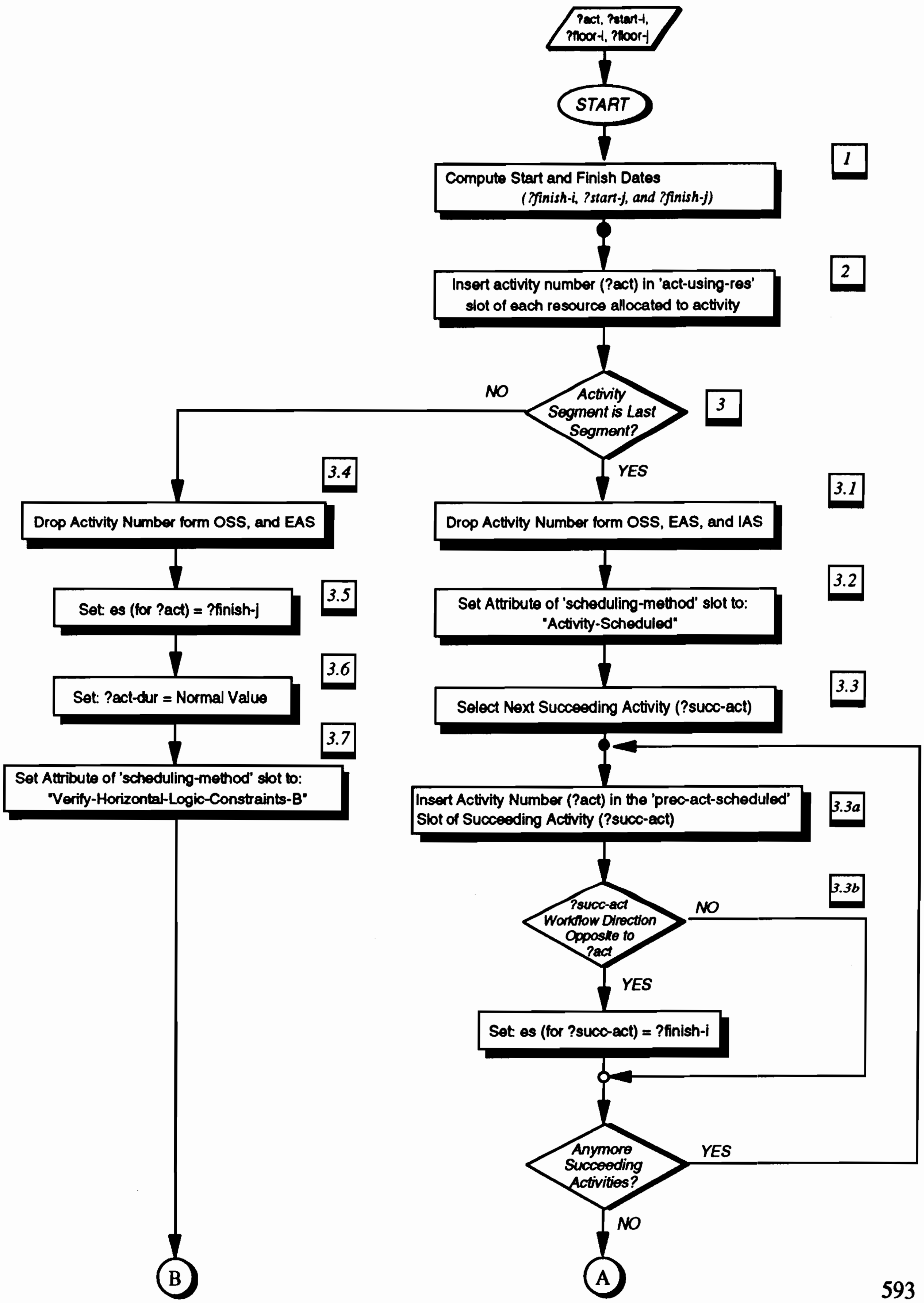

?act, ?start-i, ?floor-i, ?floor-j
START
1
Compute Start and Finish Dates (?finish-i, ?start-j, and ?finish-j)
2
Insert activity number (?act) in 'act-using-res' slot of each resource allocated to activity
3
Drop Activity Number form OSS, EAS, and IAS
4
Set Attribute of 'scheduling-method' slot to: "Activity-Scheduled"
5
Select Next Succeeding Activity (?succ-act)
5.1
Insert Activity Number (?act) in the 'prec-act-scheduled' Slot of Succeeding Activity (?succ-act)
?succ-act Workflow Direction = ?act
NO
YES
5.3
Set: es (for ?succ-act) = ?finish-j
5.2
Set: es (for ?succ-act) = ?finish-i
Anymore Succeeding Activities?
YES
NO
Insert Activity Schedule Data in 'output-schedule' Slot
RETURN

Figure C.23 - Flowchart for Function: Schedule-A



?act, ?start-i, ?floor-i, ?floor-j
START
Compute Start and Finish Dates (?finish-i, ?start-j, and ?finish-j)
1
Insert activity number (?act) in 'act-using-res' slot of each resource allocated to activity
2
Activity Segment is Last Segment?
3
NO
YES
Drop Activity Number form OSS, and EAS
3.4
Set: es (for ?act) = ?finish-j
3.5
Set: ?act-dur = Normal Value
3.6
Set Attribute of 'scheduling-method' slot to: "Verify-Horizontal-Logic-Constraints-B"
3.7
Drop Activity Number form OSS, EAS, and IAS
3.1
Set Attribute of 'scheduling-method' slot to: "Activity-Scheduled"
3.2
Select Next Succeeding Activity (?succ-act)
3.3
Insert Activity Number (?act) in the 'prec-act-scheduled' Slot of Succeeding Activity (?succ-act)
3.3a
?succ-act Workflow Direction Opposite to ?act
3.3b
NO
YES
Set: es (for ?succ-act) = ?finish-i
Anymore Succeeding Activities?
YES
NO
B
A

Figure C.24 - Flowchart for Function: Schedule-B

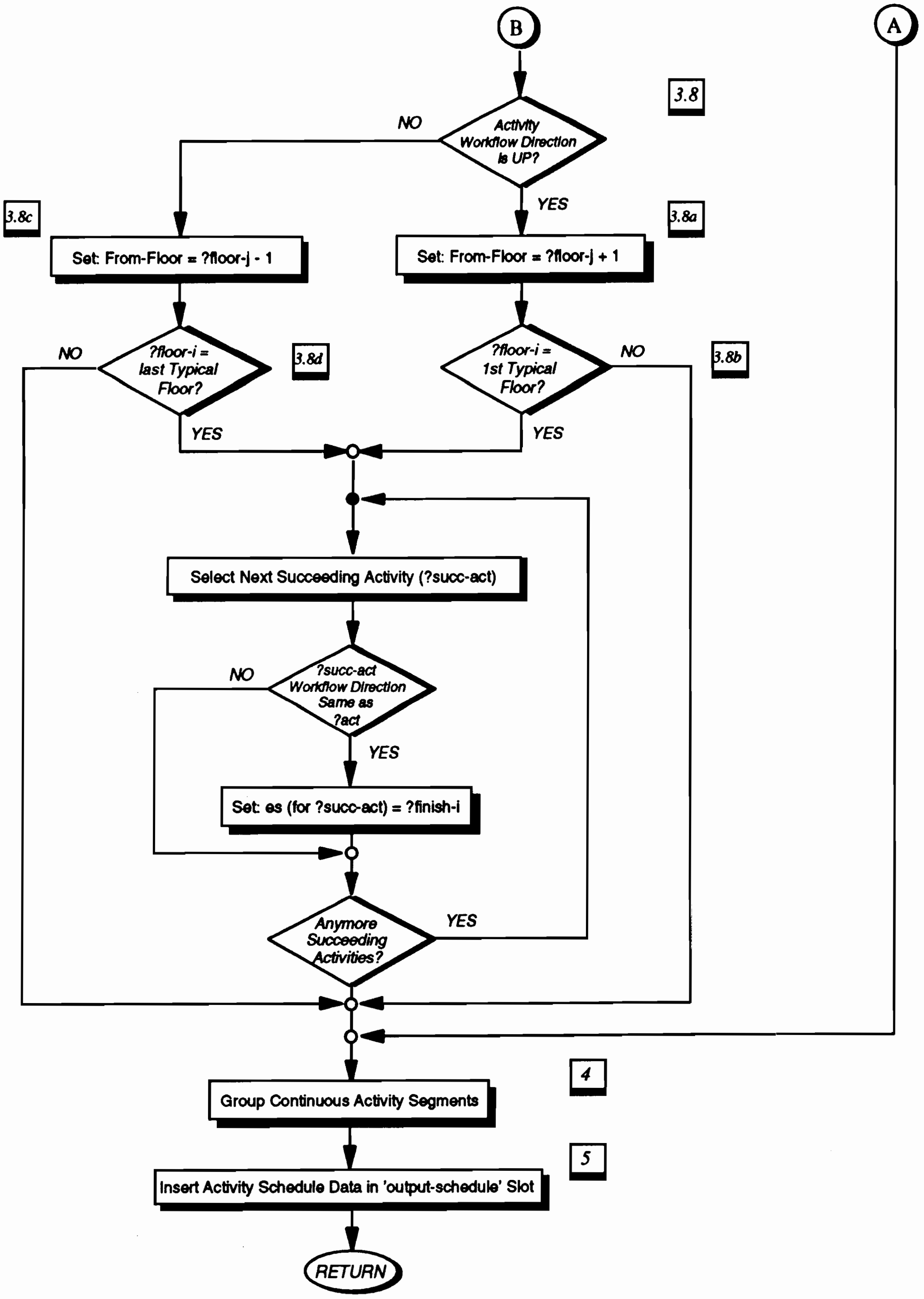

B
A
3.8
NO
Activity Workflow Direction Is UP?
YES
3.8c
3.8a
Set: From-Floor = ?floor-j - 1
Set: From-Floor = ?floor-j + 1
NO
?floor-i = last Typical Floor?
3.8d
?floor-i = 1st Typical Floor?
NO
3.8b
YES
YES
Select Next Succeeding Activity (?succ-act)
NO
?succ-act Workflow Direction Same as ?act
YES
Set: es (for ?succ-act) = ?finish-i
Anymore Succeeding Activities?
YES
Group Continuous Activity Segments
4
Insert Activity Schedule Data in 'output-schedule' Slot
5
RETURN

Figure C.24 - Continued

```
3.  Drop activity from ordered activity set (OSS), eligible activity set (EAS),
    and initial activity set (IAS).

4.  Modify the attribute value of the 'scheduling-method' slot to: activity-
    scheduled.

5.  For each succeeding activity defined for the activity Do,
        5.1 Insert activity number in the "prec-act-scheduled" slot of
        succeeding activity.

        5.2 IF : succeeding activity workflow direction is equal to activity,
           THEN:  set: es (succeeding activity) = finish-i

        5.3 IF : succeeding activity workflow direction is opposite to
               activity,
           THEN:  set: es (succeeding activity) = finish-j

6.  Schedule activity. Insert start and finish floors, selected activity duration,
    and computed start and finish dates in the "output-schedule" slot of
    activity.
```

Function: ***Schedule-B***

```
1.  Given the start date (start-i) of the first typical floor of activity, compute
    the other start and finish dates (finish-i, start-j, and finish-j).

2.  For each resource defined in the attribute value of the 'new-res-demand'
    slot of activity, insert activity number in the 'act-using-res' slot of the
    resource schema.

3.  IF: activity segment is the last segment,
    THEN:
      3.1 Drop activity from ordered activity set (OSS), eligible activity set
          (EAS), and initial activity set (IAS).

      3.2 Modify the attribute value of the 'scheduling-method' slot to:
          "Activity-Scheduled".

      3.3 For each succeeding activity defined for the activity, Do
            3.3a Insert activity number in the "prec-act-scheduled" slot of
                 succeeding activity.
            3.3b IF: succeeding activity workflow direction is opposite to
```

```
                activity,
                THEN:  set: es (succeeding activity) = finish-j

      ELSE:
        3.4  Drop activity from ordered activity set (OSS) and eligible activity
             set (EAS). The activity remains in the initial activity set (IAS).

        3.5  Modify early start value of next activity segment to be equal to the
             scheduled finish time of last floor of current activity segment,
                          set:  es (next segment) = finish-j

        3.6  Modify activity duration to normal value,
                          set:  ?act-dur = normal value

        3.7  Modify the attribute value of 'scheduling-method' slot to the
             function name of the horizontal logic processor for class B
             activities (function: Verify-Horizontal-Logic-Constraints-B).

        3.8  IF: activity workflow direction is UP,
             THEN:
                    3.8a   Modify the attribute value of the 'from-floor' slot in
                           activity schema to the number of the next floor,
                                 set: from-floor = floor-j + 1

                    3.8b   IF: floor-i = first typical floor
                           THEN: For each succeeding activity defined, Do
                                 IF: succeeding activity workflow direction is
                                     the same as activity,
                                 THEN:
                                       set: es (succeeding activity) = finish-i

             ELSE:
                    3.8c   Modify the attribute value of the 'from-floor' slot to
                           the number of the next floor,
                                       set: from-floor = floor-j - 1

                    3.8d   IF: floor-i = last typical floor
                           THEN: For each succeeding activity defined, Do
                                 IF: succeeding activity workflow direction is
                                     the same as activity,
                                 THEN:
                                       set: es (succeeding activity) = finish-i

4.   IF: activity is continuous with previous scheduled segment,
```

THEN: group both segments into one full segment.

5. Schedule activity. Insert start and finish floors, selected activity duration, and computed start and finish dates in the 'output-schedule' slot of activity.

The complete source code of both functions is provided in Appendix G.

C.7 The Duration Processor

The *Duration Processor* is responsible for computing the total duration between any two floors during the scheduling process to determine the start and finish dates of these floors. Given the start date of any floor and the number of the first and last floors, the processor computes the total duration between the start point of the first floor and the finish point of the last floor.

The processor is executed from within all the other six processors in order to determine preliminary and final start and finish dates during constraints verification and final scheduling of activities. The processor is executed using the ART-IM function call:

(Get-Total-Dur ?act ?act-dur ?floor-i ?floor-j)

where,.

get-total-dur	: is the function name of the duration processor,
?act	: is a variable that denotes the activity number under consideration,
?act-dur	: is a variable that denotes the selected duration for the activity segment,
?floor-i	: is a variable that denotes the number of the first floor of activity

segment,

?floor-j	: is a variable that denotes the number of the second floor of activity segment,

To compute the duration between two points of a floor or number of floors, the processor employs two mathematical functions; a straight line function, and a theoretical step function to take into consideration loss in productivity of construction crews.

The straight line function computes the duration of activity as follows:

Total Duration = Activity Duration x Total Number of Floors

The theoretical step function computes the duration taking into consideration the loss in productivity as a result of the combined effect of travel time and learning curve phenomena associated with repetitive work in multi-story projects. To implement this effect, the theoretical function assumes that the activity duration at the first floor increases by 0.1 days every floor. This can be expressed as follows:

$$d_i = d_0 + \sum 0.1 \text{ (for each floor up to floor i)}$$

where,

d_0: denotes the initial activity duration at the first typical floor
d_i: denotes the reduced duration at floor i

For example, given $d_0 = 1$ (floor/day) for an activity, the duration at the eleventh floor will be

$$d_{11} = 1 + (0.1 \text{ x } 10) = 2 \text{ days/floor}$$

The execution of the straight line function or the theoretical step function is based on the '*variable-production*' parameter defined by the user for each activity. If the attribute of this parameter is defined as 'YES', the theoretical function is executed for the activity. On the other hand, if the attribute is defined as 'NO', the straight line function is used to compute the duration for the activity.

Execution Function

the ART-IM user defined function:

Function: *Get-Total-Dur*

is responsible for computing the total duration between any two floors. Figure C.25 depicts a flowchart of the function logic. The general format of the function is as follows:

Function: *Get-Total-Dur*

```
1.  Set variable ?total-time = 0.0

2.  IF: attribute of variable production parameter = YES
    THEN:
            2.1 Determine initial duration of starting floor (floor-i) for activity
            segment
                    set: activity duration = [(floor-i -1) x 0.1] x activity duration

            2.2 Determine total time between first and last floors defined. For
            each floor, Do

                    2.2a  set: total-time = total-time + activity duration
                    2.2b  set: activity duration = activity duration + 0.1

    ELSE:
            2.3 Determine total duration using straight line function
                    set: total-time = total floors x activity duration
```

The complete source code of the function is provided in Appendix G.

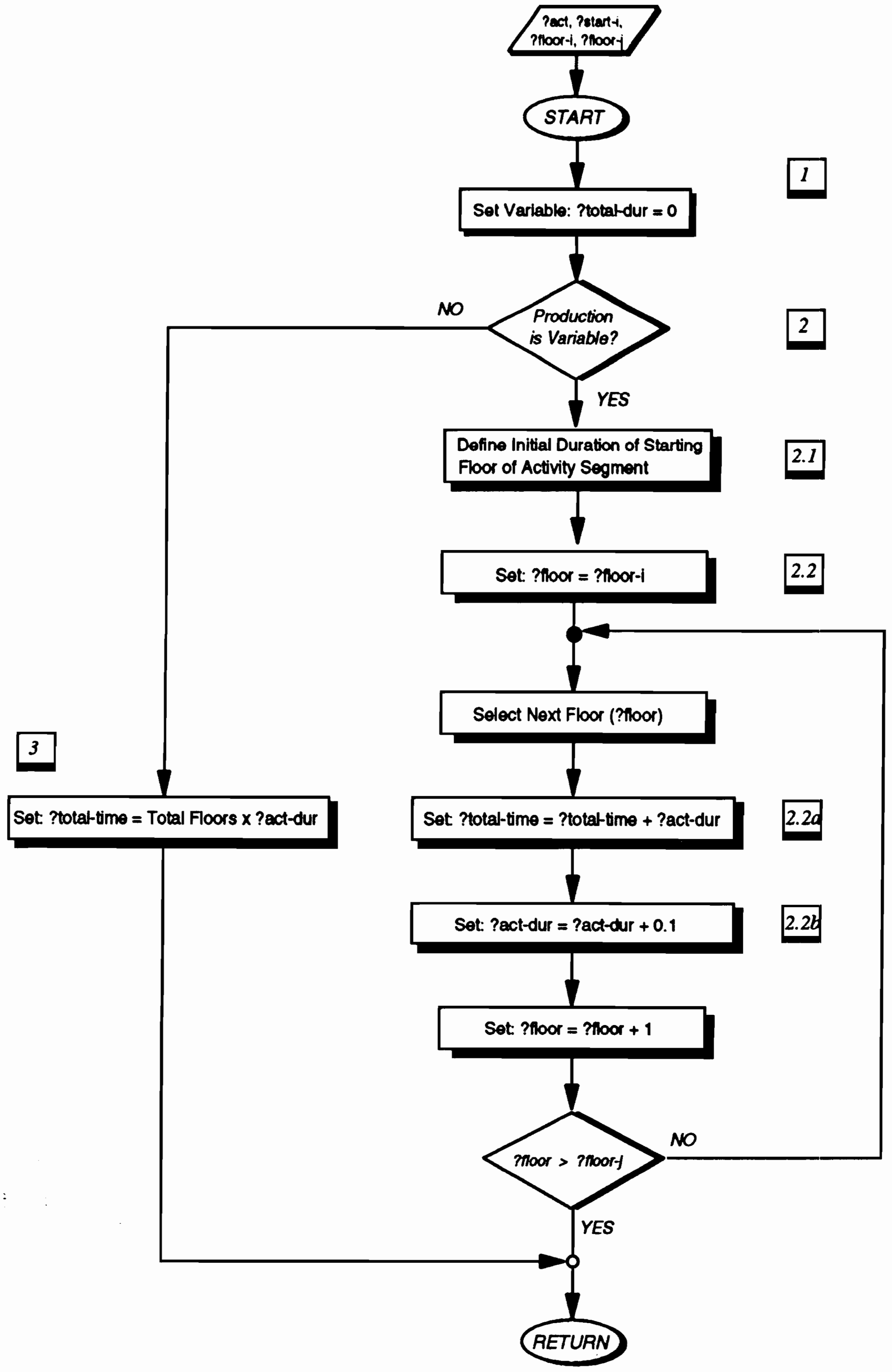

?act, ?start-i, ?floor-i, ?floor-j
START
1
Set Variable: ?total-dur = 0
2
Production is Variable?
NO
YES
2.1
Define Initial Duration of Starting Floor of Activity Segment
2.2
Set: ?floor = ?floor-i
Select Next Floor (?floor)
2.2a
Set: ?total-time = ?total-time + ?act-dur
2.2b
Set: ?act-dur = ?act-dur + 0.1
Set: ?floor = ?floor + 1
?floor > ?floor-j
NO
YES
3
Set: ?total-time = Total Floors x ?act-dur
RETURN

Figure C.25 - Flowchart for Function: Get-Total-Dur

Appendix D

WORKED EXAMPLES USING SCaRC

This section involves several example runs using SCaRC to demonstrate the capabilities of the program to verify the different constraints for different case scenarios and generate a schedule.

For each case scenario considered, data is defined to demonstrate one particular type of constraint tested. A hard copy of the activity flow line graphical output produced by SCaRC is also provided for each example. The hard copy is produced by capturing the graphical image from the screen utilizing *Pizzaz Plus* (ver 1.3).

Four examples are included to test four case scenarios:

1- **Example#1:** *Demonstrate Horizontal Logic Constraints*

2- **Example#2:** *Demonstrate Vertical Logic Constraints*

3- **Example#3:** *Demonstrate Resource Constraints*

4- **Example#4:** *Demonstrate Space Constraints*

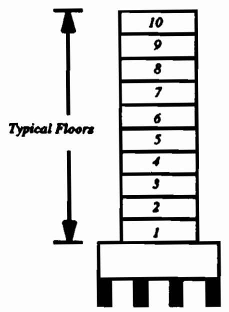
D.1 Example#1: Demonstrate Horizontal Logic Constraints

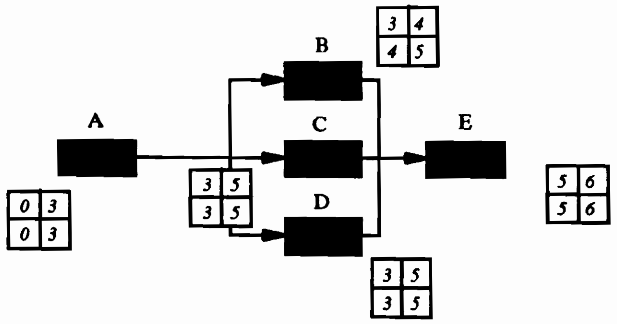
This example demonstrates how the program accounts for *Horizontal Logic Constraints* between activities of the same floor and resulting from the dominant activity pace. Figure D.1 depicts input data defined and Figure D.2 depicts the graphical output schedule as produced by SCaRC.

Key parameters influencing the results produced in Figure D.2 are:

1- Continuity class for activities C and D.

2- Maximum splits parameter for activities C and D.





General Activity Data

	Normal Dur	Maximum Dur	Prec Act	Succ Act	ES	LS
A	3	3	None	B,C,D	0	0
B	1	1	A	E	3	4
C	1	1	A	E	3	3
D	2	2	A	E	3	3
E	1	1	B,C,D	None	5	5

Special Activity Data

	Workflow Dir	Varible Prod	Continuity Class	Max Splits	Ver Logic Act	Ver Lag
A	Up	No	A	0	None	0
B	Up	No	A	0	None	0
C	Up	No	B	10	None	0
D	Up	No	B	3	None	0
E	Down	No	A	0	None	0

Figure D.1 - Input Data for Example #1

Graphical Output for Generated Schedule

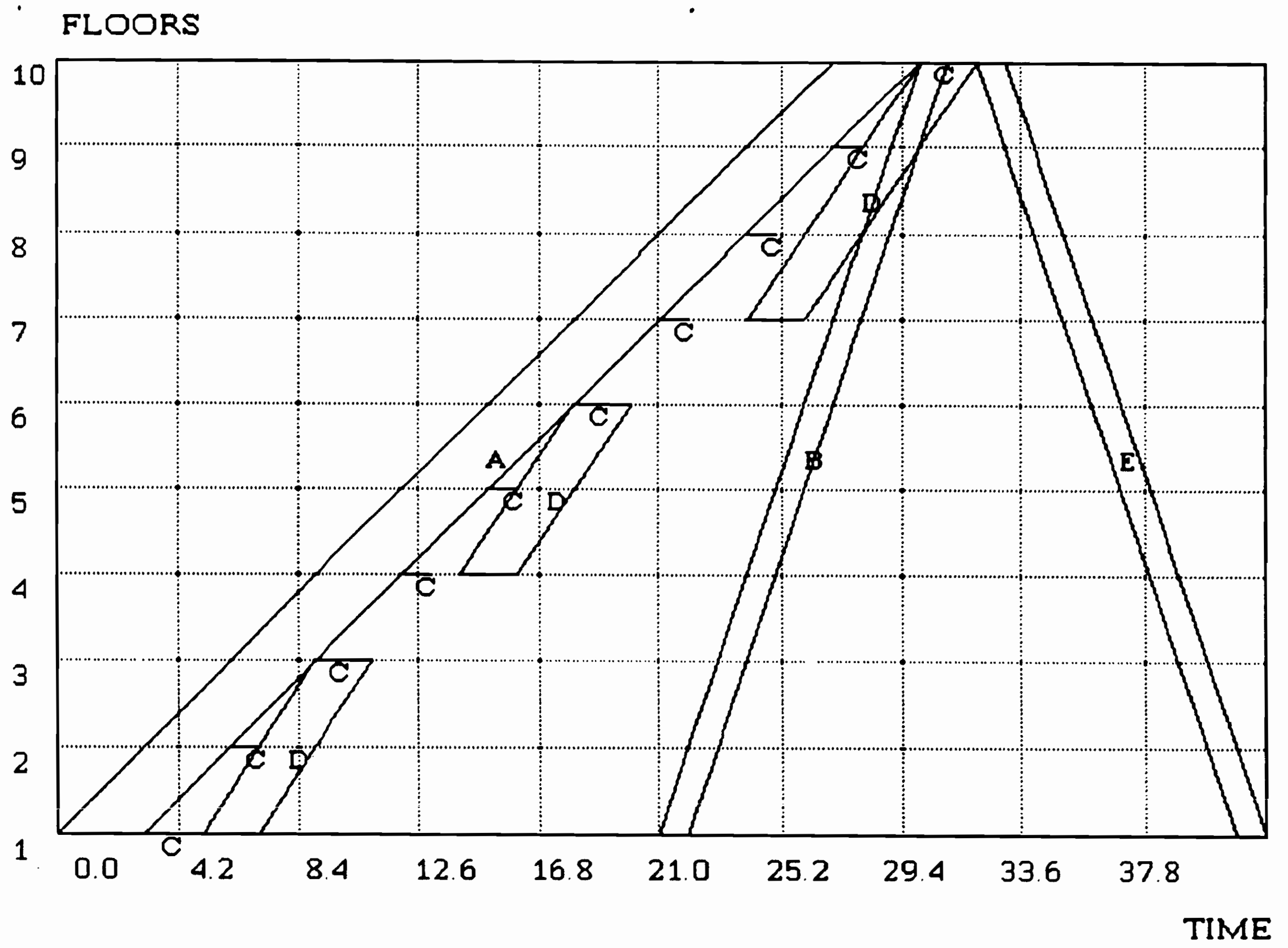


Figure D.2 - Output Schedule Results for Example #1

D.2 Example#2: Demonstrate Vertical Logic Constraints

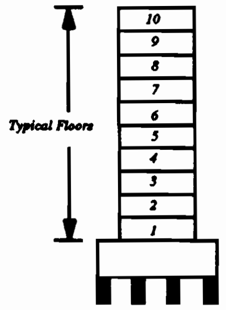
This example demonstrates how the program accounts for *Vertical Logic Constraints* between activities in different floors. Figure D.3 depicts input data defined and Figure D.4 depicts the graphical output schedule as produced by SCaRC.

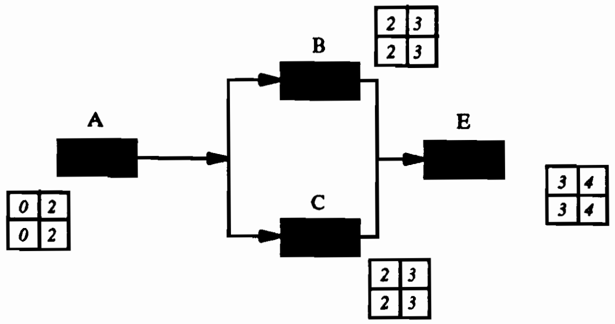
Key parameters influencing the results produced in Figure D.4 are:

1- Continuity class for activities B and C.

2- Maximum splits parameter for activities B and C.

3- Vertical logic activity and vertical lag for activities B and C.





General Activity Data

	Normal Dur	Maximum Dur	Prec Act	Succ Act	ES	LS
A	2	2	None	B,C	0	0
B	1	1	A	E	2	2
C	1	2	A	E	2	2
E	1	1	B,C	None	3	3

Special Activity Data

	Workflow Dir	Varible Prod	Continuity Class	Max Splits	Ver Logic Act	Ver Lag
A	Up	No	A	0	None	0
B	Up	No	B	5	A	2
C	Up	No	B	3	A	3
E	Down	No	A	0	None	0

Figure D.3 - Input Data for Example #2

Graphical Output for Generated Schedule

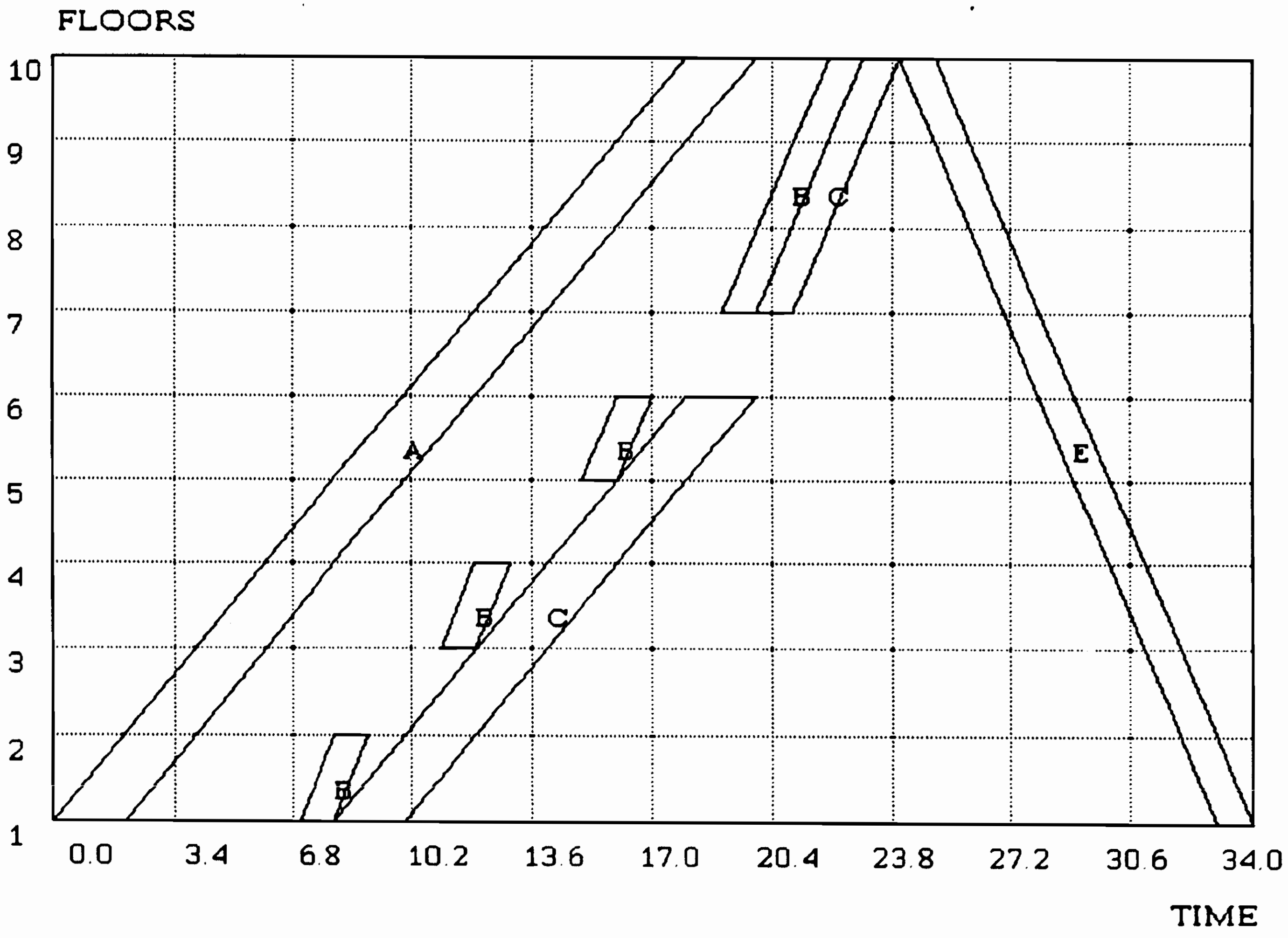


Figure D.4 - Output Schedule Results for Example #2

D.3 Example#3: Demonstrate Resource Constraints

This example demonstrates how the program accounts for ***Resource Constraints*** as a result of the conflict between resource required by each activity and resource that can be made available to the project on a daily basis. Figure D.5 depicts input data defined and Figure D.6 depicts the graphical output schedule as produced by SCaRC.

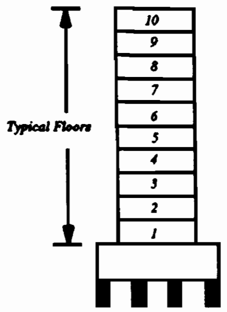
Key parameters influencing the results produced in Figure D.6 are:

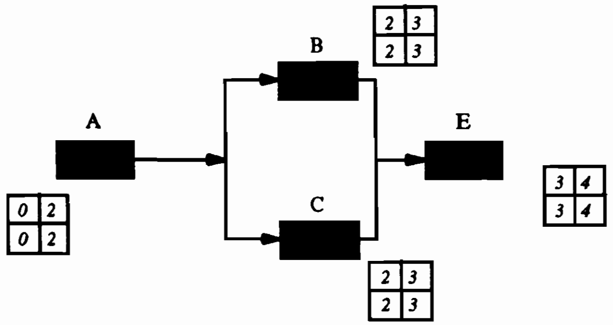
1- Continuity class for activities B and C.

2- Maximum splits parameter for activities B and C.

3- Resource demand for activities A, B and C.

4- Resource availability for resources AA and BB.





General Activity Data

Activity	Normal Dur	Maximum Dur	Prec Act	Succ Act	ES	LS
A	2	2	None	B, C	0	0
B	1	1	A	E	2	2
C	1	2	A	E	2	2
E	1	1	B, C	None	3	3

Special Activity Data

Activity	Workflow Dir	Varible Prod	Continuity Class	Max Splits	Ver Logic Act	Ver Lag
A	Up	No	A	0	None	0
B	Up	No	B	2	None	0
C	Up	No	B	2	None	0
E	Down	No	A	0	None	0

Resource Activity Data

Activity	Resource Code	Resource Type	Resource Qty	Resource Code	Resource Type	Resource Qty
A	AA	Mpw	1	–	–	–
B	AA	Mpw	1	BB	Eqp	1
C	AA	Mpw	1	BB	Eqp	1
E	AA	Mpw	1	–	–	–

Resource Data

Resource	Resource Type	Available Qty	Mpw/Eqp Space Demand	Material Space Demand
AA	MPW	2	–	–
BB	EQP	1	–	–

Figure D.5 - Input Data for Example #3

Graphical Output for Generated Schedule

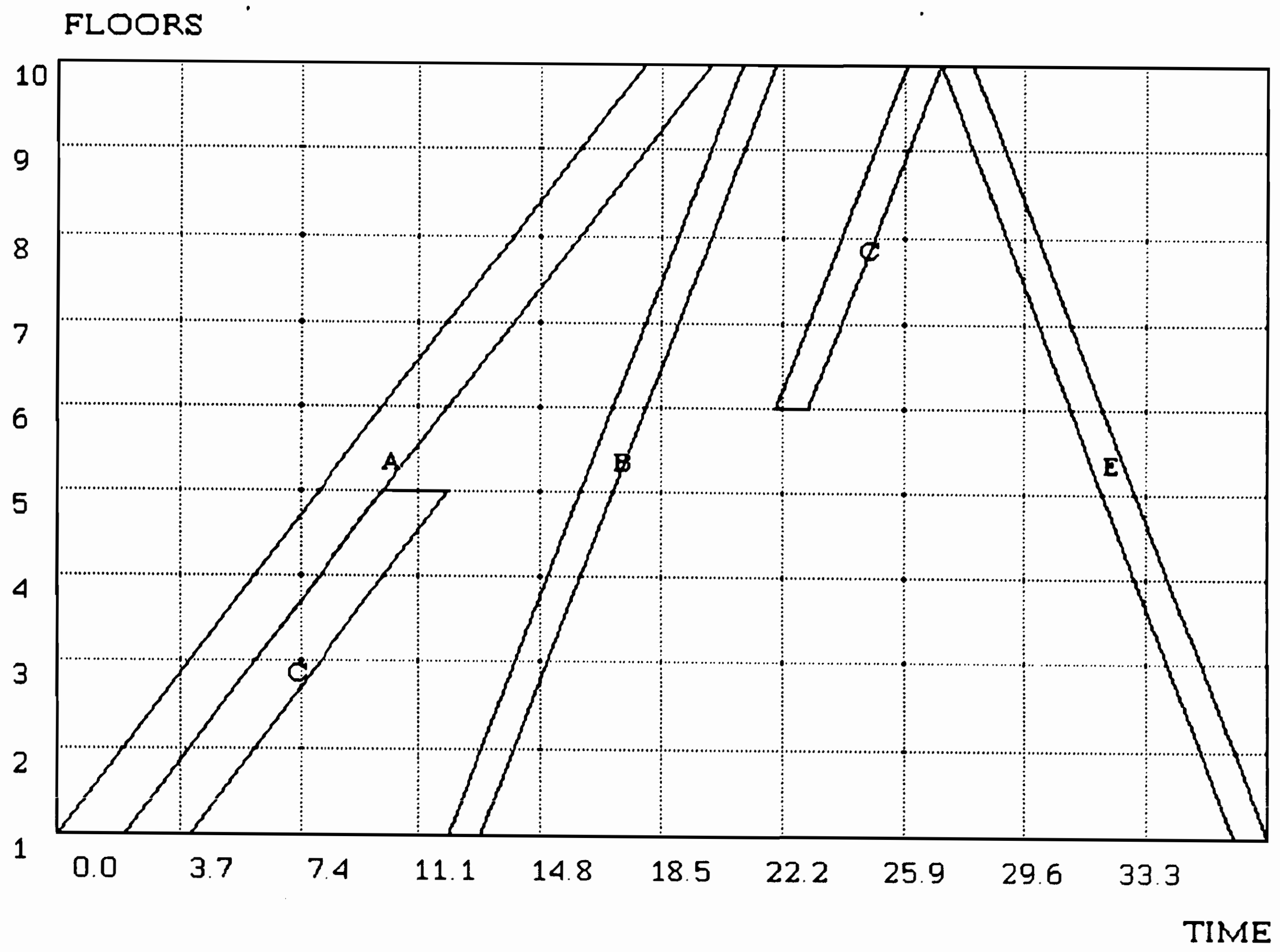


Figure D.6 - Output Schedule Results for Example #3

D.4 Example#4: Demonstrate Space Constraints

This example demonstrates how the program accounts for *Space Constraints* as a result of the conflict between work space required by each activity (work space demand) and work space availability of each zone/layer work block. Data defined in Chapter 7 are used to demonstrate this example. Figure D.7 depicts the graphical output schedule as produced by SCaRC.

Graphical Output for Generated Schedule

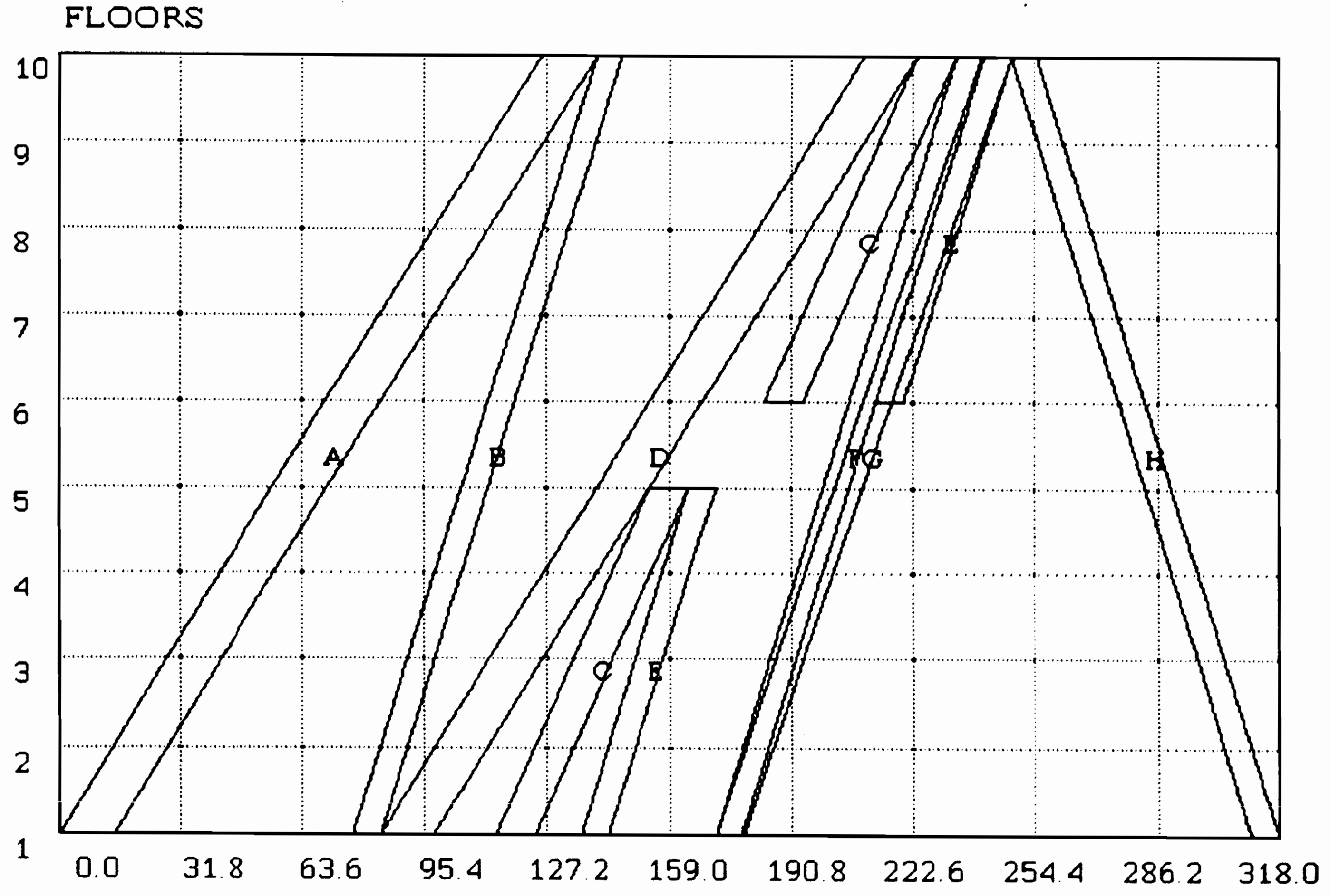


Figure D.7 - Output Schedule Results for Example #4

Appendix E

GRAPHICAL OUTPUT PROGRAM LISTING
(C Code)

```
#include <stdio.h>
#include <graph.h>
#include <conio.h>

#define SIZE 50
struct videoconfig screen;

char  act_num[SIZE][5];
int   f1[SIZE], f2[SIZE];
float t1[SIZE], t2[SIZE], t3[SIZE], t4[SIZE];

main ()
{
  char  floor[3], time[8];
  int   i, n, x, y, x1, y1, rows, cols, min_floor=1000, max_floor=1;
  float t, min_time=10000.0, max_time=0.0;
  FILE  *fpin;


  if ( (fpin = fopen ("d:\\database\\output1.art", "r")) == NULL)
    {
     printf ("ERROR...Can Not Open Designated File \n");
     exit (0);
    }


  n=0;
  while (!feof(fpin))
    {
     fscanf (fpin, "%4s%3d%3d%6f%6f%6f%6f\n", act_num[n], &f1[n], &f2[n], &t1[n], &t2[n], &t3[n], &t4[n]);

     if ( t1[n] < min_time)
          min_time = t1[n];
     if ( t4[n] > max_time)
          max_time = t4[n];
     if (f1[n] < min_floor)
          min_floor = f1[n];
     if (f2[n] > max_floor)
          max_floor = f2[n];

     n +=1;
    }

  fclose(fpin);
  if (_setvideomode (_VRES16COLOR) == 0)
    {
      printf ("ERROR...This Program Requires a VGA Graphics Card \n");
      exit (0);
    }

  if (_registerfonts ("d:\\qc25\\fonts\\*.FON") < 0)
    {
      printf ("ERROR...Can Not Register Fonts");
      exit (0);
    }


 _getvideoconfig (&screen);

  x = screen.numxpixels;
  y = screen.numypixels;
  cols = screen.numtextcols;
  rows = screen.numtextrows;

 _setfont ("t'tms rmn'");
 _setcolor (3);
 _moveto (55, 10);
 _outgtext ("Graphical Output for Generated Schedule");

 _setfont (" h15w8 t'tms rmn'");
 _setcolor (4);
 _moveto_w (55, 65);
 _outgtext ("FLOORS");
 _moveto_w (550, 465);
 _outgtext ("TIME");


 _setviewport (0, 50, 50, y-10);
 _setwindow (1, 0, min_floor-1, 3, max_floor+1);
  for (i=min_floor; i<=max_floor; i+=1)
    {
      _setfont ("h13w5 t'helv'");
      _setcolor (4);
      _moveto_w (2, i);
          sprintf(floor, "%d", i);
      _outgtext (floor);
    }


 _setviewport (0, y-40, x-20, y);
 _setwindow (1, min_time-((max_time-min_time)/10.), 0 , max_time+((max_time-min_time)/10.), 1);
  for (t=min_time; t<=max_time; t+=(max_time-min_time)/10.)
    {
      _setfont ("h13w5 t'helv'");
      _setcolor (4);
      _moveto_w (t, 1);
          sprintf(time, "%5.1f", t);
      _outgtext (time);
    }
  _setviewport (0, 50, x, y-10);
  _setwindow (1, min_time-((max_time-min_time)/10.), 0, max_time+((max_time-min_time)/10.), max_floor+1);
  _setcolor (4);
  _rectangle_w (_GBORDER, min_time, min_floor, max_time, max_floor);

   for (i=min_floor; i<max_floor; i+=1)
     {
       _setcolor(4);
       _setlinestyle (0xAAAA);
       _moveto_w (min_time, i);
       _lineto_w (max_time, i);
     }

   for (t=min_time+((max_time-min_time)/10.), i=3; t<max_time; t+=(max_time-min_time)/10.)
     {
       _setcolor (4);
       _setlinestyle (0xAAAA);
       _moveto_w (t, min_floor);
       _lineto_w (t, max_floor);
     }

   for (i=0; i<n; ++i)
     {

       _setcolor (2);
       _setlinestyle (0xffff);
       _moveto_w (t1[i], f1[i]);
       _lineto_w (t2[i], f1[i]);
       _lineto_w (t4[i], f2[i]);
       _lineto_w (t3[i], f2[i]);
       _lineto_w (t1[i], f1[i]);

       _setfont ("h12w6 t'tms rmn'");
       _setcolor (7);
       _moveto_w ((t2[i]+t3[i])/2., (f1[i]+f2[i])/2.);
       _outgtext (act_num[i]);

     }

   getchar();
  _unregisterfonts ();
  _clearscreen (_GCLEARSCREEN);
  _setvideomode (_DEFAULTMODE);
}
```

Appendix F

DATABASE SYSTEM PROGRAM LISTING
(dBASE IV Code)

```
set talk off
set status off
set scoreboard off
set escape off


*---------Define path for database files
set path to d:\database
clear


*---------Define popup menu system
define popup main from 7,13  to 21,66
define bar 1 of main prompt  "            M-1  MAIN MENU   " skip
define bar 3 of main prompt  "  1- Define System Data            (M-2) "
define bar 5 of main prompt  "  2- Clear all Input/Output Records "
define bar 7 of main prompt  "  3- Define Input Data            (M-3)  "
define bar 9 of main prompt  "  4- View Output Data              (M-7)  "
define bar 13 of main prompt "  Q  Exit to DOS"

define popup system from 7,15 to 17,65
define bar 1 of system prompt  "          M-2  SYSTEM DATA MENU " skip
define bar 3 of system prompt  "  1- Input/Modify Global Space Demand Data "
define bar 7 of system prompt  "  Q  Return to Main Menu           (M-1) "

define popup project from 7,9 to 22,70
define bar 1 of project prompt  "           M-3  PROJECT DATA INPUT MENU  " skip
define bar 3 of project prompt  "  1- Define Activity Data         (M-4) "
define bar 5 of project prompt  "  2- Define Resource Data          (M-5)"
define bar 7 of project prompt  "  3- Define Work Block Data        (M-6)"
define bar 9 of project prompt  "  4- Input Other Project Data "
define bar 11 of project prompt "  5- Generate Input Data ASCII Files from DBF files"
define bar 13 of project prompt "  Q  Return to Main Menu           (M-1)"


define popup activity from 7,15 to 21,63
define bar 1 of activity prompt  "       M-4  ACTIVITY DATA INPUT MENU " skip
define bar 3 of activity prompt  "  1- Input General Activity Data"
define bar 5 of activity prompt  "  2- Input Special Activity Data"
define bar 7 of activity prompt  "  3- Input Activity Resource Demand Data"
define bar 9 of activity prompt  "  4- Input Activity Work Space Data"
define bar 12 of activity prompt "  Q  Return to Project Menu       (M-3)"


define popup resource from 7,13 to 17,66
define bar 1 of resource prompt  "        M-5  RESOURCE DATA INPUT MENU" SKIP
define bar 3 of resource prompt  "  1- Extract Work Space Demand for Unit Resources"
define bar 5 of resource prompt  "  2- Input Resource Data"
define bar 8 of resource prompt  "  Q  Return to Project Menu                 (M-3)"


define popup block from 7,8 to 23,70
define bar 1 of block prompt  "            M-6   WORK BLOCK DATA INPUT MENU " skip
define bar 3 of block prompt  "  1- Input Zone and Layer Work Block Numbers"
define bar 5 of block prompt  "  2- Extract Design Elements Data from CAD Model ASCII File"
define bar 7 of block prompt  "  3- Allocate Work Blocks to Design Elements"
define bar 9 of block prompt  "  4- Generate Total Work Space Availability for Work Blocks"
define bar 11 of block prompt "  5- View Generated Space Availability for Work Blocks"
define bar 14 of block prompt "  Q  Return to Project Data Menu               (M-3)"


define popup output from 7,12 to 22,67
define bar 1 of output prompt  "         M-7   SCHEDULE DATA OUTPUT MENU " skip
define bar 3 of output prompt  "  1- Generate Output Data DBF File from ASCII File"
define bar 5 of output prompt  "  2- View Output Results (Graphical Format) "
define bar 7 of output prompt  "  3- View Output Results (Screen  Format)  "
define bar 9 of output prompt  "  4- View Output Results (Tabular Format) "
define bar 13 of output prompt "  Q  Return to Main Menu                     (M-1)"



*--------Set reactions for popup menus
on selection popup main do menu_1
on selection popup system do menu_2
on selection popup project do menu_3
on selection popup activity do menu_4
on selection popup resource do menu_5
on selection popup block do menu_6
on selection popup output do menu_7



*-------Activate the menu system
@ 0,0 fill to 2,79 color w+/r+
@ 1,22 say "  SCHEDULING  SYSTEM  DATABASE" COLOR gr+/r+

activate popup main



procedure menu_1
   Do Case
      Case Bar()=3
         Do sys_menu
      Case Bar()=5
         Do del_data
      Case Bar()=7
         Do pro_menu
      Case Bar()=9
         Do out_menu
      Case Bar()=13
         Do quit
   EndCase
```

```
procedure menu_2
    Do Case
        Case Bar()=3
            Do global
        Case Bar()=7
            Do retn_m_m
    EndCase


procedure menu_3
    Do Case
        Case Bar()=3
            Do act_menu
        Case Bar()=5
            Do res_menu
        Case Bar()=7
            Do blk_menu
        Case Bar()=9
            Do other
        Case Bar()=11
            Do gn_ascii
        Case Bar()= 13
            Do retn_m_m
    EndCase


procedure menu_4
    Do Case
        Case Bar()=3
            Do act_gen
        Case Bar()=5
            Do act_par
        Case Bar()=7
            Do act_res
        Case Bar()=9
            Do act_spc
        Case Bar()=12
            Do retn_p_m
    EndCase


procedure menu_5
    Do Case
        Case Bar()=3
            Do algthm_1
        Case Bar()=5
            Do res_data
        Case Bar()=8
            Do retn_p_m
    EndCase
```

```
procedure menu_6
    Do Case
        Case Bar()=3
            Do blk_no
        Case Bar()=5
            Do algthm_2
        Case Bar()=7
            Do e_alloc
        Case Bar()=9
            Do algthm_3
            Case Bar()=11
                Do view_blk
        Case Bar()=14
            Do retn_p_m
    EndCase



procedure menu_7
    Do Case
        Case Bar()=3
            Do gn_dbf
            Case Bar()=5
                Do graph
        Case Bar()=7
            Do v_screen
        Case Bar()=9
            Do v_table
        Case Bar()=13
            Do retn_m_m
    EndCase





*--------------------- MENU-1 PROCEDURES
procedure sys_menu
    clear
    @ 0,0 fill to 2,79 color w+/r+
    @ 1,25 say "SCHEDULING  SYSTEM  DATABASE" COLOR gr+/r+
    activate popup system
    return



procedure del_data

  Decide = "N"
  close format
  clear
  @ 10,24 fill to 14,54 color GR+/R
  @ 12,28 Say "Are You Sure (Y/N) ?" color GR+/R, GR+/R
  @ 12,51 Get Decide color GR+/R
  Read
  IF Upper(Decide) = "Y"

      Clear
      @ 10,24 fill to 14,52 color GR+/R
      @ 12,29 Say "Clearing All Files" color GR+*/R

      use act-gen
      delete all
      pack

      use act-par
      delete all
      pack

      use act-res
      delete all
      pack

      use act-spc
      delete all
      pack

      use res
      delete all
      pack

      use block
      delete all
      pack

      use element
      delete all
      pack

      use other
      delete all
      pack

      use output
      delete all
      pack

      clear

  ELSE
     clear
     return
  ENDIF


procedure pro_menu
     clear
     @ 0,0 fill to 2,79 color w+/r+
     @ 1,25 say "SCHEDULING  SYSTEM  DATABASE" COLOR gr+/r+
     activate popup project
     return
```

```
procedure out_menu
    clear
    @ 0,0 fill to 2,79 color w+/r+
    @ 1,25 say "SCHEDULING  SYSTEM  DATABASE" color gr+/r+
    activate popup output
    return


procedure quit
    quit



*------------------------------ MENU-2 PROCEDURES
procedure global
    clear
    use res-glb
    set format to s-1
    append
    return



*------------------------------ MENU-3 PROCEDURES
procedure act_menu
    clear
    @ 0,0 fill to 2,79 color w+/r+
    @ 1,25 say "SCHEDULING  SYSTEM  DATABASE" color gr+/r+
    activate popup activity
    return


procedure res_menu
    clear
    @ 0,0 fill to 2,79 color w+/r+
    @ 1,25 say "SCHEDULING  SYSTEM  DATABASE" color gr+/r+
    activate popup resource
    return


procedure blk_menu
    clear
    @ 0,0 fill to 2,79 color w+/r+
    @ 1,25 say "SCHEDULING  SYSTEM  DATABSE" color gr+/r+
    activate popup block
    return



procedure other
    clear
    use other
    set format to s-9
    append
    return


Procedure gn_ascii

    clear
    @ 10,24 fill to 14,55 color GR+/R
    @ 12,27 say "Insert Diskatte in Drive a:" color GR+/R
    @ 13,27 say "... press any key when done" color GR+/R
    wait " "
    clear
    @ 10,24 fill to 14,55 color GR+/R
    @ 12,29 say "Generating ASCII Files" color GR+*/R

    !del a:\*.art

    Use act-gen
    Copy to a:\act-gen.art Type sdf

    Use act-par
    Copy to a:\act-par.art Type sdf Fields act_no, wkflow_dir, var_prod,;
    cont_class, max_splits, vlog_act, vlog_lag

    Use act-res
    Copy to a:\act-res.art Type sdf Fields act_no, code_1,;
    code_2, code_3, code_4, type_1, type_2, type_3, type_4, qty_1, qty_2,;
    qty_3, qty_4

    Use act-spc
    Copy to a:\act-spc.art Type sdf Fields act_no, a_sp_class,;
    mp_eq_blok, mat_blok

    Use res
    Copy to a:\res.art Type sdf Fields res_code, res_type, avl_qty, u_total

    Use block
    Copy to a:\block.art Type sdf

    Use other
    Copy to a:\other.art Type sdf

    clear
    return


*---------------------------------- MENU-4 PROCEDURES

procedure act_gen
    clear
    use act-gen
    set format to s-2
    append
    replace all max_dur with nor_dur for max_dur = 0.0
    n=0
    Scan
            num = act_no

            Use act-par order number
               Locate For act_no = num
               IF .NOT. Found()
                         Append From act-gen For act_no = num
               ENDIF

            Use act-res order number
               Locate For act_no = num
               IF .NOT. Found()
                         Append From act-gen For act_no = num
               ENDIF

            Use act-spc
               Locate For act_no = num
               IF .NOT. Found()
                    Append From act-gen For act_no = num
               ENDIF

            Use act-gen
              Go Top
              Skip n
              n = n + 1

    EndScan
    Return



procedure act_par
    clear
    use act-par order number
    n=0
    Scan
            num = act_no
            Use act-gen order number
              Locate For act_no = num
              IF .NOT. Found()
                Use act-par
                Delete For act_no = num
                Pack
              ENDIF

            Use act-par order number
              Go Top
              Skip n
              n = n + 1

    EndScan

    set format to s-3
    set fields to act_no/r, act_dscrp/r, wkflow_dir, cont_class, max_splits,;
    var_prod, vlog_act, vlog_lag
    Go Top
    Edit Noappend Nomenu
    Replace All vlog_act With "NONE" For vlog_act=" "
    Return



procedure act_res
    Clear
    Use act-res order number
    n=0
    Scan
        num = act_no
         Use act-gen order number
           Locate For act_no = num
           IF .NOT. Found()
             Use act-res
             Delete For act_no = num
             Pack
          ENDIF

         Use act-res order number
           Go Top
           Skip n
           n = n + 1

    EndScan

    Set Format To s-4
    Set Fields To act_no/r, act_dscrp/r, code_1, code_2, code_3, code_4,;
    type_1, type_2, type_3, type_4, qty_1, qty_2, qty_3, qty_4
    Go Top
    Edit NoAppend NoMenu
    Declare array_1 [1,2], array_2 [1,2], array_3 [1,2], array_4 [1,2]

    n=0
    scan.
         res1  = code_1
         type1 = type_1
         res2  = code_2
         type2 = type_2
         res3  = code_3
         type3 = type_3
         res4  = code_4
         type4 = type_4

         Store res1  To array_1 [1,1]
         Store type1 To array_1 [1,2]
         Store res2  To array_2 [1,1]
         Store type2 To array_2 [1,2]
         Store res3  To array_3 [1,1]
         Store type3 To array_3 [1,2]
         Store res4  To array_4 [1,1]
         Store type4 To array_4 [1,2]

         Use res Order code
```

```
          Locate For res_code = res1
          IF .NOT. Found()
            IF res1 <> " "
                    Append From Array array_1
            ENDIF
          ENDIF

          Locate For res_code = res2
          IF .NOT. Found()
            IF res2 <> " "
                    Append From Array array_2
            ENDIF
          ENDIF

          Locate For res_code = res3
          IF .NOT. Found()
            IF res3 <> " "
                    Append From Array array_3
            ENDIF
          ENDIF

          Locate For res_code = res4
          IF .NOT. Found()
            IF res4 <> " "
                    Append From Array array_4
            ENDIF
          ENDIF

        Use act-res order number
          Go Top
          Skip n
          n = n + 1

     EndScan
     Return



procedure act_spc
     clear
     use act-spc order number
     n=0
     Scan
            num = act_no
            Use act-gen Order number
              Locate For act_no = num
              IF .NOT. Found()
                Use act-spc
                Delete For act_no = num
                Pack
              ENDIF

            Use act-spc order number
              Go top
              Skip n
              n = n + 1
```

```
      EndScan

      Set Format to s-5
      Set Fields to act_no/r, act_dscrp/r, a_sp_class, mp_eq_blok, mat_blok
      Go Top
      Edit Noappend Nomenu
      Replace All mp_eq_blok With "NONE" For mp_eq_blok=" "
      Replace All mat_blok With "NONE" For mat_blok=" "
      Return


*------------------------------- MENU-5 PROCEDURES
procedure algthm_1

     Clear
     Use res Order code
     n = 0
     Scan
          resource = Upper(res_code)

          Use res-glb Order Code
          Locate For Upper(res_code) = resource

          IF Found()
             space_1 = g_physical
                 space_2 = g_surround
          ENDIF

          Use res Order code
          Replace g_physical With space_1 For res_code = resource
              Replace g_surround With space_2 For res_code = resource

          Go Top
          Skip n
          n = n + 1

    EndScan
    Return


procedure res_data

   Clear
   Use res Order code

   n=0
   Scan
          res = res_code
          Use act-res order number
            Locate For code_1=res .OR. code_2=res .OR. code_3=res .OR. code_4=res
            IF .NOT. Found()
              Use res
              Delete For res_code = res
            Pack
          ENDIF

        Use res Order code
          Go Top
          Skip n
          n = n + 1
   EndScan

   Set Format TO s-6
   Set Fields To res_code/r, res_type/r, g_physical/r, g_surround/r,;
               u_total, avl_qty, u_physical, u_surround
   Go Top
   Edit Noappend Nomenu
   Replace All u_total With u_physical+u_surround For u_total = 0.0
   Return


*------------------------------- MENU 6 PROCEDURES
procedure blk_no

        clear
        use block
        set format to s-7
        set fields to block_no, space_aval/r
        append
        return


procedure algthm_2
        clear
        use element
        append from a:\cad.dat type sdf
        replace all d1 with 1.0 for d1 = 0.0
        replace all d2 with 1.0 for d2 = 0.0
        replace all d3 with 1.0 for d3 = 0.0
       .return


procedure e_alloc
        clear
        use element order type
        set format to s-8
        set fields to elmnt_no/r, elmnt_type/r, d1/r, d2/r, d3/r, blk_no_1,;
        blk_no_2, blk_no_3
        Go Top
     edit noappend nomenu
        return


procedure algthm_3

        clear
        use Block Order num
        n=0
        Scan
               cons_spc = 0.0
               b_num = Upper(block_no)

               use element order type
                 Locate for Upper(elmnt_type) = "SLAB" .AND. Upper(blk_no_1) = b_num
                 If Found()
                    grss_spc = d1*d2*d3
                 EndIf

                 Locate for Upper(blk_no_1) = b_num .AND. Upper(elmnt_type) <> "SLAB"
                 If Found()
                    cons_spc = cons_spc + (d1*d2*d3)
                 EndIf

                 Locate for Upper(blk_no_2) = b_num .AND. Upper(elmnt_type) <> "SLAB"
                 If Found()
                    cons_spc = cons_spc + (d1*d2*d3)
                 EndIf

                 Locate for Upper(blk_no_3) = b_num .AND. Upper(elmnt_type) <> "SLAB"
                 If Found()
                    cons_spc = cons_spc + (d1*d2*d3)
                 EndIf

               use Block Order num
                 replace space_aval with grss_spc-cons_spc For Upper(block_no) = b_num

               Go Top
               Skip n
               n = n + 1
        EndScan
        Return



procedure view_blk
        clear
        use block order num
        set format to s-7
        set fields to block_no/r, space_aval/r
        Go Top
        edit noappend nomenu
        return


*-------------------------------- MENU 7 PROCEDURES
procedure gn_dbf
    clear
    @ 10,24 fill to 14,55 color GR+/R
    @ 12,30 say "Generating DBF File" color GR+*/R

    use output
    delete all
    pack
    append from a:\output.art Type sdf

    !del d:\database\output1.art
    copy to d:\database\output1.art type sdf fields act_no, from_floor,;
           to_floor, from_start, from_finsh, to_start, to_finsh
    return

procedure graph
    clear
    !cd\qc25\bin
    !graph
    clear
    return

procedure v_screen
    clear
    use output order numstart
    set format to s-10
    edit noedit noappend
    clear
    return

procedure v_table
    clear
    use output order numstart
    browse noedit noappend
    clear
    return

*---------------------------------- GENERAL PROCEDURES
procedure retn_m_m
    deactivate popup
    activate popup main

procedure retn_p_m
    deactivate popup
    activate popup project
```

Appendix G

KNOWLEDGE-BASED SCHEDULING SYSTEM PROGRAM LISTING
(ART-IM Code)

```
(deffacts time (new-time 0.0))

(defglobal ?*dur* = 0.0)
(defglobal ?*from-floor* = 0)
(defglobal ?*to-floor*   = 0)

;*********************************************************************
(defschema prec-act
  (instance-of slot)
  (cardinality multiple))

(defschema succ-act
  (instance-of slot)
  (cardinality multiple))

(defschema prec-act-scheduled
  (instance-of slot)
  (cardinality multiple))

(defschema output-schedule
  (instance-of slot)
  (cardinality multiple))

(defschema act-sharing-space
  (instance-of slot)
  (cardinality multiple))

(defschema ias
  (instance-of slot)
  (cardinality multiple))

(defschema eas
  (instance-of slot)
  (cardinality multiple))

(defschema res-demand
  (instance-of slot)
  (cardinality multiple))

(defschema new-res-demand
  (instance-of slot)
  (cardinality multiple))

(defschema act-using-res
  (instance-of slot)
  (cardinality multiple))

 (defschema saturation-curve
  (instance-of slot)
  (cardinality multiple))

 (defschema compare
  (act-num compare)
  (ls 1000000))

 (defschema act-set
  (ias)
  (eas)
  (oss))

 (defschema temp-activity)

 (defschema activity
  (prec-act-scheduled)
  (output-schedule)
  (act-dur)
  (res-demand)
  (workflow-dir)
  (continuity-class)
  (max-splits)
  (variable-production)
  (ver-logic-act)
  (ver-logic-lag)
  (space-demand-class)
  (mpw-eqp-block)
  (mat-block)
  (new-res-demand)
  (act-sharing-space)
  (output-schedule)
 )

 (defschema resource
  (act-using-res)
 )

 (defschema work-block
 )

 (defschema Productivity-SCF
  (saturation-curve (0.0 1.0 1.0) (1.0 1.3 1.25)
                    (1.3 1.5 1.5) (1.5 1000000. 10000000.))
 )

 (defschema none
  (instance-of work-block)
  (space-available 100000000.))
```

```
(def-art-fun  Get-Total-Dur (?act ?act-dur ?floor-i ?floor-j)

  (bind ?total-time 0)
  (IF (= (get-schema-value ?act variable-production) YES) THEN
    (IF (= (get-schema-value ?act workflow-dir) UP)
       THEN (bind ?inc 0.1)
            (bind ?x ?floor-i)
            (bind ?y ?floor-j)
       ELSE (bind ?inc -0.1)
            (bind ?x ?floor-j)
            (bind ?y >fkoor-i)
    )

    (bind ?t (- ?floor-i 1))
    (bind ?act-dur (+ (* ?t 0.1) ?act-dur))

    (FOR ?floor FROM ?x TO ?y DO
       (bind ?total-time (+ ?total-time ?act-dur))
       (bind ?act-dur (+ ?act-dur ?inc))
    )
  )

  (IF (= (get-schema-value ?act variable-production) NO) THEN
    (bind ?total-time (* ?act-dur (+ (abs (- ?floor-i ?floor-j)) 1)))
  )

  (bind ?total-time ?total-time)
)


(def-art-fun Schedule-A (?act ?start-i ?floor-i ?floor-j)

  (bind ?act-dur (get-schema-value ?act act-dur))

  (bind ?finish-i (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i ?floor-i)))
  (bind ?finish-j (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i ?floor-j)))
  (bind ?start-j (- ?finish-j (get-total-dur ?act ?act-dur ?floor-j ?floor-j)))

  (For ?new-res-demand in-slot-values-of ?act new-res-demand Do
    (bind ?res (nth$ ?new-res-demand 1))
    (assert (schema ?res (act-using-res ?act)))
  )

  (retract-schema-value act-set oss ?act)
  (retract-schema-value act-set eas ?act)
  (retract-schema-value act-set ias ?act)

  (modify-schema-value ?act scheduling-method Activity-Scheduled)

  (For ?succ-act in-slot-values-of ?act succ-act Do
    (IF (not (= ?succ-act none))
      THEN
         (put-schema-value ?succ-act prec-act-scheduled ?act)
         (IF (= (get-schema-value ?act workflow-dir) (get-schema-value ?succ-act workflow-dir))
           THEN
              (IF (< (get-schema-value ?succ-act es) ?finish-i)
                THEN (modify-schema-value ?succ-act es ?finish-i))
           ELSE
              (IF (< (get-schema-value ?succ-act es) ?finish-j)
                THEN (modify-schema-value ?succ-act es ?finish-j))
         )
      )
   )


   (assert (schema ?act (output-schedule (?floor-i ?floor-j ?act-dur ?start-i ?finish-i ?start-j ?finish-j))))


)



(def-art-fun Schedule-B (?act ?start-i ?floor-i ?floor-j)

  (bind ?act-dur (get-schema-value ?act act-dur))

  (bind ?finish-i (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i ?floor-i)))
  (bind ?finish-j (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i ?floor-j)))
  (bind ?start-j (- ?finish-j (get-total-dur ?act ?act-dur ?floor-j ?floor-j)))

  (For ?new-res-demand in-slot-values-of ?act new-res-demand Do
     (bind ?res (nth$ ?new-res-demand 1))
     (assert (schema ?res (act-using-res ?act)))
  )


  (IF (or (and (= ?floor-j ?*to-floor*)
              (= (get-schema-value ?act workflow-dir) UP))
          (and (= ?floor-j ?*from-floor*)
              (= (get-schema-value ?act workflow-dir) DOWN)))

     THEN
        (retract-schema-value act-set oss ?act)
        (retract-schema-value act-set eas ?act)
        (retract-schema-value act-set ias ?act)

        (slotd ?act from-floor)
        (slotd ?act to-floor)
        (slotd ?act floors-per-segment)
        (slotd ?act floors-last-segment)

        (modify-schema-value ?act scheduling-method Activity-Scheduled)

        (For ?succ-act in-slot-values-of ?act succ-act Do
           (IF (not (= ?succ-act none))
             THEN
                (put-schema-value ?succ-act prec-act-scheduled ?act)
                (IF (and (not (= (get-schema-value ?act workflow-dir) (get-schema-value ?succ-act workflow-dir)))
                         (< (get-schema-value ?succ-act es) ?finish-j))
                   THEN (modify-schema-value ?succ-act es ?finish-j)
                )
        .
            )
        )

    ELSE
        (retract-schema-value act-set oss ?act)
        (retract-schema-value act-set eas ?act)

        (modify-schema-value ?act es ?finish-j)
        (modify-schema-value ?act act-dur (get-schema-value ?act nor-act-dur))
        (modify-schema-value ?act scheduling-method Verify-Horizontal-Logic-Constraints-B)

        (IF (= (get-schema-value ?act workflow-dir) UP)
          THEN
            (modify-schema-value ?act from-floor (+ ?floor-j 1))

            (IF (= ?floor-i ?*from-floor*) THEN
              (For ?succ-act in-slot-values-of ?act succ-act Do
                (IF (not (= ?succ-act none))
                  THEN
                    (IF (and (= (get-schema-value ?act workflow-dir) (get-schema-value ?succ-act workflow-dir))
                             (< (get-schema-value ?succ-act es) ?finish-i))
                      THEN (modify-schema-value ?succ-act es ?finish-i)
                    )
                )
              )
            )

          ELSE
            (modify-schema-value ?act from-floor (- ?floor-j 1))

            (IF (= ?floor-i ?*to-floor*) THEN
              (For ?succ-act in-slot-values-of ?act succ-act Do
                (IF (not (= ?succ-act none))
                  THEN
                    (IF (and (= (get-schema-value ?act workflow-dir) (get-schema-value ?succ-act workflow-dir))
                             (< (get-schema-value ?succ-act es) ?finish-i))
                      THEN (modify-schema-value ?succ-act es ?finish-i)
                    )
                )
              )
            )
        )
)


(IF (slot-null ?act output-schedule)
  THEN
    (assert (schema ?act (output-schedule (?floor-i ?floor-j ?act-dur ?start-i ?finish-i ?start-j ?finish-j))))
  ELSE
    (FOR ?output-schedule in-slot-values-of ?act output-schedule DO

      (IF (and (= (abs (- ?floor-i (nth$ ?output-schedule 2))) 1)
               (= ?act-dur (nth$ ?output-schedule 3))
               (= ?start-i (nth$ ?output-schedule 7)))
        THEN
          (bind ?floor-i (nth$ ?output-schedule 1))
```

```
                (bind ?start-i (nth$ ?output-schedule 4))
                (bind ?finish-i (nth$ ?output-schedule 5))

                (retract-schema-value ?act output-schedule ?output-schedule)
            )
          )
          (assert (schema ?act (output-schedule (?floor-i ?floor-j ?act-dur ?start-i ?finish-i ?start-j ?finish-j))))
  )

); end function



(def-art-fun Modify-Resource-Demand-Pool (?act ?act-dur)

  (IF (not (slot-null ?act new-res-demand))
    THEN (retract-all-schema-values ?act new-res-demand)
  )

  (IF (= ?act-dur (get-schema-value ?act nor-act-dur))
    THEN
      (FOR ?act-res-demand in-slot-values-of ?act res-demand DO
        (assert (schema ?act (new-res-demand ?act-res-demand)))
      )
  )


  (IF (> ?act-dur (get-schema-value ?act nor-act-dur))
    THEN
      (FOR ?act-res-demand in-slot-values-of ?act res-demand DO
        (IF (= (nth$ ?act-res-demand 2) material)
          THEN (assert (schema ?act (new-res-demand ?act-res-demand)))
          ELSE
              (bind ?res-ratio (/ (get-schema-value ?act nor-act-dur) ?act-dur))
              (bind ?new-res-mag (ceiling (* ?res-ratio (nth$ ?act-res-demand 3))))
              (assert (schema ?act (new-res-demand ( =(nth$ ?act-res-demand 1) =(nth$ ?act-res-demand 2)
                                                                          ?new-res-mag))))
        )
      )
  )
) ;end func modify resource



(def-art-fun Workspace-Modify (?act ?start-i ?floor-i ?floor-j ?max-lag-1 ?max-lag-2)

  (bind ?act-dur (get-schema-value ?act act-dur))
  (bind ?max-req-dur ?*dur*)

  (IF (> ?max-lag-1 0)
    THEN
      (IF (= (get-schema-value ?act continuity-class) A)
        THEN (modify-schema-value ?act scheduling-method Verify-Horizontal-Logic-Constraints-A)
        ELSE (modify-schema-value ?act scheduling-method Verify-Horizontal-Logic-Constraints-B))

      (modify-schema-value ?act act-dur (get-schema-value ?act nor-act-dur))
      (modify-schema-value ?act es (+ ?start-i ?max-lag-1))
      (retract-schema-value act-set oss ?act)
      (retract-schema-value act-set eas ?act)
  )


  (IF (and (<= ?max-lag-1 0)
        (> ?max-lag-2 0))
   THEN
    (IF (<= (get-schema-value ?act max-act-dur) (get-schema-value ?act nor-act-dur))
     THEN
       (modify-schema-value ?act scheduling-method Verify-Resource-Constraints)
       (modify-schema-value ?act es (+ ?start-i ?max-lag-2))
       (retract-schema-value act-set oss ?act)
       (retract-schema-value act-set eas ?act)

     ELSE
      (IF (= ?act-dur (get-schema-value ?act max-act-dur))
       THEN
         (IF (= (get-schema-value ?act continuity-class) A)
           THEN (modify-schema-value ?act scheduling-method Verify-Horizontal-Logic-Constraints-A)
           ELSE (modify-schema-value ?act scheduling-method Verify-Horizontal-Logic-Constraints-B))

         (modify-schema-value ?act es (+ ?start-i ?max-lag-2))
         (modify-schema-value ?act act-dur (get-schema-value ?act nor-act-dur))
         (retract-schema-value act-set oss ?act)
         (retract-schema-value act-set eas ?act)

       ELSE
         (IF (<= ?max-req-dur (get-schema-value ?act max-act-dur))
           THEN
              (modify-schema-value ?act act-dur ?max-req-dur)
           ELSE
              (modify-schema-value ?act act-dur (get-schema-value ?act max-act-dur))
         )
      )

    )

  )

) ;end function



(def-art-fun Workspace-Lag-2 (?act ?start-i ?floor-i ?floor-j ?act-sharing-space)

  (bind ?act-dur (get-schema-value ?act act-dur))
  (bind ?act-workflow-dir (get-schema-value ?act workflow-dir))

  (bind ?lag-2 0)

  (IF (= ?act-workflow-dir up)

    THEN
```

```
        (FOR ?output-schedule in-slot-values-of ?act-sharing-space output-schedule DO
           (IF (and (not (= ?floor-i ?floor-j))
                    (>= ?floor-j (nth$ ?output-schedule 1))
                    (<= ?floor-j (nth$ ?output-schedule 2)))
             THEN
                (bind ?act-start-j (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (- ?floor-j 1))))
                (bind ?act-sharing-space-finish-j (+ (nth$ ?output-schedule 4) (get-total-dur ?act-sharing-space
                                                      (nth$ ?output-schedule 3) (nth$ ?output-schedule 1) ?floor-j)))
                (bind ?difference (- ?act-sharing-space-finish-j ?act-start-j))
                (IF (> ?difference ?lag-2) THEN (bind ?lag-2 ?difference))

                (bind ?req-dur (/ (- ?act-sharing-space-finish-j ?start-i)
                                  (- ?floor-j ?floor-i)))
                (IF (> ?req-dur ?*dur*) THEN (bind ?*dur* ?req-dur))
           )


           (IF (and (not (= ?floor-i ?floor-j))
                    (> ?floor-j (nth$ ?output-schedule 1))
                    (< ?floor-i (nth$ ?output-schedule 1)))
             THEN
                (bind ?floor-x (nth$ ?output-schedule 1))
                (bind ?act-start-x (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (- ?floor-x 1))))
                (bind ?act-sharing-space-finish-x (nth$ ?output-schedule 5))
                (bind ?difference (- ?act-sharing-space-finish-x ?act-start-x))
                (IF (> ?difference ?lag-2) THEN (bind ?lag-2 ?difference))

                (bind ?req-dur (/ (- ?act-sharing-space-finish-x ?start-i)
                                  (- ?floor-x ?floor-i)))
                (IF (> ?req-dur ?*dur*) THEN (bind ?*dur* ?req-dur))
           )

           (IF (and (not (= ?floor-i ?floor-j))
                    (< ?floor-i (nth$ ?output-schedule 2))
                    (> ?floor-j (nth$ ?output-schedule 2)))
             THEN
                (bind ?floor-y (nth$ ?output-schedule 2))
                (bind ?act-start-y (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (- ?floor-y 1))))
                (bind ?act-sharing-space-finish-y (nth$ ?output-schedule 7))
                (bind ?difference (- ?act-sharing-space-finish-y ?act-start-y))
                (IF (> ?difference ?lag-2) THEN (bind ?lag-2 ?difference))

                (bind ?req-dur (/ (- ?act-sharing-space-finish-y ?start-i)
                                  (- ?floor-y ?floor-i)))
                (IF (> ?req-dur ?*dur*) THEN (bind ?*dur* ?req-dur))
           )
        )
    )


    (IF (= ?act-workflow-dir down)

      THEN
        (FOR ?output-schedule in-slot-values-of ?act-sharing-space output-schedule DO
            (IF (and (not (= ?floor-i ?floor-j))
                    (<= ?floor-j (nth$ ?output-schedule 1))
                    (>= ?floor-j (nth$ ?output-schedule 2)))
              THEN
                (bind ?act-start-j (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (+ ?floor-j 1))))
                (bind ?act-sharing-space-finish-j (+ (nth$ ?output-schedule 4)
                        (get-total-dur ?act-sharing-space (nth$ ?output-schedule 3) (nth$ ?output-schedule 1) ?floor-j)))
                (bind ?difference (- ?act-sharing-space-finish-j ?act-start-j))
                (IF (> ?difference ?lag-2) THEN (bind ?lag-2 ?difference))

                (bind ?req-dur (/ (- ?act-sharing-space-finish-j ?start-i)
                                (- ?floor-i ?floor-j)))
                (IF (> ?req-dur ?*dur*) THEN (bind ?*dur* ?req-dur))
            )

            (IF (and (not (= ?floor-i ?floor-j))
                    (<= ?floor-j (nth$ ?output-schedule 1))
                    (> ?floor-i (nth$ ?output-schedule 1)))
              THEN
                (bind ?floor-x (nth$ ?output-schedule 1))
                (bind ?act-start-x (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (+ ?floor-x 1))))
                (bind ?act-sharing-space-finish-x (nth$ ?output-schedule 5))
                (bind ?difference (- ?act-sharing-space-finish-x ?act-start-x))
                (IF (> ?difference ?lag-2) THEN (bind ?lag-2 ?difference))

                (bind ?req-dur (/ (- ?act-sharing-space-finish-x ?start-i)
                                (- ?floor-i ?floor-x)))
                (IF (> ?req-dur ?*dur*) THEN (bind ?*dur* ?req-dur))
            )

            (IF (and (not (= ?floor-i ?floor-j))
                    (> ?floor-i (nth$ ?output-schedule 2))
                   (< ?floor-j (nth$ ?output-schedule 2)))
              THEN
                (bind ?floor-y (nth$ ?output-schedule 2))
                (bind ?act-start-y (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (+ ?floor-y 1))))
                (bind ?act-sharing-space-finish-y (nth$ ?output-schedule 7))
                (bind ?difference (- ?act-sharing-space-finish-y ?act-start-y))
                (IF (> ?difference ?lag-2) THEN (bind ?lag-2 ?difference))

                (bind ?req-dur (/ (- ?act-sharing-space-finish-y ?start-i)
                                (- ?floor-i ?floor-y)))
                (IF (> ?req-dur ?*dur*) THEN (bind ?*dur* ?req-dur))
            )
        )
  )

  (bind ?lag-2 ?lag-2)

) ;end function


(def-art-fun Workspace-Lag-1 (?act ?start-i ?floor-i ?floor-j ?act-sharing-space)

  (FOR ?output-schedule in-slot-values-of ?act-sharing-space output-schedule DO
    (IF (or (and (>= ?floor-i (nth$ ?output-schedule 1))
                 (<= ?floor-i (nth$ ?output-schedule 2)))
            (and (<= ?floor-i (nth$ ?output-schedule 1))
                 (>= ?floor-i (nth$ ?output-schedule 2))))
      THEN
         (bind ?act-sharing-space-finish-i (+ (nth$ ?output-schedule 4) (get-total-dur ?act-sharing-space
                                              (nth$ ?output-schedule 3) (nth$ ?output-schedule 1) ?floor-i)))
         (bind ?lag-1 (- ?act-sharing-space-finish-i ?start-i))
      )
  )

  (bind ?lag-1 ?lag-1)

)
```

(def-art-fun Verify-Work-Space-Constraints (?act ?start-i ?floor-i ?floor-j)

```
 (bind ?block-1 (get-schema-value ?act mpw-eqp-block))
 (bind ?block-2 (get-schema-value ?act mat-block))

 (IF (or (not (= ?block-1 none))
         (not (= ?block-2 none))) THEN

  (bind ?act-dur (get-schema-value ?act act-dur))
  (bind ?class (get-schema-value ?act space-demand-class))

  (retract-all-schema-values ?act act-sharing-space)

  (IF (= (get-schema-value ?act workflow-dir) up)
    THEN
       (bind ?val-1 1)
       (bind ?x ?floor-i)
       (bind ?y ?floor-j)
    ELSE
       (bind ?val-1 -1)
       (bind ?x ?floor-j)
       (bind ?y ?floor-i)
  )

  (FOR ?floor FROM ?x TO ?y DO

     (IF (= ?floor ?floor-i)
       THEN (bind ?floor-start ?start-i)
       ELSE (bind ?floor-start (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (- ?floor ?val-1))))
     )
     (bind ?floor-finish (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i ?floor)))


     (FOR ?other-act in-schema-children-of activity DO

        (IF (= (get-schema-value ?other-act workflow-dir) UP)
          THEN (bind ?val-2 1)
```

```
            ELSE (bind ?val-2 -1)
          )

          (IF (and (not (= ?other-act ?act))
                   (not (slot-null ?other-act output-schedule))
                   (not (schema-value-p ?act act-sharing-space ?other-act))
                   (not (schema-value-p ?act prec-act ?other-act))
                   (or (= (get-schema-value ?other-act mpw-eqp-block) ?block-1)
                      (= (get-schema-value ?other-act mat-block) ?block-2)))

           THEN
              (FOR ?output-schedule in-slot-values-of ?other-act output-schedule DO
                 (IF (and (not (schema-value-p ?act act-sharing-space ?other-act))
                          (or (and (>= ?floor (nth$ ?output-schedule 1))
                                   (<= ?floor (nth$ ?output-schedule 2)))
                              (and (<= ?floor (nth$ ?output-schedule 1))
                                   (>= ?floor (nth$ ?output-schedule 2))))
                          (or (not (> (nth$ ?output-schedule 4) ?floor-finish))
                              (not (< (nth$ ?output-schedule 7) ?floor-start))))
                   THEN
                      (IF (= ?floor (nth$ ?output-schedule 1))
                         THEN (bind ?other-floor-start (nth$ ?output-schedule 4))
                         ELSE (bind ?other-floor-start (+ (nth$ ?output-schedule 4) (get-total-dur  ?other-act
                                                   (nth$ ?output-schedule 3) (nth$ ?output-schedule 1) (- ?floor ?val-2))))
                      )
                      (bind ?other-floor-finish (+ (nth$ ?output-schedule 4) (get-total-dur ?other-act
                                                   (nth$ ?output-schedule 3) (nth$ ?output-schedule 1) ?floor)))

                      (IF (and (< ?floor-start ?other-floor-finish)
                               (> ?floor-finish ?other-floor-start))
                         THEN
                            (assert (schema ?act (act-sharing-space ?other-act)))
                      )
                 )
              )
          )
      )
   )

   (bind ?max-lag-1 0)
   (bind ?max-lag-2 0)


   (IF (and (not (slot-null ?act act-sharing-space))
            (= ?class A))  THEN

      (IF (not (slot-null ?act act-sharing-space)) THEN
        (bind ?*dur* 0)
        (FOR ?act-sharing-space in-slot-values-of ?act act-sharing-space DO
          (IF (= (get-schema-value ?act workflow-dir) (get-schema-value ?act-sharing-space workflow-dir))
            THEN
               (bind ?lag-1 (Workspace-Lag-1 ?act ?start-i ?floor-i ?floor-j ?act-sharing-space))
               (IF (> ?lag-1 ?max-lag-1) THEN (bind ?max-lag-1 ?lag-1))

               (bind ?lag-2 (Workspace-Lag-2 ?act ?start-i ?floor-i ?floor-j ?act-sharing-space))
               (IF (> ?lag-2 ?max-lag-2) THEN (bind ?max-lag-2 ?lag-2))
          ELSE
             (bind ?lag-1 (Workspace-Lag-1 ?act ?start-i ?floor-i ?floor-j ?act-sharing-space))
             (IF (> ?lag-1 ?max-lag-1) THEN (bind ?max-lag-1 ?lag-1))
       )
     )
   )

   (Workspace-Modify ?act ?start-i ?floor-i ?floor-j ?max-lag-1 ?max-lag-2)

) ;end 1st If



(IF (and (not (slot-null ?act act-sharing-space))
        (not (= ?class A))) THEN

   (bind ?*dur* 0)
   (FOR ?act-sharing-space in-slot-values-of ?act act-sharing-space DO
      (IF (= (get-schema-value ?act-sharing-space space-demand-class) A) THEN

          (IF (= (get-schema-value ?act workflow-dir) (get-schema-value ?act-sharing-space workflow-dir))
            THEN
               (bind ?lag-1 (Workspace-Lag-1 ?act ?start-i ?floor-i ?floor-j ?act-sharing-space))
               (IF (> ?lag-1 ?max-lag-1) THEN (bind ?max-lag-1 ?lag-1))

               (bind ?lag-2 (Workspace-Lag-2 ?act ?start-i ?floor-i ?floor-j ?act-sharing-space))
               (IF (> ?lag-2 ?max-lag-2) THEN (bind ?max-lag-2 ?lag-2))
            ELSE
               (bind ?lag-1 (Workspace-Lag-1 ?act ?start-i ?floor-i ?floor-i ?act-sharing-space))
               (IF (> ?lag-1 ?max-lag-1) THEN (bind ?max-lag-1 ?lag-1))
          )
      )
   )

   (IF (or (> ?max-lag-1 0)
           (> ?max-lag-2 0))
      THEN
          (Workspace-Modify ?act ?start-i ?floor-i ?floor-j ?max-lag-1 ?max-lag-2)
   )


   (IF (and (<= ?max-lag-1 0)
          (<= ?max-lag-2 0)) THEN

   (bind ?space-block-1 (get-schema-value ?block-1 space-available))
   (bind ?space-block-2 (get-schema-value ?block-2 space-available))

   (bind ?SD-1 0)
   (bind ?SD-2 0)
   (FOR ?res in-slot-values-of ?act new-res-demand DO
      (IF (= (nth$ ?res 2) material)
         THEN (bind ?SD-2 (+ ?SD-2 (get-schema-value (nth$ ?res 1) space-demand)))
         ELSE (bind ?SD-1 (+ ?SD-1 (* (nth$ ?res 3) (get-schema-value (nth$ ?res 1) space-demand))))
      )
   )
```

```
(bind ?total-segment-dur 0)

(bind ?block1-max-lag-1 0)
(bind ?block1-max-lag-2 0)
(bind ?block2-max-lag-1 0)
(bind ?block2-max-lag-2 0)
(bind ?min-floor-lag 1000000)

(FOR ?floor FROM ?x TO ?y DO

  (bind ?usage-block-1 0)
  (bind ?usage-block-2 0)

  (IF (= ?floor ?floor-i)
    THEN (bind ?floor-start ?start-i)
    ELSE (bind ?floor-start ?floor-finish)
  )
  (bind ?floor-finish (+ ?floor-start (get-total-dur ?act ?act-dur ?floor ?floor)))

  (FOR ?act-sharing-space in-slot-values-of ?act act-sharing-space DO

     (IF (= (get-schema-value ?act-sharing-space workflow-dir) UP)
       THEN (bind ?val-2 1)
       ELSE (bind ?val-2 -1)
     )


     (FOR ?output-schedule in-slot-values-of ?act-sharing-space output-schedule DO
        (IF (or (and (>= ?floor (nth$ ?output-schedule 1))
                    (<= ?floor (nth$ ?output-schedule 2)))
                (and (<= ?floor (nth$ ?output-schedule 1))
                    (>= ?floor (nth$ ?output-schedule 2))))
           THEN
            (IF (= ?floor (nth$ ?output-schedule 1))
              THEN (bind ?other-floor-start (nth$ ?output-schedule 4))
              ELSE (bind ?other-floor-start (+ (nth$ ?output-schedule 4) (get-total-dur  ?act-sharing-space
                                   (nth$ ?output-schedule 3) (nth$ ?output-schedule 1) (- ?floor ?val-2))))
            )
            (bind ?other-floor-finish (+ (nth$ ?output-schedule 4) (get-total-dur ?act-sharing-space
                                       (nth$ ?output-schedule 3) (nth$ ?output-schedule 1) ?floor)))

            (IF (and (< ?floor-start ?other-floor-finish)
                   (> ?floor-finish ?other-floor-start))
               THEN
                 (bind ?other-SD-1 0)
                 (bind ?other-SD-2 0)
                 (bind ?act-sharing-space-dur (nth$ ?output-schedule 3))
                 (Modify-Resource-Demand-Pool ?act-sharing-space ?act-sharing-space-dur)
                 (FOR ?res in-slot-values-of ?act-sharing-space new-res-demand DO
                    (IF (= (nth$ ?res 2) material)
                       THEN (bind ?other-SD-2 (+ ?other-SD-2 (get-schema-value (nth$ ?res 1)
                                        space-demand)))
                       ELSE (bind ?other-SD-1 (+ ?other-SD-1 (* (nth$ ?res 3) (get-schema-value (nth$ ?res
                                              1) space-demand))))
                    )
                 )
```

```
                    (IF (and (= (get-schema-value ?act-sharing-space space-demand-class) C)
                             (>= ?floor-start (+ ?other-floor-start (/ (* (nth$ ?output-schedule 3) 2) 3))))
                      THEN
                          (bind ?other-SD-2 (/ ?other-SD-2 2))
                    )

                    (IF (= (get-schema-value ?act-sharing-space mpw-eqp-block) ?block-1)
                       THEN (bind ?usage-block-1 (+ ?usage-block-1 ?other-SD-1)))

                    (IF (= (get-schema-value ?act-sharing-space mpw-eqp-block) ?block-2)
                       THEN (bind ?usage-block-2 (+ ?usage-block-2 ?other-SD-1)))

                    (IF (= (get-schema-value ?act-sharing-space mat-block) ?block-1)
                       THEN (bind ?usage-block-1 (+ ?usage-block-1 ?other-SD-2)))

                    (IF (= (get-schema-value ?act-sharing-space mat-block) ?block-2)
                       THEN (bind ?usage-block-2 (+ ?usage-block-2 ?other-SD-2)))


                    (bind ?floor-lag (- ?other-floor-finish ?floor-start))
                    (IF (< ?floor-lag ?min-floor-lag) THEN (bind ?min-floor-lag ?floor-lag))

                    (IF (not (= ?floor ?floor-i))
                       THEN
                           (bind ?req-dur (/ (- ?other-floor-finish ?start-i)
                                             (abs (- ?floor ?floor-i))))
                           (IF (> ?req-dur ?*dur*) THEN (bind ?*dur* ?req-dur))
                    )
                )
            )
        )
)

(IF (= ?block-1 ?block-2)
   THEN
       (bind ?usage-block-1 (+ ?usage-block-1 ?SD-2))
)

(IF (< ?space-block-2 (+ ?SD-2 ?usage-block-2))
   THEN
      (IF (= ?floor ?floor-i)
         THEN (IF (> ?min-floor-lag ?block2-max-lag-1) THEN (bind ?block2-max-lag-1 ?min-floor-lag))
         ELSE (IF (> ?min-floor-lag ?block2-max-lag-2) THEN (bind ?block2-max-lag-2 ?min-floor-lag))
      )
      (bind ?total-segment-dur (+ ?total-segment-dur ?act-dur))

   ELSE
      (IF (< ?space-block-1 (+ ?SD-1 ?usage-block-1))
         THEN
            (IF (= ?floor ?floor-i)
               THEN (IF (> ?min-floor-lag ?block1-max-lag-1) THEN (bind ?block1-max-lag-1 ?min-floor-lag))
               ELSE (IF (> ?min-floor-lag ?block1-max-lag-2) THEN (bind ?block1-max-lag-2 ?min-floor-lag))
            )

            (bind ?current-space (- ?space-block-1 ?usage-block-1))
```

```
                (IF (> ?current-space 0)
                  THEN
                     (bind ?SCF (/ ?SD-1 ?current-space))
                  ELSE
                     (bind ?SCF 10000)
                )

                (FOR ?curve in-slot-values-of Productivity-SCF saturation-curve DO
                   (IF (and (>  ?SCF (nth$ ?curve 1))
                            (<= ?SCF (nth$ ?curve 2)))
                      THEN
                         (bind ?mod-dur (* ?act-dur (nth$ ?curve 3)))
                         (bind ?floor-finish (+ ?floor-start ?mod-dur))
                         (bind ?total-segment-dur (+ ?total-segment-dur ?mod-dur))
                   )
                )
              ELSE
                (bind ?total-segment-dur (+ ?total-segment-dur ?act-dur))
            )
        )

      ) ;end for for each floor


      (bind ?ave-mod-dur (/ ?total-segment-dur (+ (abs (- ?floor-j ?floor-i)) 1)))

      (IF (or (> ?block2-max-lag-1 0)
              (> ?block2-max-lag-2 0))
        THEN
           (Workspace-Modify ?act ?start-i ?floor-i ?floor-j ?block2-max-lag-1 ?block2-max-lag-2)

        ELSE
           (IF (<= ?ave-mod-dur (* 1.5 ?act-dur))
             THEN
               (modify-schema-value ?act act-dur ?ave-mod-dur)

               (IF (= (get-schema-value ?act continuity-class) A)
                  THEN (modify-schema-value ?act scheduling-method Schedule-A)
                  ELSE (modify-schema-value ?act scheduling-method Schedule-B)
               )
             ELSE
               (Workspace-Modify ?act ?start-i ?floor-i ?floor-j ?block1-max-lag-1 ?block1-max-lag-2)
           )
      )
    )
  ) ;end 2nd IF

) ;end main if


(IF  (slot-null ?act act-sharing-space) THEN

      (IF (= (get-schema-value ?act continuity-class) A)
         THEN (modify-schema-value ?act scheduling-method Schedule-A)
         ELSE (modify-schema-value ?act scheduling-method Schedule-B)
      )
```

```
 ) ; end 3rd IF


) ;end main function



(def-art-fun Verify-Resource-Constraints (?act ?start-i ?floor-i ?floor-j)

  (bind ?act-dur (get-schema-value ?act act-dur))
  (bind ?act-workflow-dir (get-schema-value ?act workflow-dir))

  (bind ?finish-j (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i ?floor-j)))

  (Modify-Resource-Demand-Pool ?act ?act-dur)

  (bind ?flag yes)
  (bind ?act-min-es 0)

  (FOR ?act-res-demand in-slot-values-of ?act new-res-demand DO

    (bind ?res (nth$ ?act-res-demand 1))
    (bind ?res-mag (nth$ ?act-res-demand 3))
    (bind ?usage 0)
    (bind ?min-es 10000)

    (IF (not (slot-null ?res act-using-res)) THEN
      (FOR ?act-using-res in-slot-values-of ?res act-using-res DO

          (FOR ?output-schedule in-slot-values-of ?act-using-res output-schedule DO

              (IF (and (< (nth$ ?output-schedule 4) ?finish-j)
                       (> (nth$ ?output-schedule 7) ?start-i))

                  THEN
                      (bind ?act-using-res-dur (nth$ ?output-schedule 3))
                      (Modify-Resource-Demand-Pool ?act-using-res ?act-using-res-dur)

                      (FOR ?other-res-demand in-slot-values-of ?act-using-res new-res-demand DO
                          (IF (= (nth$ ?other-res-demand 1) ?res)
                            THEN (bind ?usage (+ ?usage (nth$ ?other-res-demand 3)))
                          )
                      )

                      (IF (< (nth$ ?output-schedule 7) ?min-es)
                         THEN (bind ?min-es (nth$ ?output-schedule 7))
                      )
              )
          )
      )
    )

    (IF (> =(+ ?usage ?res-mag)
          (get-schema-value (nth$ ?act-res-demand 1) res-mag))
       THEN
           (bind ?flag no)
```

```
            (IF (> ?min-es ?act-min-es)
              THEN (bind ?act-min-es ?min-es)
            )
    )

  ) ;end main for


  (IF (= ?flag yes)
     THEN
         (modify-schema-value ?act scheduling-method Verify-Work-Space-Constraints)
     ELSE
         (retract-schema-value act-set oss ?act)
         (retract-schema-value act-set eas ?act)
         (modify-schema-value ?act es ?act-min-es)
         (modify-schema-value ?act act-dur (get-schema-value ?act nor-act-dur))

         (IF (= (get-schema-value ?act continuity-class) A)
           THEN (modify-schema-value ?act scheduling-method Verify-Horizontal-Logic-Constraints-A)
           ELSE (modify-schema-value ?act scheduling-method Verify-Horizontal-Logic-Constraints-B)
         )
  )


); end function



(def-art-fun Verify-Vertical-Logic-Constraints (?act ?start-i ?floor-i ?floor-j)


  (bind ?lag-1 0)
  (bind ?max-lag-2 0)
  (bind ?max-req-dur 0)

  (bind ?ver-act (get-schema-value ?act ver-logic-act))

  (IF (not (= ?ver-act NONE))
     THEN
        (bind ?act-dur (get-schema-value ?act act-dur))
        (bind ?ver-lag (get-schema-value ?act ver-logic-lag))


        (FOR ?ver-act-output-schedule in-slot-values-of ?ver-act output-schedule DO

           (IF (and (>= (+ ?floor-i ?ver-lag) (nth$ ?ver-act-output-schedule 1))
                    (<= (+ ?floor-i ?ver-lag) (nth$ ?ver-act-output-schedule 2)))
             THEN
                 (bind ?ver-act-finish (+ (nth$ ?ver-act-output-schedule 4)
                                          (get-total-dur ?ver-act (nth$ ?ver-act-output-schedule 3)
                                            (nth$ ?ver-act-output-schedule 1) (+ ?floor-i ?ver-lag))))
                 (bind ?lag-1 (- ?ver-act-finish ?start-i))
           )
        (IF (and (not (= ?floor-i ?floor-j))
                 (>= (+ ?floor-j ?ver-lag) (nth$ ?ver-act-output-schedule 1))
                 (<= (+ ?floor-j ?ver-lag) (nth$ ?ver-act-output-schedule 2)))
          THEN
              (bind ?ver-act-finish (+ (nth$ ?ver-act-output-schedule 4)
                                       (get-total-dur ?ver-act (nth$ ?ver-act-output-schedule 3)
                                         (nth$ ?ver-act-output-schedule 1) (+ ?floor-j ?ver-lag))))
              (bind ?start-j (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (- ?floor-j 1))))
              (bind ?lag-2 (- ?ver-act-finish ?start-j))
              (IF (> ?lag-2 ?max-lag-2) THEN (bind ?max-lag-2 ?lag-2))
              (bind ?req-dur (/ (- ?ver-act-finish ?start-i)
                                (- ?floor-j ?floor-i)))
              (IF (> ?req-dur ?max-req-dur) THEN (bind ?max-req-dur ?req-dur))
        )


        (IF (and (not (= ?floor-i ?floor-j))
                 (not (= (nth$ ?ver-act-output-schedule 1) ?*from-floor*))
                 (> (- (nth$ ?ver-act-output-schedule 1) ?ver-lag) ?floor-i)
                 (< (- (nth$ ?ver-act-output-schedule 1) ?ver-lag) ?floor-j))
          THEN
             (bind ?act-start (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i
                                        (- (nth$ ?ver-act-output-schedule 1) ?ver-lag 1))))
             (bind ?lag-2 (- (nth$ ?ver-act-output-schedule 5) ?act-start))
             (IF (> ?lag-2 ?max-lag-2) THEN (bind ?max-lag-2 ?lag-2))

             (bind ?req-dur (/ (- (nth$ ?ver-act-output-schedule 5) ?start-i)
                               (- (nth$ ?ver-act-output-schedule 1) ?ver-lag ?floor-i)))
             (IF (> ?req-dur ?max-req-dur) THEN (bind ?max-req-dur ?req-dur))
        )


        (IF (and (not (= ?floor-i ?floor-j))
                 (> (- (nth$ ?ver-act-output-schedule 2) ?ver-lag) ?floor-i)
                 (< (- (nth$ ?ver-act-output-schedule 2) ?ver-lag) ?floor-j))
          THEN
             (bind ?act-start (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i
                                        (- (nth$ ?ver-act-output-schedule 2) ?ver-lag 1))))
             (bind ?lag-2 (- (nth$ ?ver-act-output-schedule 7) ?act-start))
             (IF (> ?lag-2 ?max-lag-2) THEN (bind ?max-lag-2 ?lag-2))

             (bind ?req-dur (/ (- (nth$ ?ver-act-output-schedule 7) ?start-i)
                               (- (nth$ ?ver-act-output-schedule 2) ?ver-lag ?floor-i)))
             (IF (> ?req-dur ?max-req-dur) THEN (bind ?max-req-dur ?req-dur))
        )
      )
)


(IF (and (<= ?lag-1 0)
         (<= ?max-lag-2 0))
  THEN
     (modify-schema-value ?act scheduling-method Verify-Resource-Constraints)
)
```

```
  (IF (> ?lag-1 0)
    THEN
       (IF (= (get-schema-value ?act continuity-class) A)
         THEN (modify-schema-value ?act scheduling-method Verify-Horizontal-Logic-Constraints-A)
         ELSE (modify-schema-value ?act scheduling-method Verify-Horizontal-Logic-Constraints-B)
       )

       (modify-schema-value ?act act-dur (get-schema-value ?act nor-act-dur))
       (modify-schema-value ?act es (+ ?start-i ?lag-1))
       (retract-schema-value act-set eas ?act)
       (retract-schema-value act-set oss ?act)
  )

  (IF (and (<= ?lag-1 0)
          (> ?max-lag-2 0))
    THEN
       (IF (<= (get-schema-value ?act max-act-dur) (get-schema-value ?act nor-act-dur))
         THEN
            (modify-schema-value ?act scheduling-method Verify-Resource-Constraints)
            (modify-schema-value ?act es (+ ?start-i ?max-lag-2))
            (retract-schema-value act-set eas ?act)
            (retract-schema-value act-set oss ?act)

         ELSE
            (IF (= ?act-dur (get-schema-value ?act max-act-dur))
              THEN
                 (modify-schema-value ?act scheduling-method Verify-Resource-Constraints)
                 (modify-schema-value ?act es (+ ?start-i ?max-lag-2))
                 (retract-schema-value act-set eas ?act)
                 (retract-schema-value act-set oss ?act)
              ELSE
                 (IF (<= ?max-req-dur (get-schema-value ?act max-act-dur))
                   THEN
                      (modify-schema-value ?act act-dur ?max-req-dur)
                      (modify-schema-value ?act scheduling-method Verify-Resource-Constraints)
                   ELSE
                      (modify-schema-value ?act act-dur (get-schema-value ?act max-act-dur))
                 )
            )
       )
  )

) ;end function
```

```
(def-art-fun Verify-Horizontal-Logic-Constraints-A (?act ?start-i ?floor-i ?floor-j)

  (bind ?act-dur (get-schema-value ?act act-dur))
  (bind ?act-workflow-dir (get-schema-value ?act workflow-dir))


  (bind ?max-lag 0)
  (bind ?max-req-dur 0)
```

```
(FOR ?prec-act in-slot-values-of ?act prec-act DO

 (IF (and (not (= ?prec-act none))
        (= (get-schema-value ?prec-act continuity-class) A)
        (= (get-schema-value ?prec-act workflow-dir) ?act-workflow-dir))
   THEN
       (FOR ?output-schedule in-slot-values-of ?prec-act output-schedule DO

          (IF (> (nth$ ?output-schedule 3) ?act-dur) THEN

            (IF (= ?act-workflow-dir up)
              THEN  (bind ?cur-floor-j (- (nth$ ?output-schedule 2) 1))
              ELSE  (bind ?cur-floor-j (+ (nth$ ?output-schedule 2) 1))
            )

            (bind ?cur-start-j (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i ?cur-floor-j)))
            (bind ?lag (- (nth$ ?output-schedule 7) ?cur-start-j))
            (IF (> ?lag ?max-lag) THEN (bind ?max-lag ?lag))

            (bind ?req-dur (nth$ ?output-schedule 3))
            (IF (> ?req-dur ?max-req-dur) THEN (bind ?max-req-dur ?req-dur))
          )
       )
 )


 (IF (and (not (= ?prec-act none))
        (= (get-schema-value ?prec-act continuity-class) B)
        (= (get-schema-value ?prec-act workflow-dir) ?act-workflow-dir))
   THEN
       (FOR ?output-schedule in-slot-values-of ?prec-act output-schedule DO

          (IF (not (= (nth$ ?output-schedule 1) ?floor-i)) THEN

            (IF (= ?act-workflow-dir up)
              THEN  (bind ?cur-floor-j-1 (- (nth$ ?output-schedule 1) 1))
                    (bind ?cur-floor-j-2 (- (nth$ ?output-schedule 2) 1))
              ELSE  (bind ?cur-floor-j-1 (+ (nth$ ?output-schedule 1) 1))
                    (bind ?cur-floor-j-2 (+ (nth$ ?output-schedule 2) 1))
            )

            (bind ?cur-start-j-1 (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i ?cur-floor-j-1)))
            (bind ?lag (- (nth$ ?output-schedule 5) ?cur-start-j-1))
            (IF (> ?lag ?max-lag) THEN (bind ?max-lag ?lag))

            (bind ?req-dur (/ (- (nth$ ?output-schedule 5) ?start-i)
                              (abs (- ?floor-i (nth$ ?output-schedule 1)))))
            (IF (> ?req-dur ?max-req-dur) THEN (bind ?max-req-dur ?req-dur))

            (bind ?cur-start-j-2 (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i ?cur-floor-j-2)))
            (bind ?lag (- (nth$ ?output-schedule 7) ?cur-start-j-2))
            (IF (> ?lag ?max-lag) THEN (bind ?max-lag ?lag))

            (bind ?req-dur (/ (- (nth$ ?output-schedule 7) ?start-i)
                              (abs (- ?floor-i (nth$ ?output-schedule 2)))))
            (IF (> ?req-dur ?max-req-dur) THEN (bind ?max-req-dur ?req-dur))
            )
          )

      )

    )



    (IF (<= ?max-lag 0)
      THEN
        (modify-schema-value ?act scheduling-method Verify-Vertical-Logic-Constraints)
    )



    (IF (> ?max-lag 0)
      THEN
        (IF (<= (get-schema-value ?act max-act-dur) (get-schema-value ?act nor-act-dur))
          THEN
              (modify-schema-value ?act es (+ ?start-i ?max-lag))
              (modify-schema-value ?act scheduling-method Verify-Vertical-Logic-Constraints)
              (retract-schema-value act-set oss ?act)
              (retract-schema-value act-set eas ?act)
          ELSE
             (IF (= ?act-dur (get-schema-value ?act max-act-dur))
               THEN
                  (modify-schema-value ?act es (+ ?start-i ?max-lag))
                  (modify-schema-value ?act act-dur (get-schema-value ?act nor-act-dur))
                  (retract-schema-value act-set oss ?act)
                  (retract-schema-value act-set eas ?act)

               ELSE
                  (IF (<= ?max-req-dur (get-schema-value ?act max-act-dur))
                    THEN
                        (modify-schema-value ?act act-dur ?max-req-dur)
                        (modify-schema-value ?act scheduling-method Verify-Vertical-Logic-Constraints)
                    ELSE
                        (modify-schema-value ?act act-dur (get-schema-value ?act max-act-dur))
                  )
             )
        )
    )
  )
```

(def-art-fun Verify-Horizontal-Logic-Constraints-B (?act ?start-i ?floor-i ?floor-j)

```
    (bind ?act-dur (get-schema-value ?act act-dur))
    (bind ?act-workflow-dir (get-schema-value ?act workflow-dir))

    (bind ?total-floors (+ (abs (- ?floor-i ?floor-j)) 1))

    (bind ?max-lag-1 0)
    (bind ?max-lag-2 0)
```

```
(bind ?max-req-dur 0)

(FOR ?prec-act in-slot-values-of ?act prec-act DO
 (IF (not (= ?prec-act none)) THEN

  (IF (and (= ?act-workflow-dir up)
         (= ?act-workflow-dir (get-schema-value ?prec-act workflow-dir)))

   THEN
    (FOR ?output-schedule in-slot-values-of ?prec-act output-schedule DO

     (IF (and (<= ?floor-i (nth$ ?output-schedule 2))
             (>= ?floor-i (nth$ ?output-schedule 1)))
       THEN
          (bind ?prec-act-finish (+ (nth$ ?output-schedule 4)
                 (get-total-dur ?prec-act (nth$ ?output-schedule 3) (nth$ ?output-schedule 1) ?floor-i)))
          (bind ?lag-1 (- ?prec-act-finish ?start-i))
          (IF (> ?lag-1 ?max-lag-1) THEN (bind ?max-lag-1 ?lag-1))
     )

     (IF (and (not (= ?floor-i ?floor-j))
             (>= ?floor-j (nth$ ?output-schedule 1))
             (<= ?floor-j (nth$ ?output-schedule 2)))
       THEN
          (bind ?act-start (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (- ?floor-j 1))))
          (bind ?prec-act-finish (+ (nth$ ?output-schedule 4)
                              (get-total-dur ?prec-act (nth$ ?output-schedule 3) (nth$ ?output-schedule 1)
                                                                        ?floor-j)))
          (bind ?lag-2 (- ?prec-act-finish ?act-start))
          (IF (> ?lag-2 ?max-lag-2) THEN (bind ?max-lag-2 ?lag-2))

          (bind ?req-dur (/ (- ?prec-act-finish ?start-i)
                      (- ?floor-j ?floor-i)))
          (IF (> ?req-dur ?max-req-dur) THEN (bind ?max-req-dur ?req-dur))
     )


     (IF (and (not (= ?floor-i ?floor-j))
             (>= ?floor-j (nth$ ?output-schedule 1))
             (<= ?floor-j (nth$ ?output-schedule 2))
             (< ?floor-i (nth$ ?output-schedule 1)))
       THEN
          (bind ?act-start (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (- (nth$ ?output-schedule 1) 1))))
          (bind ?prec-act-finish (nth$ ?output-schedule 5))
          (bind ?lag-2 (- ?prec-act-finish ?act-start))
          (IF (> ?lag-2 ?max-lag-2) THEN (bind ?max-lag-2 ?lag-2))

          (bind ?req-dur (/ (- ?prec-act-finish ?start-i)
                      (- (nth$ ?output-schedule 1) ?floor-i)))
          (IF (> ?req-dur ?max-req-dur) THEN (bind ?max-req-dur ?req-dur))
     )

    )
  )

 (IF (and (= ?act-workflow-dir down)
        (= ?act-workflow-dir (get-schema-value ?prec-act workflow-dir)))

    THEN
     (FOR ?output-schedule in-slot-values-of ?prec-act output-schedule DO

      (IF (and (>= ?floor-i (nth$ ?output-schedule 2))
               (<= ?floor-i (nth$ ?output-schedule 1)))
        THEN
             (bind ?prec-act-finish (+ (nth$ ?output-schedule 4)
                      (get-total-dur ?prec-act (nth$ ?output-schedule 3) (nth$ ?output-schedule 1) ?floor-i)))
             (bind ?lag-1 (- ?prec-act-finish ?start-i))
             (IF (> ?lag-1 ?max-lag-1) THEN (bind ?max-lag-1 ?lag-1))
      )

      (IF (and (not (= ?floor-i ?floor-j))
               (>= ?floor-j (nth$ ?output-schedule 2))
               (<= ?floor-j (nth$ ?output-schedule 1)))
        THEN
            (bind ?act-start (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (+ ?floor-j 1))))
            (bind ?prec-act-finish (+ (nth$ ?output-schedule 4)
                                  (get-total-dur ?prec-act (nth$ ?output-schedule 3) (nth$ ?output-schedule 1)
                                                                                 ?floor-j)))
            (bind ?lag-2 (- ?prec-act-finish ?act-start))
            (IF (> ?lag-2 ?max-lag-2) THEN (bind ?max-lag-2 ?lag-2))

            (bind ?req-dur (/ (- ?prec-act-finish ?start-i)
                             (- ?floor-i ?floor-j)))
            (IF (> ?req-dur ?max-req-dur) THEN (bind ?max-req-dur ?req-dur))
      )

      (IF (and (not (= ?floor-i ?floor-j))
               (>= ?floor-j (nth$ ?output-schedule 2))
               (<= ?floor-j (nth$ ?output-schedule 1))
               (> ?floor-i (nth$ ?output-schedule 1)))
        THEN
            (bind ?act-start (+ ?start-i (get-total-dur ?act ?act-dur ?floor-i (+ (nth$ ?output-schedule 1) 1))))
            (bind ?prec-act-finish (nth$ ?output-schedule 5))
            (bind ?lag-2 (- ?prec-act-finish ?act-start))
            (IF (> ?lag-2 ?max-lag-2) THEN (bind ?max-lag-2 ?lag-2))

            (bind ?req-dur (/ (- ?prec-act-finish ?start-i)
                             (- ?floor-i (nth$ ?output-schedule 1))))
            (IF (> ?req-dur ?max-req-dur) THEN (bind ?max-req-dur ?req-dur))
      )

    )
  )

 )
)

(IF (and (<= ?max-lag-1 0)
         (<= ?max-lag-2 0))
  THEN
   (modify-schema-value ?act scheduling-method Verify-Vertical-Logic-Constraints)
)
```

```
  (IF (>  ?max-lag-1 0)
     THEN
         (modify-schema-value ?act es (+ ?start-i ?max-lag-1))
         (retract-schema-value act-set oss ?act)
         (retract-schema-value act-set eas ?act)
  )


  (IF (and (<= ?max-lag-1 0)
          (>  ?max-lag-2 0))
    THEN
      (IF (<= (get-schema-value ?act max-act-dur) (get-schema-value ?act nor-act-dur))
        THEN
            (modify-schema-value ?act es (+ ?start-i ?max-lag-2))
            (modify-schema-value ?act scheduling-method Verify-Vertical-Logic-Constraints)
            (retract-schema-value act-set oss ?act)
            (retract-schema-value act-set eas ?act)

        ELSE
          (IF (= ?act-dur (get-schema-value ?act max-act-dur))

            THEN
                (modify-schema-value ?act es (+ ?start-i ?max-lag-2))
                (modify-schema-value ?act act-dur (get-schema-value ?act nor-act-dur))
                (retract-schema-value act-set oss ?act)
                (retract-schema-value act-set eas ?act)

            ELSE
                (IF (<= ?max-req-dur (get-schema-value ?act max-act-dur))
                  THEN
                      (modify-schema-value ?act act-dur ?max-req-dur)
                      (modify-schema-value ?act scheduling-method Verify-Vertical-Logic-Constraints)

                  ELSE
                      (modify-schema-value ?act act-dur (get-schema-value ?act max-act-dur))
                )
          )

      )

  )

) ;end function
```

```
(def-external-data gen-data  114
  (act-num          0    4  :char  :symbol)
  (act-dscrp       4   30  :char  :string)
  (nor-act-dur    34    5  :char  :float)
  (max-act-dur    39    5  :char  :float)
  (temp-prec-act  44   30  :char  :string)
  (temp-succ-act  74   30  :char  :string)
  (es            104    5  :char  :float)
  (ls            109    5  :char  :float))

(def-external-data gen-buffer 114
  (all 0 114 :char))


(defglobal ?*gen-buf* = (alloc-buffer gen-buffer))
```

```
(defrule  RULE-1-Activity-Schemata-Creation-And-Activity-General-Data-Extraction-Rule

   (start-data-extraction)
  =>

   (bind ?stream (open "a:\\ACT-GEN.art" "r"))

   (WHILE (not (feof ?stream)) DO
     (bind ?line-stream (read-line ?stream))

     (IF (not (equal ?line-stream *eof*))
       THEN
         (poke ?*gen-buf* gen-buffer all ?line-stream)
         (bind ?schema-name (peek ?*gen-buf* gen-data act-num))
         (map-schema ?*gen-buf* gen-data ?schema-name)
         (assert (schema ?schema-name (is-a activity)))

         (bind ?string-stream-1 (open-string (get-schema-value ?schema-name temp-prec-act) "r"))
         (bind ?string-stream-2 (open-string (get-schema-value ?schema-name temp-succ-act) "r"))

         (WHILE (not (feof ?string-stream-1)) DO
           (bind ?prec-act (read ?string-stream-1))
           (IF (not (= ?prec-act *eof*))
             THEN
                 (assert (schema ?schema-name (prec-act ?prec-act)))
           )
         )

         (WHILE (not (feof ?string-stream-2)) DO
           (bind ?succ-act (read ?string-stream-2))
           (IF (not (= ?succ-act *eof*))
             THEN
                 (assert (schema ?schema-name (succ-act ?succ-act)))
           )
         )
         (slotd ?schema-name temp-prec-act)
         (slotd ?schema-name temp-succ-act)
     )
   )
```

```
    (close ?stream)
    (close ?string-stream-1)
    (close ?string-stream-2)
    (assert (activity-general-data-extracted))
)


(def-external-data par-data  20
  (act-num              0   4  :char  :symbol)
  (workflow-dir          4   4  :char  :symbol)
  (variable-production   8   3  :char  :symbol)
  (continuity-class     11   1  :char  :symbol)
  (max-splits           12   2  :char  :integer)
  (ver-logic-act        14   4  :char  :symbol)
  (ver-logic-lag        18   2  :char  :integer))

(def-external-data par-buffer 20
  (all 0 20 :char))

(defglobal ?*par-buf* = (alloc-buffer par-buffer))

(defrule   RULE-2-Activity-Special-Data-Extraction-Rule
    (activity-general-data-extracted)
  =>
    (bind ?stream (open "a:\\ACT-PAR.art" "r"))

    (WHILE (not (feof ?stream)) DO
      (bind ?line-stream (read-line ?stream))
      (IF (not (equal ?line-stream *eof*)) THEN
        (poke ?*par-buf* par-buffer all ?line-stream)
        (bind ?schema-name (peek ?*par-buf* par-data act-num))
        (map-schema ?*par-buf* par-data ?schema-name)

      )
    )

  (close ?stream)
  (assert (activity-special-data-extracted))
)


(def-external-data res-dmnd-data
  92
  (act-num     0  4  :char  :symbol)
  (res-1       4  10  :char  :symbol)
  (res-2      14  10  :char  :symbol)
  (res-3      24  10  :char  :symbol)
  (res-4      34  10  :char  :symbol)
  (res-t-1    44  9  :char  :symbol)
  (res-t-2    53  9  :char  :symbol)
  (res-t-3    62  9  :char  :symbol)
  (res-t-4    71  9  :char  :symbol)
  (res-q-1    80  3  :char  :integer)
  (res-q-2    83  3  :char  :integer)
  (res-q-3    86  3  :char  :integer)
  (res-q-4    89  3  :char  :integer))
```

```
(def-external-data res-dmnd-buffer 92
  (all 0 92 :char))

(defglobal ?*res-dmnd-buf* = (alloc-buffer res-dmnd-buffer))

(defrule   RULE-3-Activity-Resource-Demand-Data-Extraction-Rule
  (activity-general-data-extracted)
  =>

    (bind ?stream (open "a:\\ACT-RES.art" "r"))

    (While (not (feof ?stream)) Do
      (bind ?line-stream (read-line ?stream))
      (If (not (equal ?line-stream *eof*))
        Then
          (poke ?*res-dmnd-buf* res-dmnd-buffer all ?line-stream)
          (map-schema ?*res-dmnd-buf* res-dmnd-data is-a temp-activity)
      )
    )

    (FOR ?schema in-schema-children-of temp-activity DO
      (bind ?act-num (get-schema-value ?schema act-num))
      (bind ?res-q-1 (get-schema-value ?schema res-q-1))
      (bind ?res-q-2 (get-schema-value ?schema res-q-2))
      (bind ?res-q-3 (get-schema-value ?schema res-q-3))
      (bind ?res-q-4 (get-schema-value ?schema res-q-4))


      (IF (not (= ?res-q-1 0))
        THEN
          (bind ?res-1   (get-schema-value ?schema res-1))
          (bind ?res-t-1 (get-schema-value ?schema res-t-1))
          (assert (schema ?act-num (res-demand =(create$ ?res-1 ?res-t-1 ?res-q-1))))
      )

      (IF (not (= ?res-q-2 0))
        THEN
          (bind ?res-2 (get-schema-value ?schema res-2))
          (bind ?res-t-2 (get-schema-value ?schema res-t-2))
          (assert (schema ?act-num (res-demand =(create$ ?res-2 ?res-t-2 ?res-q-2))))
      )

      (IF (not (= ?res-q-3 0))
        THEN
          (bind ?res-3 (get-schema-value ?schema res-3))
          (bind ?res-t-3 (get-schema-value ?schema res-t-3))
          (assert (schema ?act-num (res-demand =(create$ ?res-3 ?res-t-3 ?res-q-3))))
      )

      (IF (not (= ?res-q-4 0))
        THEN
          (bind ?res-4 (get-schema-value ?schema res-4))
          (bind ?res-t-4 (get-schema-value ?schema res-t-4))
          (assert (schema ?act-num (res-demand =(create$ ?res-4 ?res-t-4 ?res-q-4))))
      )
    )

    (close ?stream)
    (assert (activity-resource-demand-data-extracted))
)


(def-external-data space-data
  13
  (act-num              0   4  :char  :symbol)
  (space-demand-class   4   1  :char  :symbol)
  (mpw-eqp-block        5   4  :char  :symbol)
  (mat-block            9   4  :char  :symbol))

(def-external-data space-buffer 13
  (all 0 13 :char))

(defglobal ?*space-buf* = (alloc-buffer space-buffer))
```

(defrule RULE-4-Activity-Space-Data-Extraction-Rule

```
    (activity-general-data-extracted)
  =>

    (bind ?stream (open "a:\\ACT-SPC.art" "r"))

    (WHILE (not (feof ?stream)) DO
      (bind ?line-stream (read-line ?stream))
      (IF (not (equal ?line-stream *eof*)) THEN
        (poke ?*space-buf* space-buffer all ?line-stream)
        (bind ?schema-name (peek ?*space-buf* space-data act-num))
        (map-schema ?*space-buf* space-data ?schema-name)
      )
    )

    (close ?stream)
    (assert (activity-space-data-extracted))
)

(def-external-data res-data
  30
  (res             0   10  :char  :symbol)
  (res-type       10    9  :char  :symbol)
  (res-mag        19    3  :char  :integer)
  (space-demand   22    8  :char  :float))

(def-external-data res-buffer 30
  (all 0 30 :char))

(defglobal ?*res-buf* = (alloc-buffer res-buffer))
```

(defrule RULE-5-Resource-Schemata-Creation-and-Data-Extraction-Rule

```
    (start-data-extraction)
  =>

    (bind ?stream (open "a:\\RES.art" "r"))
    (WHILE (not (feof ?stream)) DO
      (bind ?line-stream (read-line ?stream))
      (IF (not (equal ?line-stream *eof*)) THEN
        (poke ?*res-buf* res-buffer all ?line-stream)
        (bind ?schema-name (peek ?*res-buf* res-data res))
        (map-schema ?*res-buf* res-data ?schema-name)
        (assert (schema ?schema-name (is-a resource)))
      )
    )

    (close ?stream)
    (assert (resource-data-extracted))
)


(def-external-data block-data
  28
  (block-num        0   4  :char  :symbol)
  (space-available  4   8  :char  :float))

(def-external-data block-buffer 28
  (all 0 28 :char))

(defglobal ?*block-buf* = (alloc-buffer block-buffer))

(defrule  RULE-6-Work-Block-Schemata-Creation-and-Data-Extraction-Rule
    (start-data-extraction)
  =>

    (bind ?stream (open "a:\\BLOCK.art" "r"))

    (WHILE (not (feof ?stream)) DO
      (bind ?line-stream (read-line ?stream))
      (IF (not (equal ?line-stream *eof*)) THEN
        (poke ?*block-buf* block-buffer all ?line-stream)
        (bind ?schema-name (peek ?*block-buf* block-data block-num))
        (map-schema ?*block-buf* block-data ?schema-name)
        (assert (schema ?schema-name (is-a work-block)))
      )
    )

    (close ?stream)
    (assert (work-block-data-extracted))
)



(def-external-data project-data
  6
  (from-floor   0   3  :char  :integer)
  (to-floor     3   3  :char  :integer))

(def-external-data project-buffer 6
  (all 0 6 :char))

(defglobal ?*project-buf* = (alloc-buffer project-buffer))
```

```
(defrule  RULE-7-Other-Project-Data-Extraction-Rule
   (start-data-extraction)
  =>

   (bind ?stream (open "a:\\OTHER.art" "r"))

   (WHILE (not (feof ?stream)) DO
     (bind ?line-stream (read-line ?stream))
     (IF (not (equal ?line-stream *eof*)) THEN
       (poke ?*project-buf* project-buffer all ?line-stream)
       (bind ?*from-floor* (peek ?*project-buf* project-data from-floor))
       (bind ?*to-floor*   (peek ?*project-buf* project-data to-floor))
     )
   )

   (close ?stream)
   (assert (other-project-data-extracted))
)


(defrule Output-Data
  ?x <- (schedule-completed)

  =>

  (bind ?output-stream (open "a:\\output.art" "w"))

  (FOR ?schema in-schema-children-of activity DO
    (bind ?act-dscrp (get-schema-value ?schema act-dscrp))
    (FOR ?output-schedule in-slot-values-of ?schema output-schedule DO
      (printf ?output-stream "%-4S%-30s%-3d%-3d%5.1f%6.1f%6.1f%6.1f%6.1f\n"
                  ?schema
                  ?act-dscrp
                  (nth$ ?output-schedule 1)
                  (nth$ ?output-schedule 2)
                  (nth$ ?output-schedule 3)
                  (nth$ ?output-schedule 4)
                  (nth$ ?output-schedule 5)
                  (nth$ ?output-schedule 6)
                  (nth$ ?output-schedule 7)
      )
    )
  )

  (close ?output-stream)
  (retract ?x)
)


(defrule Control-Rule-1
 (declare (salience 1000))
 =>
   (assert (start-data-extraction))
)
```

```
(defrule Control-Rule-2
  ?x <- (start-data-extraction)
  ?a <- (activity-general-data-extracted)
  ?b <- (activity-special-data-extracted)
  ?c <- (activity-resource-demand-data-extracted)
  ?d <- (activity-space-data-extracted)
  ?e <- (resource-data-extracted)
  ?f <- (work-block-data-extracted)
  ?g <- (other-project-data-extracted)

  =>
    (assert (start-schedule))
    (retract ?x ?a ?b ?c ?d ?e ?f ?g)
)


(defrule Control-Rule-3
  (declare (salience -1000))
  ?x <- (start-schedule)
  (old-time ?)
  (schema act-set (ias &:(slot-null act-set ias)))

  =>
    (assert (schedule-completed))
    (retract ?x)
)



(defrule Verify-Overall-Resource-Pool
  (declare (salience 1000))
  (start-schedule)

  =>

  (FOR ?res in-schema-children-of resource DO

     (bind ?res-mag (get-schema-value ?res res-mag))
     (bind ?max-res-demand 0)

     (FOR ?act in-schema-children-of activity DO
       (FOR ?res-demand in-slot-values-of ?act res-demand DO
          (IF (and (= (nth$ ?res-demand 1) ?res)
                 (> (nth$ ?res-demand 3) ?max-res-demand))
            THEN
                (bind ?max-res-demand (nth$ ?res-demand 3)))
       )
     )

     (IF (> ?max-res-demand ?res-mag)
       THEN
          (modify-schema-value ?res res-mag ?max-res-demand))

  )
)
```

```
(defrule Initialize-Activities
  (declare (salience 1000))
  (start-schedule)

  =>

  (FOR ?act in-schema-children-of activity DO

    (modify-schema-value ?act act-dur (get-schema-value ?act nor-act-dur))

    (IF (= (get-schema-value ?act continuity-class) A)
      THEN
          (assert (schema ?act (scheduling-method Verify-Horizontal-Logic-Constraints-A)))

      ELSE
          (bind ?max-splits (get-schema-value ?act max-splits))
          (bind ?total-floors (+ (- ?*to-floor* ?*from-floor*) 1))
          (bind ?floors-per-segment (truncate (/ ?total-floors ?max-splits)))
          (bind ?floors-last-segment (- ?total-floors (* ?floors-per-segment (- ?max-splits 1))))

          (assert (schema ?act (scheduling-method Verify-Horizontal-Logic-Constraints-B)
                           (floors-per-segment ?floors-per-segment)
                           (floors-last-segment ?floors-last-segment)))

          (IF (= (get-schema-value ?act workflow-dir) up)
            THEN (assert (schema ?act (from-floor ?*from-floor*) (to-floor)))
            ELSE (assert (schema ?act (from-floor ?*to-floor*) (to-floor)))
          )
    )
  )
)


(defrule Define-IAS
  (declare (salience 500))
  (start-schedule)

  (or (schema ?act
        (prec-act none))

      (schema ?act
        (prec-act $?x)
        (prec-act-scheduled $?x)))
  =>

  (put-schema-value act-set ias ?act)
)


(defrule Define-EAS
  (new-time ?t)
  (schema act-set (ias ?))

  =>
  (FOR ?act in-slot-values-of act-set ias DO
    (IF (= (get-schema-value ?act es) ?t)
      THEN (put-schema-value act-set eas ?act)
    )

    (IF (< (get-schema-value ?act es) ?t)
      THEN
           (modify-schema-value ?act es ?t)
           (put-schema-value act-set eas ?act)
    )
  )
)



(defrule Define-Cycle-Time
   ?x <- (old-time ?t)
   (schema act-set (ias ?))

  =>

   (bind ?min-es 1000000)
   (FOR ?act in-slot-values-of act-set ias DO
     (bind ?es (get-schema-value ?act es))
     (IF (< ?es ?min-es) THEN (bind ?min-es ?es))
   )

   (assert (new-time ?min-es))

   (retract ?x)
)


(defrule Process-Scheduling-Cycles

  ?x <- (new-time ?t)
  (schema act-set (eas ?))

  =>

  (While (not (slot-null act-set eas)) Do

    (bind ?act (get-schema-value compare act-num))

    (FOR ?eas-act in-slot-values-of act-set eas DO

      (IF  (< (get-schema-value ?eas-act ls) (get-schema-value ?act ls))
        THEN
            (bind ?act ?eas-act)
      )

      (IF  (= (get-schema-value ?eas-act ls) (get-schema-value ?act ls))
        THEN
            (IF (< (get-schema-value ?eas-act nor-act-dur) (get-schema-value ?act nor-act-dur))
              THEN
                  (bind ?act ?eas-act)
```

```
          )
      )

      (IF  (and (= (get-schema-value ?eas-act ls) (get-schema-value ?act ls))
              (= (get-schema-value ?eas-act nor-act-dur) (get-schema-value ?act nor-act-dur)))
        THEN
          (IF (< (get-schema-value ?eas-act max-act-dur) (get-schema-value ?act max-act-dur))
            THEN
               (bind ?act ?eas-act)
          )
      )
    )

    (modify-schema-value act-set oss ?act)

    (While (not ( slot-null act-set oss)) Do

       (IF (= (get-schema-value ?act continuity-class) A)
         THEN
           (IF (= (get-schema-value ?act workflow-dir) up)
             THEN (send scheduling-method ?act ?t ?*from-floor* ?*to-floor*)
             ELSE (send scheduling-method ?act ?t ?*to-floor* ?*from-floor*)
           )
       )

       (IF (= (get-schema-value ?act continuity-class) B)
         THEN
           (bind ?floor-i (get-schema-value ?act from-floor))
           (bind ?floors-per-segment (get-schema-value ?act floors-per-segment))
           (bind ?floors-last-segment (get-schema-value ?act floors-last-segment))

           (IF (= (get-schema-value ?act workflow-dir) UP)
             THEN
               (IF (= (+ ?floor-i ?floors-last-segment -1) ?*to-floor*)
                 THEN (modify-schema-value ?act to-floor (+ ?floor-i ?floors-last-segment -1))
                 ELSE (modify-schema-value ?act to-floor (+ ?floor-i ?floors-per-segment -1))
               )

               (send scheduling-method ?act ?t ?floor-i (get-schema-value ?act to-floor))

             ELSE
               (IF (= (- ?floor-i ?floors-last-segment -1) ?*from-floor*)
                 THEN (modify-schema-value ?act to-floor (- ?floor-i ?floors-last-segment -1))
                 ELSE (modify-schema-value ?act to-floor (- ?floor-i ?floors-per-segment -1))
               )

               (send scheduling-method ?act ?t ?floor-i (get-schema-value ?act to-floor))

           )
       )
    )
  ) ;end big while

  (assert (old-time ?t))
  (retract ?x)
)
```

Vita

The author was born on August 9, 1960 in Kuwait. He received his Bachelor of Science degree in Civil Engineering from Kuwait University in May 1981. Between May and December of 1981 he joined Kuwait Institute for Scientific Research (KISR) as a Research Engineer. In January 1981 he started graduate studies in the Civil Engineering Department at the University of Waterloo (Ontario, Canada) where he was awarded a Master of Science degree in Structural Engineering in August 1983. Between August 1983 and December 1984 he joined a construction firm in Kuwait as a Planning and Scheduling Engineer. In January 1984 he joined an A/E consulting firm (Kuwaiti Engineers Office) as a Contract Administrator. In August 1988, he continued his graduate studies in the Construction Engineering and Management Division at Virginia Polytechnic Institute and State University (VPI&SU) where he was awarded a Doctor of Philosophy degree in Civil Engineering in December 1992.