A Knowledge-Based Model of Time-Cost Tradeoff Analysis for Construction Schedules

Alan T. Axworthy

TECHNICAL REPORT
Number 21 A

January, 1990

Stanford University
Preface

The work presented in this thesis was funded by the Center for Integrated Facility Engineering at Stanford University (CIFE), whose support is gratefully acknowledged. The ability to generate accurate construction plans, schedules and cost estimates is at the core of CIFE’s goals for design-construction integration. This research project involves the application of advanced artificial intelligence techniques to a practical problem that has been difficult to approach with traditional programming techniques. Earlier work has demonstrated many of the techniques utilized in this research. This project brings these techniques together in a unique manner to apply to a specific problem of interest to engineers and managers in many fields where good planning is important.

Work such as this research project are certainly not individual efforts. They require and are made stronger by the involvement of a number of people. Professor Raymond E. Levitt has been a regular source of inspiration and direction. As an advisor and project manager, he has provided focus, feedback and impetus at strategic times in order to keep the project on track. As a friend, he has provided moral support and personal direction during the process of defining the work, implementing the prototype and getting the written document out the door. Professor John Fondahl’s class in project management and scheduling introduced the author to the time-cost tradeoff problem. Professor Fondahl’s thoughts on the project scheduling environment also helped the author focus his own thinking. John Kunz’s class on the design of engineering knowledge systems provided the author the opportunity to formalize his thinking and gave him a framework (John’s maximum anxiety heuristic) for developing the prototype.

The author has benefitted from other work in the field of knowledge-based scheduling which is noted in the body of this thesis. A special acknowledgement is necessary, however, to Diego Echeverry for sharing his object-oriented CPM code, which helped the author complete his own.

Finally and certainly not least, the author acknowledges the help and support of his wife, Mary. She provided not only moral support but also aid in organizing the thesis and cogent criticisms – substantive as well as grammatic – of the various drafts.
# Table of Contents

Preface .................................................................................................................. iii

Table of Contents ................................................................................................. iv

List of Figures ......................................................................................................... v

List of Tables .......................................................................................................... vi

Chapter 1. Summary .............................................................................................. 1

Chapter 2. Introduction
  2.1. Overview ....................................................................................................... 2
  2.2. Planning, scheduling and project management .............................................. 2
  2.3. Project participants ...................................................................................... 3
  2.4. Expediting tactics: Traditional time-cost tradeoff analysis ......................... 4
  2.5. Combining traditional project management analysis with
       organizational analysis using AI tools .......................................................... 6

Chapter 3. Literature Review ................................................................................ 8
  3.1. AI literature ................................................................................................ 8
  3.2. Project management literature .................................................................. 12
  3.3. Organizational literature ............................................................................ 19
  3.4. Contributions of the research ................................................................... 24

Chapter 4. Research Approach .......................................................................... 26
  4.1. Goals of the research ................................................................................ 26
  4.2. The approach to address these goals ......................................................... 27
  4.3. Validation of the system ........................................................................... 27
  4.4. Development style of prototyping: “maximum anxiety” ......................... 28
  4.5. Choice of KEE as a programming environment ....................................... 28

Chapter 5. Implementation of JANUS ................................................................. 29
  5.1. The components & their interactions ......................................................... 29
  5.2. OOP in KEE .............................................................................................. 40

Chapter 6. Example Problem ............................................................................ 41
  6.1 Project Network .......................................................................................... 41
  6.2 Project Estimate ......................................................................................... 42
  6.3 Project Participants .................................................................................... 43
  6.4 How JANUS approaches the analysis ....................................................... 44

Chapter 7. Conclusions ...................................................................................... 50
  7.1. Contributions ............................................................................................ 50
  7.2. Suggested Extensions .............................................................................. 50

References ............................................................................................................ 53
List of Figures

Figure 1: Traditional approximation of activity time-cost relationships ...................... 5
Figure 2: Activity time-cost relationships using expediting tactics .............................. 7
Figure 3: Governance structures and commercial transaction classifications .................. 21
Figure 4: Graph of the Activities hierarchy in JANUS ............................................. 30
Figure 5: Graph of the Participants hierarchy in JANUS .......................................... 33
Figure 6: Graph of the Contracts hierarchy in JANUS ............................................. 34
Figure 7: Graph of the Tactics hierarchy in JANUS ................................................ 36
Figure 8: Graph of the Crews hierarchy in JANUS .................................................. 37
Figure 9: Determining the cost slope multiplier for an activity .................................. 38
Figure 10: "Fondahl’s Famous Pier Problem," adapted for JANUS ................................ 41
Figure 11: Employment relationships among the pier project participants .................. 44
List of Tables

Table 1: Formulas for activity estimates ................................................................. 37
Table 2: Formulas for activity cost slope factors ....................................................... 39
Table 3: Tactic cost slope factors ........................................................................ 40
Table 4: Initial estimate data for the example pier project ........................................... 42
Table 5: Initial durations and cost slopes and final durations for two test scenarios for the example pier project ................................................................. 43
Table 6: Cost and duration impacts of the application of different tactics to the activity Install.Girts ................................................................. 47
Chapter 1.

Summary

"Buying time" is a common phrase in the English language. Historical approaches to applying this notion to formal critical path method (CPM) schedule analysis have suffered from two major problems. First, formal time-cost tradeoff analysis requires extensive numeric data which is difficult to acquire, is of suspect validity and requires complicated numeric manipulation to be of use. Second, the perceived value of time varies among the various project participants. One participant's analysis will disagree with another's because of their different perceptions of the value of shortening activity durations.

This thesis presents the conceptual architecture for JANUS, a knowledge-based time-cost tradeoff simulator that addresses these shortcomings of traditional approaches. The first problem is dealt with by utilizing a set of generic expediting tactics. These are standard means of shortening activity duration such as overtime, multiple shifts, additional crews or alternative equipment or methods. The second problem is dealt with by explicitly modeling the project participants, their goals and their relationships with one another. These extensions to traditional CPM analysis allow JANUS to evaluate the activity costs resulting from specific expediting methods more realistically and to simulate the effect of different contracting strategies on a project's schedule.

JANUS is an attempt to capture both the motivational and the technical components of buying time on a project. JANUS uses activity sequence, time and cost relationships along with a set of generic expediting tactics for its time-cost tradeoff analysis. It also represents project participants and derives their goals from the owner's view of the project and the contractual provisions under which each project participant is employed. Each activity has a responsible participant who determines the value of time and therefore the maximum amount that may be spent expediting that activity. It can thus provide a more meaningful forecast of the likely results of alternative strategies – contractual as well as technical – to shorten project duration.
Chapter 2.

Introduction

2.1. Overview

Time-cost tradeoff analysis is a formal means of analyzing when and where to buy time on a project. It is an effective tool for analyzing schedules. It can be equally effective during initial project scheduling and during the rescheduling that always seems to be required at some point during a project. However, the nature of critical path schedules and the amount of data necessary to enable a full time-cost tradeoff analysis make it too complex a process to be utilized often.

A number of artificial intelligence (AI) technologies have evolved over the past few years that may be applied to time-cost tradeoff analysis to make it more useful and valid. These technologies include frame-based knowledge representation and object-oriented programming. JANUS, a prototype system using these AI programming techniques, has been developed to demonstrate their utility in analyzing time-cost tradeoffs among activities in project schedules. This prototype considers realistic activity expediting tactics and the goals of multiple project participants in its analysis.

2.2. Planning, scheduling and project management

Planning is fundamental to project success. The planning process helps define the scope of the project, and the tasks and resources necessary for its successful completion. Planning also builds commitment to the project among the participants. Scheduling is just one of many important aspects of planning. Ideally, the schedule embodies the project plan. It can be an excellent means of communicating complex constraints and of documenting the support and commitment of the participants. Along with the project cost estimate, the schedule forms the baseline for all control systems which are applied throughout the life of the project.

Tools to help project managers with scheduling tasks have been used since the late 1950's. Traditional CPM and PERT methodologies provide a great deal of help in ordering the tasks involved in a project and in allocating resources to an initial plan. They are, however, less useful in determining where and how much to spend to expedite a project which is behind schedule. They also do not help in evaluating the effect of the varying perceptions of the value of time among a project's participants.
Consider a situation typical of a construction project: the late delivery of a key piece of equipment. Traditional CPM scheduling techniques help determine whether or not the delivery has an effect on the project completion date. They also help determine the magnitude of an impact to the schedule. However, the main question that comes up in this situation is not, “What is the time impact of this delay?” It is rather, “How can we best mitigate the overall time and cost impact of this delay on the project?” In other words, the answer is not that the scheduled completion date must be delayed. The answer is that the project must still be completed on time – we just need to figure out how to do that as cheaply as possible.

Traditional CPM tools do not help with answering the latter question. Traditional approaches to time-cost tradeoff, using normal and crash data for all project activities, can help us determine on which activities to focus our expediting attention. But even that analysis provides no guidelines for figuring out exactly how to go about expediting any given activity.

Moreover, these traditional analytical approaches do not consider the fragmented nature of the construction environment. Implicit in traditional time-cost tradeoff analysis are the assumptions that there exists a single value of a day, that everyone on the project agrees with that value, that everyone acts in accordance with that belief and that every activity, regardless of whose responsibility, is equally capable of being expedited. Observation of the construction work place indicates that those assumptions are seldom valid.

2.3. Project participants

Construction project participants include facility owners, architects, consulting engineers, contractors, subcontractors and material suppliers. In the construction domain, it is tempting to oversimplify the analysis of a participant’s perceived value of time. A facility owner may view its project as providing a certain income which, coupled with its expenses and capital structure, determines an expectation of profit-earning potential. This owner, then, may have a relatively clear view of what it thinks the project is worth on a daily basis. However, simply writing this value into its contracts with its designers, suppliers and contractors does not guarantee that those participants will agree with that valuation of a day saved or lost in completion of the project.

Consider a typical lump-sum contract agreement between a facility owner and a general contractor containing a liquidated damages clause. A realistic value of profit
potentially for the owner may be so great with respect to the contractor’s capital structure that the contractor cannot justify accepting the financial risk of a daily bonus-penalty clause reflecting the owner’s value of time. A contractor’s typical response is to gain more control over the risk by padding the schedule (providing contingency for the unexpected) and by subcontracting portions of the work (spreading the risk). Padding the schedule is certainly not in the best interest of the owner. Spreading the risk by subcontracting may do little more than worsen the control situation and introduce smaller, more vulnerable links into the chain. The subcontractors are generally smaller than the prime contractor and even less able to accept the financial risks of the delay damages clause.

The result is that the contracting community simply does not behave as if the owner’s value of time is what time is worth to them. They have their own perceptions of the value of time, based on their experience, their exposure and their perceived ability to control the financial risk posed by the damages clause. In the case of the prime contractor, this value often turns out to be the cost of managing the project, i.e., the daily indirect cost of the project, often called the jobsite overhead. Small subcontractors are usually resource constrained. That is, they have few resources (equipment and people) and try to utilize them as efficiently as possible. They tend to schedule their resources across several projects at once, rather than one project at a time. Therefore, they may have absolutely no incentive to spend anything to reduce the duration of their work on a given project – overall resource smoothing is how they maximize their profit.

2.4. Expediting tactics: Traditional time-cost tradeoff analysis

Buying time is an intuitively appealing way of looking at scheduling and rescheduling. There is evidence that scheduling professionals start their scheduling process by optimizing activity durations piecewise, then modifying this initial schedule to accommodate the various goals of the project [Echeverry 88]. Implicit in such a process are the assumptions that time has a determinable value and that some means of exchanging money for time can be found. Experienced schedulers perform this schedule analysis intuitively based on their experience. Formal procedures for increasing cost to decrease time require a great deal of project-specific data, including at least two estimates of cost and duration for each activity: “normal” cost and duration and “crash” cost and duration (see Figure 1). The normal figures represent the most efficient way of performing the activity. The crash figures represent the absolutely quickest way of performing the activity. Given all this information, there are well-established procedures for choosing activities to “crash,”
for determining how much to crash them, and for evaluating the effect of these changes on the overall schedule.

![Graph showing time-cost relationships](image)

**Figure 1: Traditional approximation of activity time-cost relationships**

\[
\begin{align*}
D_N &= \text{Normal Duration} \\
C_N &= \text{Normal Cost} \\
D_C &= \text{Crash Duration (minimum possible duration)} \\
C_C &= \text{Crash Cost (cost to achieve minimum possible duration)} \\
\text{Cost Slope} &= \frac{\Delta C}{\Delta D} = \frac{(C_C - C_N)}{(D_C - D_N)}
\end{align*}
\]

This data-intensive approach is impractical for most real world problems. One objection is simply the amount of data required. There is seldom enough time to develop even one estimate adequately, let alone a second for a different set of constraints. Even when such information is made available, the time-cost relationships are estimated. That is, the path between the normal and crash limits is often assumed to be linear, as shown in Figure 1. A related objection then arises concerning what is meant by some point along this line between the end points: exactly how might one achieve a particular activity duration at a given cost? This question is not answerable using this approach. Such estimates and assumptions color the analysis and limit its utility.

Another objection concerns the differing interpretations of the project data among the various project participants. Project-wide time-cost tradeoff analysis requires a consistent value of time, but each participant views the value of time differently. Again, consider the construction domain. An owner may have hundreds of thousands, if not millions, of dollars daily production riding on a project. Contractors' management personnel tend to value time in relation to their daily jobsite overhead or indirect costs.
Field supervisory personnel value time in relation to how they are evaluated by their superiors. Accounting systems, company policies toward foreman costs and other indirect allocations, bonus policies and consideration of the impact of delay on their reputation are all factors that affect the value different project participants put on time. Currently-available tools do not provide any framework for analyzing these factors.

2.5. Combining traditional project management analysis with organizational analysis using AI tools

The research described in this thesis suggests an alternate approach to time-cost tradeoff analysis. By defining broadly-applicable expediting tactics and relating them to individual activities during the analysis, more realistic time-cost relationships are actually used, giving more realistic results (see Figure 2). Explicitly modelling the participants, their relationships and their perceived values of time also provides for a more robust and useful analysis.

Consider that a tactic has a characteristic premium cost based on the nature of the tactic – the number of total hours worked in a day, the number of overtime hours worked, the additional cost of the overtime work, etc. – and its efficiency. It also has a characteristic duration improvement based on many of the same factors. Both factors, cost premium and duration improvement, can be expressed in terms of a percentage of normal cost and time. These tactic characteristics can be used as a relative measure of cost and duration improvement.

An activity’s cost premium will be slightly different from a tactic’s because an activity’s labor-related costs include labor fringes and equipment costs in addition to direct labor costs. The cost slope of an activity, then, can be expressed in terms of its normal cost per day and a coefficient related to its cost premium and a tactic’s characteristic duration improvement. (Refer to Figure 9 for an explanation of these calculations and relationships.) Tactics can be successively applied to an activity to incrementally reduce its duration by iteratively selecting those with increasing duration reduction.
Figure 2: Activity time-cost relationships using expediting tactics

\[
\begin{align*}
C_N &= \text{Normal Cost} \\
C_{T1} &= \text{Cost using tactic T1} \\
C_{T2} &= \text{Cost using tactic T2} \\
D_N &= \text{Normal Duration} \\
D_{T1} &= \text{Duration using tactic T1} \\
D_{T2} &= \text{Duration using tactic T2}
\end{align*}
\]

Refer to Figure 9 for an explanation of the calculations. To illustrate the idea of activity time-cost relations from expediting tactics, consider a 10-hour shift as expediting tactic T2. Characteristics of this tactic include:
- Overtime hours: 2 @ time-and-a-half
- Efficiency = 95%.

From this information, other attributes can be calculated:
- Cost Premium = 15.8%
- Duration Improvement = 18.8%
- Cost Slope Multiplier = -1.000.

The pertinent attributes for expediting tactic T1, a six-day work week are:
- Overtime hours: 8 @ time-and-a-half
- Efficiency = 95%
- Cost Premium = 14.0%
- Duration Improvement = 14.0%
- Cost Slope Multiplier = -1.142

Traditional computer programming approaches have limited the types of analysis that may be done by a computer to numerically-intensive problems. CPM calculation of a schedule network is a good example of a numerically-intensive calculation that has benefitted from the use of computers in the past. Traditional programming approaches, however, break down in situations requiring symbolic rather than numeric manipulation. The analysis of relationships among project participants is an example where numeric manipulation is of little use and where symbolic manipulation may help. JANUS uses a frame-based scheme to represent the participants and uses object-oriented programming to describe, manipulate and reason about their relationships among one another.
Chapter 3. Literature Review

This section reviews literature related to the research in three major areas:

- artificial intelligence (AI)
- project management
- organizational behavior.

3.1. AI literature

The work described in this thesis makes use of concepts that developed from research into artificial intelligence. This section introduces some AI ideas which are employed in the thesis and refers the interested reader to other sources for more detail. It also reviews related work which uses these ideas to model organizations.

3.1.1. General AI concepts

Expert systems, knowledge representation and model-based reasoning are all ideas used in this research project. Understanding these ideas is important to understanding the implementation of JANUS, the prototype system developed to demonstrate the concepts of the research.

3.1.1.1. Expert systems and knowledge representation

The realm of artificial intelligence encompasses several fields of study: from natural language processing and visual interpretation to expert systems and knowledge representation. From a practical viewpoint, artificial intelligence can be viewed as a set of computer programming techniques. The research in this thesis has benefitted from a number of techniques evolved from AI research, specifically frame-based knowledge representation and object-oriented programming.

The fundamental ideas of frame-based knowledge representation concern the abstraction hierarchy within which "objects" and classes of objects are described. Frames store descriptions of objects, which can represent essentially anything, from abstract concepts such as schedule activities to concrete objects such as bricks and mortar. Within each frame, object characteristics are held in "slots" containing attribute-value pairs. The power of this means of representation comes from the arrangement of objects in a "taxonomy" or hierarchy of abstraction. The attributes or characteristics of classes of
objects can be defined, and then all instances, or members, of the class inherit their characteristics. Inherited slot values of member objects can be overridden by local values. Object instances can belong to more than one class, inheriting relevant characteristics from each parent. This arrangement allows a great deal of knowledge to be defined very high in the frame hierarchy, which is efficient. It also allows a great deal of deduction about the attributes of objects to occur simply by inheritance of class characteristics from several superclasses.

A related concept is that of object-oriented programming (OOP). OOP extends a frame-based knowledge representation to allow frame slots to contain not only constant values, but also procedures or programs to simulate the behavior of an object. These procedures can perform calculations or perform inferences with production rules and can change the values of other slots. They can also “send messages” to other objects. These procedures can be triggered in two ways: by receiving a message from another object to execute or from a “demon” attached to a slot. Demons are themselves programs or procedures that automatically run whenever a slot value is accessed or changed. The overriding metaphor is that of a group of independent, self-aware objects passing messages back and forth and updating their own states as messages are received and as their attribute values change.

A more detailed discussion of these and other AI techniques is beyond the scope of this thesis, but the interested reader can refer to [Parsaye 88] for more information.

This research uses frames to represent the various components required for time-cost tradeoff analysis, e.g., schedule activities, project participants, expediting tactics, etc. It uses OOP to represent the behavior of those components, e.g., updating network data when activity durations change.

3.1.1.2. Model-based reasoning: structure & function

A significant refinement of traditional expert systems and frame-based knowledge representation is known as model-based reasoning. This refinement uses a model of a system as its underlying representation. This model is defined in such a way that the system’s components’ structures and their functions are explicit and both can be used for reasoning. Particularly important is the model’s ability to mix procedural system behavior with knowledge-based representation. [Kunz 87] contains a detailed discussion of the model-based reasoning metaphor and examples of its use in differing domains.
This research uses the model-based reasoning paradigm to describe and model the entire system of project actors and activities, including their relationships among one another and their behavior.

### 3.1.2. Modeling of organizations

Computers have been used for modeling of organizations, both with traditional programming approaches and utilizing AI techniques. Some of this work is related to the present research and is reviewed in this section.

#### 3.1.2.1. AROUSAL

[Lansley 87] describes AROUSAL: A Real Organization Unit Simulated As Life. AROUSAL is a computer simulation that presents a user or set of users with a fictional construction company and business environment. The user(s) make business decisions for each quarterly business cycle for a number of years. AROUSAL simulates the effects of those decisions on the overall business. The decisions each user must make each business cycle include:

- responding to bid invitations, including pricing and bid loading
- determining the organizational structure of the company, including firing staff, recruiting new staff, reassigning existing staff, determining salary levels and defining new functions and positions
- allocating staff to specific projects
- determining the level of subcontracting
- determining the level and focus of marketing.

AROUSAL is a commercial simulation product that is based on data from a number of privately-held construction firms. It is used to train managers of construction businesses. It is quite realistic, but its internal organizational model is not structured to be visible to users. It has therefore been of little help to researchers in pursuing analyses unrelated to those specifically included in the program.

AROUSAL is implemented in a traditional programming language, is distributed in compiled form and runs on IBM-compatible personal computers running MS-DOS.
3.1.2.2. HOWSAFE

Expert systems have been most notably successful when applied to diagnostic problems. The nature of diagnostic problems is such that all solutions can be enumerated allowing the system to infer degrees of certainty for each of the possible solutions from a range of available evidence. In effect, such expert systems apply the notion described in [Clancey 85] of heuristic classification. [Levitt 86] describes HOWSAFE, an expert system for evaluating construction company safety performance and diagnosing areas of weakness. HOWSAFE is an example of computers being used to model a social system rather than a physical system.

HOWSAFE represents its knowledge in the form of hierarchical hypotheses, which may be either proved or disproved by lower level hypotheses. Each hypothesis may be proved or disproved by combinations of other hypotheses or by factual assertions whose values can be objectively determined by the user. In this way, HOWSAFE decomposes the analysis of a contractor's organization for supporting safety into a series of specific questions that can be answered by a user with little room for judgement or interpretation.

HOWSAFE is implemented in The Deciding Factor, an expert system shell that runs on IBM-compatible personal computers running MS-DOS. The Deciding Factor is a shell that was developed from the work surrounding the development of Prospector at SRI [Campbell 82]. Prospector is an expert system which determines the presence of mineral deposits in a region based on observable surface geological data, interpretation of test data and sample analysis.

The research described in this thesis and demonstrated in the prototype JANUS differs from the heuristic classification type of expert system exemplified by HOWSAFE in its use of model-based reasoning as its fundamental representation and reasoning paradigm. While HOWSAFE represents its knowledge in the form of hypotheses and assertions of fact, JANUS contains a model of each of the components required for its time-cost tradeoff analysis. JANUS uses this underlying model to simulate the behavior and interactions of the system's components, ultimately resulting in effects on a schedule network.

3.1.2.3. Actors

[Paulson 89] discusses work currently under way to simulate the knowledge environment needed to support autonomous construction robot agents. Robots and other
knowledge sources – which may include people – are referred to as agents. One type of knowledge needed for agents is organizational knowledge, i.e., knowledge about what agents exist and what types of information each type of agent may contain. Representing the interactions and communications among numerous autonomous agents can become extremely complex, requiring new programming paradigms to become feasible.

One programming paradigm which may prove useful for such environments is *Actors*, described in [Agha 86]. *Actors* is a concurrent programming framework which allows programming of intelligent agents. These agents take incoming communications and, in response to those communications:

- send communications to other agents
- do something (exhibit some behavior)
- create new actors.

The *Actors* framework is much like OOP with the important extension that actors can all operate concurrently. Systems like *Actors* will take advantage of new and developing parallel computing architectures, which are not yet commonly used in industry. The ideas presented in this thesis may be able to take advantage of the features of systems such as *Actors* in the future.

### 3.2. Project management literature

This section reviews pertinent literature concerning critical path method (CPM) scheduling and project management, previous work on time-cost tradeoff analysis and previous work to extend traditional CPM analysis.

#### 3.2.1. Traditional CPM scheduling

CPM scheduling is taught today to students in many disciplines from business to industrial engineering to construction management [Barrie 84]. This section will review some of that literature.

[Kelley 89] describes the development of the critical path method of project analysis between 1956 and 1959 in Du Pont's Integrated Engineering Control (IEC) Group. This work developed the mathematical foundations for critical path analysis. What we now call the “critical path” of a project was originally called the “main chain” by IEC. The phrase “critical path” was coined by the developers of PERT and Du Pont christened its approach the “Critical Path Method” in May 1959.
The Du Pont IEC work included consideration of the expeditable nature of construction activities. That is, activities have characteristic "optimal" or "normal" ways of being performed, i.e., preferred methods, crews, etc. Expediting an activity would cause its direct cost to increase. Activities also have characteristic "crash" durations – minimum feasible durations – and associated "crash" costs. These ideas underlie a great deal of the work that has been done to date on time-cost tradeoff analysis.

This work at Du Pont has had a profound impact on the way in which construction work is planned and executed. It provided a mathematical foundation for schedule network analysis and some notation conventions that are in common use today. It did not, however, deal with the organizational aspects of the project. It provided no framework for analyzing a number of separate participants in the project or for analyzing the effect of their varying goals on the project schedule. It also did not provide any help to the scheduler in determining how to expedite an activity.

JANUS addresses the organizational aspects of project schedule analysis. It adds the notions of multiple project participants to the analysis. With this added dimension, JANUS can now help to show how contracting strategies can affect participant perceptions of the value of time and how those perceptions can affect the overall project schedule.

[Fondahl 87] describes the research that was conducted at Stanford at the same time as and independent of the Du Pont work. It also mentions the difficult nature of the time-cost tradeoff problem with respect to construction schedules. [Fondahl 61] introduced simplified means of performing network calculations by hand and introduced what came to be known as the precedence diagramming method (PDM) of network notation. This work includes discussion of resource levelling and resource-constrained scheduling. [Fondahl 64] introduced manual methods for managing and updating subnetworks.

Today, Fondahl teaches a non-computer approach to time-cost tradeoff analysis. This analysis requires a crash cost, crash duration, normal cost and normal duration for each activity. Given this data, his method involves:

1. selecting an activity to expedite – the critical activity with the lowest cost slope;
2. determining the "Network Interaction Limit" (NIL) for the network around that activity, i.e., the amount of time that the activity may be reduced before it is no longer critical;
3. reducing the activity’s duration either to its NIL or to its crash duration, whichever is greater;
4. updating the project cost and duration bookkeeping;
5. iterating through the process until further duration reduction costs more than is saved by the reduction.

Fondahl’s efforts to simplify schedule networks helped to increase the use of formal scheduling analysis in the construction industry. His manual methods clarified the mechanics of manipulating schedules and allowed the user the greatest ability to exercise his judgement during the analysis. His approach to time-cost tradeoff provides the scheduler no assistance in determining how to expedite an activity. It simply postulates the existence of crash costs and durations for all activities and assumes a linear relationship between the normal point and the crash point. It also makes no allowance for different project participants having different opinions of the value of time. It assumes an overall project indirect cost rate that limits the amount that may be spent to expedite any portion of the project.

JANUS addresses the problem of how to expedite activities by adding the notion of expediting tactics, e.g., overtime work or additional shifts. This addition allows JANUS to determine alternative activity costs and durations, based on specific means of shortening activity durations. Coupling this extension with the addition of multiple project participants, JANUS can now help to predict specific expediting tactics that particular project participants may be willing to use in order to expedite the project.

3.2.2. Traditional time-cost tradeoff analysis

Traditional time-cost tradeoff analysis has focused on the mathematics of working with networks of activities, with manipulating changes to the activities and with managing the effect of those changes on the overall network. This section will review some of these traditional ideas.

Procedures to analyze time-cost tradeoff decisions are standard fare in modern project management coursework [Moder 83]. Most of this type of analysis is built upon the ideas outlined in the Du Pont work of the 1950’s. Traditional time-cost tradeoff analysis provides mathematical rigor, but requires all the expediting alternatives to be explicitly spelled out up front. That is, normal and crash cost and duration data is required for each activity. Such analysis also makes no provision for any organizational concerns such as the
relationships among the project participants or for the varying perceptions of the value of time to those participants.

JANUS addresses activities’ up-front data requirements by inferring activity time-cost relationships from the tactics that may be used to expedite the activity. It addresses the organizational concerns of the project by explicitly modeling the various project participants and their contractual relationships.

[Crowston 67] provides a good example of a mathematically-rigorous approach to time-cost tradeoff analysis. The authors formulate CPM as an integer programming problem which includes alternate activities for some tasks. These alternate activities are incorporated into the network by the use of decision nodes.

The heuristic technique the authors use to solve their problem effectively includes activity cost slopes and a procedure to find the activity’s NIL, although their terminology is somewhat different. Alternative methods of performing an activity – each containing a specific cost, a specific duration and a set of technological dependencies – are defined before the analysis can begin. The analysis then considers all competing methods that have been defined to accomplish each task and selects those which give the lowest overall project cost.

This approach requires all expediting alternatives for each activity to be fully defined before the analysis can begin. It also provides no means of considering varying project participants’ goals on the schedule. JANUS addresses these limitations by the addition of explicit models of project participants and by modeling broadly-applicable expediting tactics separate from activities.

[Barber 88] describes an implementation of Siemens’ algorithm for time-cost tradeoff analysis. This algorithm utilizes six “constraints” on an activity to determine an effective cost for that activity. These constraints are:

- the feasibility of expediting the activity;
- the cost of expediting the activity;
- the effect on available resources of expediting the activity;
- the desirability of continuing to expedite one activity rather than jump to another;
- the number of paths that share the activity;
- how close to the current time the activity occurs in the network.
Activities are selected for expediting based on their "effective cost," a heuristic value derived from evaluating these six constraints. The analysis is performed step by step with a single activity being expedited or sold back at each step. After each step, the network and project costs are updated.

A particularly interesting feature of this work is the abstraction hierarchy it introduces for expediting strategies. This hierarchy describes a number of different ways of expediting an activity:

- **Control** implies getting more out of the available resources. Control can be achieved either through *incentives*, a *better* (more efficient) *organization*, or *closer monitoring* of the work.
- **More time** creates more working time without increasing project time, primarily through *shifts* and *overtime*.
- **Resources** can be added to an activity to expedite its completion, either of *men, equipment, or materials*.
- **Change contract** can be used for subcontracted activities to achieve duration reductions.
- **Change specification** may be a valid option in some cases where material delivery is affecting completion of an activity.
- **Abort** is the final catch-all option which effectively removes the activity from the project.

Each activity’s time-cost tradeoff function is based on this abstraction hierarchy.

This is a very interesting piece of work. It implements a well-established mathematical model of time-cost tradeoff analysis. It separates the activities from the possible strategies for expediting them and maps them together when determining an activity’s expedited cost and duration. It does not make any provision for evaluating the effect on the schedule of a number of differing project participants and their differing views of the value of time.

### 3.2.3. Extensions to traditional CPM

A substantial amount of work has been done to extend the capabilities of CPM using AI techniques.

- [Hendrickson 87] describes a model that determines individual activity durations in a limited domain.
• [Levitt 85] describes a system that reasons about historical performance in an attempt to predict the durations of individual activities later in the project.

• [Logcher 88] proposes a richer activity representation & OOP as a good paradigm for construction scheduling.

• [De La Garza 87] describes a hierarchical approach to activity representation.

3.2.3.1. MASON

[Hendrickson 87] describes MASON, a system to estimate single activity durations in the domain of masonry construction. MASON develops activity durations in a hierarchical, rule-based fashion considering:

• quantities of work,
• prevailing weather conditions,
• numbers of crews and
• estimates of expected productivity.

MASON begins with estimates of maximum possible productivity from references or historical experience. It then adjusts these productivity figures based on details of the work or of the site, e.g., the height of the workplace off the ground or the prevailing weather conditions at the site. MASON also determines estimates of down time to be coupled with its expected productivity estimates in order to properly calculate activity duration. After making a preliminary determination of the activity duration, MASON applies another level of adjustment for factors which require a preliminary estimate. In this fashion, it determines an estimate for the duration of a given activity.

MASON focuses on the reasoning needed to determine individual activity durations, which can supplement traditional CPM analysis. This type of analysis could also be used to support time-cost tradeoff analysis, but such an application is not discussed in this work. This work focuses on a single application domain and does not address the roles of different participants. It does not address the value of time in any manner.

3.2.3.2. PLATFORM

[Levitt 85] describes PLATFORM, a system which utilizes user-supplied optimistic and pessimistic estimates of activity durations, along with actual durations observed to date, to forecast expected durations for remaining activities. The way in which this is implemented is through the use of “risk factors” associated with each activity. After gaining
some data about actual project performance, PLATFORM compares actual durations with
the original optimistic and pessimistic estimates. It then reasons about which risk factors
may have affected the actual performance and become either knights (those risk factors
having a positive effect on activity durations) or villains (those having a negative effect on
activity durations). It then revises the durations of as-yet-uncompleted activities affected by
knights downward and those affected by villains upward.

PLATFORM extends traditional CPM analysis by focusing on the risk factors
present in the project. It identifies knights and villains explicitly and attempts to predict
future behavior based on experience. PLATFORM makes no attempt to evaluate time-cost
relationships among activities in a network. It also ignores the varying viewpoints of
different project participants.

3.2.3.3. Logcher

[Logcher 88] discusses knowledge-based systems in the context of project control
systems. In a sense, it builds upon the view described in [Levitt 85] of activities as self-
aware entities or objects and suggests object-oriented programming as a reasonable
paradigm for project scheduling. Object-oriented programming allows objects to be
hierarchically defined and allows object behavior to be represented as procedures assigned
to attributes of the object. As an example, Logcher describes how a CPM analysis might be
implemented in an object-oriented manner.

JANUS utilizes Logcher's object-oriented paradigm for activity and schedule
network representation. It does not use the CPM algorithm strictly as outlined in the paper,
but makes use of similar ideas.

3.2.3.4 De La Garza & Ibbs

[De La Garza 87] presents a discussion of knowledge representation for
construction scheduling. It describes a semantic network that ties together project
components, specifications, activities and schedule impacts. While its approach to schedule
impacts is somewhat different than that of knights and villains described in [Levitt 85], it is
similar and is integrated with the other aspects of construction scheduling.
3.3.4. Summary of the limitations of traditional project management approaches to time-cost tradeoff analysis

Traditional approaches to time-cost tradeoff analysis have been limited by:

- the need for a great deal of up-front, activity-specific data, most notably two or more estimates of activity cost and duration for each activity
- lack of realistic activity time-cost relationships
- lack of consideration of organizational aspects of the analysis, particularly the differing motivations of the various project participants toward activity expediting.

JANUS addresses these limitations by:

- separating activity representation from that of expediting tactics, allowing little activity-specific data to be required up front and also providing more realistic activity time-cost relationships to be used in the analysis
- explicitly modeling project participants and their perceptions of the value of time, allowing for a more realistic representation of the activity expediting environment in which the analysis occurs.

3.3. Organizational literature

One of the primary contributions of this work is the integration of organizational analysis with traditional project management CPM analysis in the evaluation of project schedules. This section covers some of the pertinent work from the organizational behavior literature.

3.3.1. Organizational Decision-Making

[March 88] examines decision-making in organizations. In the context of the construction industry, an organization could be a project organization as well as a single corporate entity. March’s work reviews the traditional statistical decision theory and microeconomic theory interpretation of decision-making as intelligent choice – intentional, consequential and optimizing. This familiar view of decision-making assumes that decisions are based on well-defined preferences of the decision maker, on presumed complete understanding of the possible alternatives available and on expectations about outcomes of each of those different alternatives.
There are a number of problems with this traditional view of decision-making within an organization, not the least of which is that it often involves multiple actors with conflicting and often inconsistent preferences—a much more apt view of a project organization than as a single rational decision-maker. Such an environment is better described as a political system. In the realm of economics, the standard approach to handling conflict in such a system is contracting. Contracts provide a means of managing the incentives of the various parties so that they display mutually-beneficial behavior. Classical theory regarding conflict resolution in political systems focuses on the mechanisms of power and exchange. Power has proved to be a disappointing concept in this context, being difficult to measure and often used as an easy way of explaining unanticipated or otherwise unexplainable phenomena. Perhaps a better way of thinking about power is as control over resources. That is, the individuals in an organization use resources at their command to pursue their own goals.

Among the resources available to many is information. One interesting question that arises from this idea of information as a resource to be used in pursuing goals is that of misrepresentation of the information if such lying proves to be in the interest of the individual or group. Ideas of manipulating data are familiar to us all. For many, accounting and its magic numbers provide endless opportunities for manipulating the data that feeds the decision-making process within their organizations. Given that such behavior does exist—and we have all observed it—the notions of trust and reputation among the participants takes on significant meaning.

3.3.2. Transaction cost accounting: Neoclassical economics

[Williamson 79] discusses the importance of considering ongoing relationships between firms in addition to the letter of the contract between them when analyzing firm behavior. This notion of “relational contracting” implicitly contains the concepts of trust and reputation that March suggested should be considered when evaluating a political system. Williamson’s view features two primary dimensions: the nature of the investment and the frequency of the transactions between the parties. Investments are viewed as either “non-specific,” such as commodity purchases, or “idiosyncratic,” like custom-designed goods where one or both parties may invest significant costs which can only be amortized over a single transaction. The frequency of transactions between two parties is described as either “occasional” or “recurrent.” Relational or obligatory contracting occurs for recurrent
idiosyncratic transactions. Figure 3 shows these relationships and their governance structures in a graphical fashion.

![Diagram](image)

**Figure 3: Governance structures and commercial transaction classifications**

The neoclassical contracting structure governing occasional, idiosyncratic transactions is exemplified by contractor-owner relations. The relational contracting structure for governing recurrent, idiosyncratic transactions is exemplified by contractor-subcontractor relations. Adapted from [Williamson 79].

The description of occasional idiosyncratic transactions applies nicely to many contractor-owner relationships while that of recurrent idiosyncratic transactions accurately represents the world of contractor-subcontractor relationships. Most prime contractors will have special relationships with certain specialty subcontractors with whom they regularly work. These relationships have strong components of Williamson’s relational contracting, wherein the parties behave as if there are more dimensions to consider in a decision than just those involved with the single current transaction – and trust and reputation are among the primary dimensions.

[Gunnarson 82] elaborates on the fluctuating nature of demand in the construction industry, particularly the demand for the primary product – the constructed facility itself – but also demand for the specialized resources needed to build that facility. Construction
firms use several different strategies to deal with these demand fluctuations. Some, especially those in the public works sector, develop mobility: supervisory staff willing to travel as necessary, elaborate training programs for local labor and capabilities of moving heavy equipment to remote jobsites. Others, especially those in the building sector, develop specialization. Different firms do earthwork, structural steel erection, cladding, masonry, plumbing, etc. They each gain their competitive advantages from their technical expertise in a very narrow domain and by gaining economies of scale by spreading their workload among many jobs and customers.

Gunnarson presents a transaction cost analysis of his “markets and hierarchies” framework to interpret the prevalence of subcontracting in the building sector of the construction industry. He focuses on the importance of asset specificity to that analysis. From the point of view of production costs only, as assets become more specialized, there is a decreasing incentive to buy that asset through a market transaction, since the producer of the specialized asset does not have the huge market necessary to give it economies of scale over anyone else producing the same asset. On the other hand, transaction costs increase dramatically in the marketplace as asset specificity increases, primarily due to the potential for extortion on the part of the producer. It is the combination of the production costs and the transaction costs that determines whether a given asset is more favorably purchased in a market (through a subcontract) or in a hierarchy (produced by the contractor).

The building construction environment is particularly interesting in that it is both a market – a contractor hires subcontractors through some kind of market transaction, and a hierarchy – that contractor has supervisory responsibility over the employees of the various subcontractors on the jobsite. Partly because of this unique feature of contractor-subcontractor relations, contracting practices in the industry are rather elaborate. The necessity for such elaborate practices tends to increase the market transaction costs of the subcontract. Contractors minimize the impact of these costs in several ways. They use standardized contracts, which help reduce the prospects of disputes. As [Williamson 79] points out, they also make use of arbitration to a great degree and they develop ongoing relationships to help minimize the size of the transaction costs. Gunnarson concludes that subcontracting is dictated by production cost economies and that the transaction costs of the market are mitigated by the contractors in the ways mentioned above.

[Stinchcombe 85] looks at the same ideas from a slightly different viewpoint. It points out that contracts are often used in situations where theory predicts hierarchical
control. Its conclusion is that in such situations, the contracts contain elements of hierarchies in order to allow clients better control of uncertainty. In effect, such contracts create hierarchies containing elements of different organizations for their administration.

3.3.3. Risk in construction

[Levitt 80] develops a conceptual model of risk allocation in the construction domain. A primary issue in this work is that the participants in a construction project have different approaches toward risk: the project owner ultimately pays for the risk and the general contractor ultimately manages it. Appropriate allocation of risk must consider both the ability of a participant to absorb risk and its ability and incentive to manage it. Major features of this model include recognition of the consequences of risk, the perceived ability of the participant to control – and hence accept – risk and the effects of these factors on contract clauses.

3.3.4. Summary of the organizational theory

To summarize the review of the organizational literature:

- [March 88] provides a theoretical basis for considering trust and reputation among actors within an organization when evaluating decision-making behavior.
- [Williamson 79] describes relational contracting between firms who engage in recurrent idiosyncratic transactions.
- [Gunnarson 82] explains that production cost economies dictate the extensive use of subcontracting in the construction industry and that transaction costs are mitigated by the parties involved in a number of ways, including the use of standardized contracts, arbitration and developing ongoing relationships.
- [Stinchcombe 85] suggests that contracts are often used to create hierarchical control structures in highly uncertain situations.
- [Levitt 80] discusses risk allocation in construction and the importance of considering ability to absorb risk and ability and incentive to manage it.

JANUS makes use of several of these ideas.

- It provides a means of reasoning about contractor-subcontractor relationships by modeling the participants and the type of transaction taking place between them.
• It provides a means of reasoning about the effect of contract clauses on participant behavior by modeling the contracts.

This research will add to the field a representation scheme and model to simulate the effect on a project schedule of changes in organizational relationships.

3.4. Contributions of the research

To recap the major points made in this review of the literature:
• Frame based knowledge representation, OOP and model-based reasoning provide powerful tools for extending the capabilities of traditional schedule analysis systems.
• Traditional and AI programming techniques have been used in the past to model organizations and new techniques are evolving.
• CPM provides powerful tools for analyzing schedules but does not provide for consideration of organizational concerns.
• Traditional time-cost tradeoff analysis is another powerful tool but does not provide adequate support for determining activity time-cost relationships or for consideration of organizational concerns.
• CPM has been extended to calculate activity durations and to predict future performance based on past performance.
• OOP has been used as a convenient programming paradigm for project scheduling.
• Contractor-subcontractor relationships are complex and their analysis requires consideration of ongoing relationships.
• Contracts often create hierarchies for their administration.

The research described in this thesis develops a knowledge-based model of time-cost tradeoff analysis. It combines organizational theory with project management analysis to simulate the effect of contract strategies on project schedules. It applies the ideas of relational contracting and of differing perceptions of project participants to its analysis.
This research project extends knowledge in the following areas in the following ways:

- **AI**: It provides a prototype system using the ideas of model-based reasoning combining the domains of project management and organizational behavior.
- **CPM**: It extends traditional CPM analysis to include consideration of organizational concerns and it determines activity time-cost relationships from separate representations of activities and expediting tactics.
- **Organizations**: It supplies a prototype knowledge-based system that incorporates Williamson's notion of relational contracting into traditional CPM analysis to allow consideration of multiple participants in the analysis.

This research does not attempt to:

- formulate a better CPM algorithm
- introduce a better way of performing resource-constrained scheduling
- provide optimum solutions to the time-cost tradeoff problem
- provide an alternative conceptual model of organizational relationships.
Chapter 4. Research Approach

4.1. Goals of the research

During the past several years, Professor Raymond E. Levitt has led a group of students and researchers at Stanford in investigating the application of AI concepts to project management and planning. The work described in this thesis utilizes and builds upon other work from that group. JANUS incorporates an activity representation similar to that of [Darwiche 88]. It has also been designed to take the output of two knowledge-based planning systems – SIPEC [Kartam 89] and OARPLAN [Darwiche 88] – as its input. This input data is supplemented with information regarding the organizational and contractual aspects of the project and with information about feasible means of expediting activities.

Specific primary goals of this research project are to:
- model the process of time-cost tradeoff analysis and
- develop a framework for analyzing a project participant’s perception of the value of time.

The author contends that traditional approaches to time-cost tradeoff analysis have been rendered effectively useless by two sets of assumptions: those regarding activity time-cost relationships and those regarding project participant perception of the relative value of saving time on the project. Both primary goals of this research project aim to rectify these faulty assumptions.

The model of time-cost tradeoff analysis developed through this research will provide more valid activity time-cost relationships than traditional approaches currently give. It will also provide for consideration of multiple project actors. As part of this consideration of the effect of multiple actors on the schedule analysis, the model will include a way of determining what each actor feels time is worth – specifically, how many dollars it is willing to spend to reduce a given activity’s duration by a day.
Secondary goals of this research project include:
- determining a set of feasible expediting tactics for construction activities
- defining the key factors necessary to enable a useful time-cost tradeoff analysis of a construction schedule
- describing important relationships among the elements of the analysis.

Each of the secondary goals are needed to support the primary goal of developing the model. The approach taken in this research project toward providing more valid activity time-cost relationships is to separate the representation of the activity from the representation of the means of shortening it – the expediting tactic. It is therefore necessary to develop a set of expediting tactics to be used in the analysis.

Developing the model also requires the explicit definition of all the various factors that will be used in the time-cost tradeoff analysis. Once the factors are defined, appropriate representation schemes can be developed, their roles in the analysis can be defined and the relationships among them can be described.

4.2. The approach to address these goals

To address these goals, the JANUS prototype represents organizational and contractual variables in the same architecture as the CPM analysis and cost computations. It is implemented through the use of object-oriented programming. Activities, participants, contracts, expediting tactics and crews are represented as frames, making extensive use of inheritance to define specific attribute values and behavior. Object behavior is implemented as methods. Reasoning is conducted with demons attached to object attributes.

4.3. Validation of the system

This research project consists of two parts: the thesis and the prototype system, JANUS. The prototype system will be validated through an example project used by Professor John Fondahl in his project management course at Stanford. That project and a description of how JANUS works through it will be discussed in Chapter 6. The knowledge within JANUS will be validated by evaluation of the schedule it produces. The overall system will be judged successful if it can be configured with different knowledge and shown to give valid results based upon that knowledge.
4.4. Development style of prototyping: "maximum anxiety"

[Kunz 89] describes the "maximum anxiety heuristic" as a means of guiding knowledge system development. JANUS was developed in such a manner. The system's purposes, its representation, its reasoning, its interfaces and its testing evolved opportunistically, based upon whichever issue caused the most anxiety at the moment. This approach allowed all issues to be understood at a more-or-less equal depth during the entire development cycle.

4.5. Choice of KEE as a programming environment

IntelliCorp's KEE - Knowledge Engineering Environment - was selected for implementing JANUS for a number of reasons. KEE is currently the state-of-the-art in knowledge system development environments. It provides a user interface to and support for production rules, frame representations, object-oriented programming and truth maintenance. Its frame representation capabilities and object-oriented programming tools provided the primary impetus for its use. The author also had access to a number of computers, Texas Instruments microExplorers, which would run the KEE system.
Chapter 5. Implementation of JANUS

JANUS is named after the Roman patron of the beginning and end of things – appropriate for a system that decides when to begin and end activities in a project.

5.1. The components & their interactions

JANUS contains several distinct components:
- its CPM portion, which is contained within the activity representation
- its day value determination portion, which is contained within the participant representation and utilizes contracts
- its duration calculation portion, which is also contained within the activity representation and utilizes tactics and crews.

This section describes these different components, their representation schemes, internal procedures and interactions. Appendix 1 contains details of each of the root objects used in JANUS – activities, crews, participants, projects and tactics. Appendix 2 contains the Lisp code used to implement all object behavior. Appendix 3 describes the KEE active values, or demons, used to implement JANUS.

5.1.1. Activities & CPM calculations

This section contains an overview of the activity representation scheme and CPM algorithm used in JANUS.

5.1.1.1. The object-oriented paradigm and how it works with CPM calculations

Object-oriented programming views the world as consisting of objects which communicate among one another by sending messages. Objects contain attributes and behavior. Attributes have values. Behavior is defined by procedures attached to attributes. This behavior is often triggered by the receipt of a message from another object.

In the example pier project, the activity Demolition has an attribute called Duration, the initial value of which is 15, meaning 15 days. It also has some behavior called Calculate.ES, which is defined by a procedure which determines its early start date by collecting the values of all its predecessors, selecting the maximum from among them and assigning that value to its own attribute called Early.Start. It actually executes this behavior
when it receives a message from its predecessor to perform its forward pass calculations, the calculation of its early start being part of that larger procedure.

![Diagram](image)

**Figure 4: Graph of the Activities hierarchy in JANUS**

Subclass relationships are denoted by solid lines and instance relationships by dashed lines. Reasoning may be done about activity class membership. For example, activities that are members of the class LaborDrivenActivities may be expedited differently than activities that are members of the class EquipmentDrivenActivities.

### 5.1.1.2. Activity representation & attributes

Activities are represented in JANUS as frames. Figure 4 shows a graph of the hierarchy of activities implemented in JANUS for the example pier project. (See Chapter 6 for a description of the example pier project.) Activity frames contain slots for a number of attributes, including:

- standard CPM data such as predecessors and successors, early start, float, criticality, etc.
- crew preference, i.e., that crew which performs the activity in the most efficient manner possible
- estimating data, including the activity’s total manhour and material estimates
• time-cost tradeoff data, including normal cost and duration – attributes of
the activity itself, and current cost and duration – dependent upon the
activity’s current tactic and current crew
• expediting data, notably the expediting tactic currently in effect
• the activity’s responsible participant.

Activity frames also contain slots that hold procedures for:
• feeding estimating data into the system
• initializing activities and the network
• performing the CPM calculations
• calculating activity estimates based on the manhour and material estimates,
  the activity’s preferred crew and its current expediting tactic
• calculating activity duration based on the current estimate, crew and
  expediting tactic.

Appendix 1 contains a description of all the slots defined for the Activities frames in
the knowledge base JANUS. The CPM-related functions are in the file JANUSCPM.LISP,
which is included in Appendix 2.

5.1.1.3. Overview of the CPM method used

The CPM algorithm used by JANUS is implemented by message-passing between
activities. (See Appendix 2 for a listing of the code used to implement the CPM in JANUS.
All functions associated with the CPM calculations are contained in the file
JANUSCPM.LISP.) To begin with, the network is initialized, i.e., each activity is
messedged to clear its predecessors slot, initialize itself, pass the network initialization
message to each of its successors and finally to notify each of its successors that it is its
predecessor. The activity initialization sets the values of all its CPM-related slots (e.g.,
early start, total float, etc.) to 0 and calculates its duration.

The first activity in the network is then messaged to calculate the network. The
network calculation is done by first messaging all activities downstream of the messaged
activity to set two flags to false: one to show that the forward pass is complete, the other to
show that the backward pass is complete. Each activity then receives the message to do its
forward pass. The forward pass consists of calculating its early start and early finish dates.
In order to calculate its early start, the activity must collect the early finish dates of its
predecessors. In order to do this, it first checks their forward pass flags: if the flags are
true, they have completed their forward pass calculations and their early finish dates are hence OK. If the flags are false, they must first perform their forward pass calculations, so they are given the message to do so. After collecting all its predecessors’ early finish dates (hence performing the forward pass calculations on all those activities requiring it), the activity sets its early start to the maximum value of those dates. It then sets its early finish to that value plus its duration and its forward pass flag to true.

The backward pass is initiated when an activity has no successors - the last activity in the network. (This implementation restricts a network to a single end activity. JANUS overcomes this limitation by using a dummy activity of 0 duration which is called Project.Complete.) This activity first messages itself to do its backward pass calculations, then messages each of its predecessors to do theirs. The backward pass calculates the activity’s late finish, its late start, its total float and its criticality. It does this in the same way as it does the forward pass calculations, except that its successors’ late start dates are collected and their minimum is used as its late finish. After performing these calculations, the activity sets its backward pass flag to true.

After initialization, the network is automatically updated by demons that send the message to an activity to calculate the network whenever its duration is changed. The duration, in turn, is updated by demons attached to each activity’s manhour estimate, current crew and current tactic slots. That is, whenever an activity’s current cost, its current crew or its current expediting tactic are changed, its duration is automatically recalculated and the network is automatically updated.

5.1.2. Participants & determination of day value

This section contains an overview of the participant and contract representation schemes and day value reasoning process used in JANUS.

5.1.2.1. Participant representation & attributes

JANUS represents project participants as frames. Figure 5 shows the hierarchy of participants defined in JANUS for the example pier project. Each participant frame has slots for:

- the participants it employs,
- the participant for whom it works,
• the type of contract under which it works,
• the value in dollars of one day to this participant on this project.

![Graph of the PARTICIPANTS in the JANUS Knowledge Base]

Figure 5: Graph of the Participants hierarchy in JANUS

Note that Piling.Sub.1 inherits information from both the class of Subcontractors and the class of Custom.Suppliers, since it is both fabricating the precast concrete piles and installing them in the field. Inheritance of slot values is managed in KEE by the order in which parents are defined for a child.

Procedures associated with participant frames include those needed to:
• input attribute data
• message its employees
• fill in its Employed.by attribute on receiving a message from its employer
• determine its dollar value of one day saved on the project.

Appendix 1 contains a listing of all the slots defined for the Participants frames in the knowledge base JANUS. The functions for project participants are in the file JANUSPAR.LISP, which is included in Appendix 2.

5.1.2.2. Contract representation & attributes

Contracts are represented in JANUS as frames. As currently implemented, JANUS reasons only on the class of contract, i.e., whether the contract is a member of the class Cost.Plus.Contracts or is a member of the class Lump.Sum.Contracts and on the class of the participant, i.e., whether the participant is a general contractor or not. The contract hierarchy is shown in Figure 6. Because of this low level of reasoning, the contracts frames contain no attributes. Further extensions could add more subtle variations on the characteristics of contracts.
Figure 6: Graph of the Contracts hierarchy in JANUS

JANUS currently reasons only about the class of the contract, so this hierarchy is not very elaborate. This structure, however, allows the model to be easily extended to incorporate deeper reasoning about specific contract clauses and their effect on the perception of a participant with respect to the value of a day.

5.1.2.3. Relationships between participants and contracts

Participants are related among one another by contracts. As currently implemented in JANUS, each participant may have a number of employees, but only one employer. Its contract is the contract it has with its employer, so each participant has just one contract.

5.1.2.4. Reasoning about day value

The value each participant puts on a day, its Day.Value, is an attribute of the participant. The owner is given an explicit value by the user for its day value. JANUS determines the value of a day for each other participant in the project. Those values are inferred from their other attributes and their locations in the class hierarchy. As currently implemented, JANUS reasons in a very shallow fashion about participant day value. Suppliers – members of the class of Suppliers – simply get the day value of their employer. If employed on a cost plus contract, contractors get the day value of their employer. Otherwise, general contractors get the value of their daily overhead rate and all others get 0 for the value of a day. In other words, on a lump sum contract, general contractors will spend up to their daily jobsite overhead rate while the others will spend nothing to expedite the project.

The structure of the participant and contract frame representations allows JANUS to be easily extended to incorporate a more extensive analysis. For example, participants can be given a propensity to cooperate based on the age of the company (greater age indicating past success and hence a greater tendency to cooperate), on the size of their contract with
respect to their annual sales (a contract comprising half of the company’s annual sales should elicit a great deal of effort on behalf of the owners to cooperate) and on the details of the contract it has with its employer (cost plus contracts should allow the employer more say over the participant’s values and lump sum contracts should make the participant more careful with respect to its own indirect costs and/or the efficient use of its resources). It is in this area that Williamson’s ideas of relational contracting can be best applied.

5.1.3. Tactics & reasoning about activity duration

Determining activity duration requires a number of different kinds of information:

- estimating information, particularly the manhour estimate for the activity
- crew information, including the number of crew members, their costs and the equipment they use
- expediting tactic information, including the efficiency of the tactic, its costs and the number of hours it works in a day.

JANUS uses all this information when determining activity durations during its analysis. Figure 2 shows how activities and tactics are combined to develop effective cost slopes for the activities. When reasoning about activity duration, JANUS considers:

- only critical activities,
- the last expediting tactic employed on the activity and
- the value of a day to the activity’s owner.

5.1.3.1. Tactic representation & attributes

The tactics currently defined in JANUS primarily correspond to the more time category mentioned in [Barber 88]. This focus is deliberate primarily because this category is the most widely applicable from the point of view of many different project participants.

Tactics are represented in JANUS as frames. Figure 7 shows the hierarchy of expediting tactics that have been defined for the example pier project. Each tactic has slots for:

- shift information, including the number of shifts worked per day and the number of hours worked per shift
- the efficiency of the tactic, as a percentage less than or equal to 100%
- the cost slope multiplier of the tactic, i.e., its relative cost.
Tactic frames also contain slots that hold procedures for:

- calculating the tactic’s hours worked and its hours paid
- determining the tactic’s cost premium and its duration improvement.

![Graph of the TACTICS Unit in JANUS](image)

**Figure 7: Graph of the Tactics hierarchy in JANUS**

The tactics shown in this graph are all those initially defined and used in JANUS. Most of these tactics represent common means of allowing more time to be worked on the project without working more days. They thus fall within the bounds of the *more time* category of expediting strategies described in [Barber 88]. Additional tactics – including those to change crew size or select alternate methods – can also be defined.

Appendix 1 contains a listing of all the slots defined for the Tactics frames in the knowledge base JANUS. The functions dealing with project tactics are in the file JANUSTAC.LISP, which is included in Appendix 2.

### 5.1.3.2. Crew representation & attributes

Crews are represented in JANUS as frames. Figure 8 shows the hierarchy of crews that has been defined for the example pier project. Each crew has slots for:

- crew members, the different trades which appear in the crew
- crew size, the number of full-time equivalent individuals in the crew
- direct cost per manhour, the average base wage of the crew members
- equipment, a list of the various pieces of equipment used by the crew
- equipment cost per crew hour, the hourly cost of all the equipment used by the crew
- labor fringe rate, the average fringe benefits rate of all crew members.
Figure 8: Graph of the *Crews* hierarchy in JANUS

Note that erection and formwork crews inherit information from both carpentry crews and from concrete crews. The inheritance has been set up so that carpentry crew data overrides that from concrete crews for both erection and formwork crews.

Appendix 1 contains a description of all the slots defined for the *Crews* frames in the knowledge base JANUS. The functions dealing with crews are in the file JANUSCRW.LISP, which is included in Appendix 2.

### Table 1: Formulas for activity estimates

5.1.3.3. *Relationships between activities, crews and tactics*

Activities, crews and tactics are all necessary to determine activity durations. To begin with, activity cost estimates are dependent on the number of manhours estimated for the activity, on the crew assigned to perform the work and on the expediting tactic employed. Table 1 contains the formulas which determine the activity cost factors.
\[ C_N = \text{Normal Cost} \]
\[ D_N = \text{Normal Duration} \]
\[ C' = \text{Expedited cost} = C_N \times (1 + \text{Premium}) \]
\[ D' = \text{Expedited duration} = D_N \times (1 / (1 + \text{Dur Imp})) \]

The cost slope of an activity depends upon the cost over and above its normal cost – its cost premium – and upon the amount its duration has been reduced below its normal duration – its duration improvement. Both of these factors are characteristics of expediting tactics and can be used to determine relative cost and schedule reduction among a number of tactics. Working from these two percentages, the activity's cost slope can be expressed in terms of its normal cost per day, its cost premium and its duration improvement:

\[
\text{Cost Slope} = \frac{\Delta C}{\Delta D} \\
= \frac{(C' - C_N)}{(D' - D_N)} \\
= \frac{(C_N (1 + \text{Premium}) - C_N)}{(D_N (1 / (1 + \text{Duration Imp})) - D_N)} \\
= \frac{C_N (1 + \text{Premium} - 1)}{D_N ((1 / (1 + \text{Duration Imp})) - 1)} \\
= \frac{C_N}{D_N} \times \text{Premium} \times \frac{(1 + \text{Duration Imp})}{(- \text{Duration Imp})}
\]

Cost Slope Multiplier = \text{Premium} \times \frac{(1 + \text{Duration Imp})}{(- \text{Duration Imp})}

The labor estimate, the labor fringe estimate and the equipment estimate – all dependent on the crew and on the expediting tactic – are added to the activity's material estimate to get the total estimate for the activity. The activity's crew hour estimate is then used along with the current crew's crew size and the current tactic's effective hours worked per day to determine the activity's duration. The cost over and above the activity's normal cost is put into its \textit{Delta.Cost} slot. The duration improvement from its normal duration is put into its \textit{Delta.Duration} slot.
An activity’s cost slope is dependent on its normal cost per day and on its cost slope multiplier (see Figure 9). This cost slope multiplier is calculated from the activity’s cost premium and from its current tactic’s duration improvement. Table 2 contains the formulas for these factors.

\[
\text{Activity’s Cost.Slope} = \frac{(\text{Next.Cost} - \text{Normal.Cost})}{(\text{Next.Duration} - \text{Normal.Duration})}
\]

\[
\text{Activity’s Cost.Premium} = \text{Current.Cost} - \text{Normal.Cost}
\]

\[
\text{Tactic’s Cost.Premium} = \frac{\text{Hours.Paid}}{\text{Hours.Effective}}
\]

\[
\text{Tactic’s Hours.Paid} = \text{Straight.Hours} + 1.5x.\text{Hours} + 2x.\text{Hours} + \text{Overtime.Premium.Hours}
\]

\[
\text{Tactic’s Hours.Effective} = \text{Hours.Worked} \times \text{Efficiency}
\]

\[
\text{Tactic’s Hours.Worked} = \text{Straight.Hours} + 1.5x.\text{Hours} + 2x.\text{Hours} - \text{Shift.Premium.Hours}
\]

\[
\text{Tactic’s Overtime.Premium.Hours} = (1.5x.\text{Hours} \times .5) + 2x.\text{Hours}
\]

\[
\text{Tactic’s Shift.Premium.Hours} = 0.5 \text{ for 2 shifts, 1.5 for 3 shifts}
\]

\[
\text{Table 2: Formulas for activity cost slope factors}
\]

5.1.3.4. Reasoning about activity tactics

Each activity’s duration is simply determined as its crew hour estimate divided by its current tactic’s effective hours per day. All activities begin with the \textit{Straight.Time.Only} tactic as their current tactic. As currently implemented, JANUS simply cycles through the available tactics. If working straight time hours only does not achieve the project’s goals, JANUS tries each of the remaining tactics in order of increasing duration improvement: first \textit{6.Eights} (6 days of single 8-hour shifts), then \textit{5.Tens} (five days of single 10-hour shifts), then \textit{6.Tens}, (six days of single 10-hour shifts), then \textit{5.Day.Swing} (5 days of two 8-hour shifts per day), and finally \textit{5.Day.Graveyard} (5 days of three 8-hour shifts). (See Table 3 for a list of cost slope factors for the various tactics.) JANUS will continue to select increasingly-quick expediting tactics only as long as the added cost of performing the activity with the new tactic divided by the number of days reduction in duration remains below the activity’s responsible participant’s day value.
Table 3: Tactic cost slope factors

<table>
<thead>
<tr>
<th>Expediting Tactic</th>
<th>Efficiency</th>
<th>Cost Premium</th>
<th>Duration Improvement</th>
<th>Cost Slope Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight.Time.Only</td>
<td>1.00</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0</td>
</tr>
<tr>
<td>6.Eights</td>
<td>0.95</td>
<td>0.1403</td>
<td>0.1400</td>
<td>-1.142</td>
</tr>
<tr>
<td>5.Tens</td>
<td>0.95</td>
<td>0.1578</td>
<td>0.1875</td>
<td>-1.000</td>
</tr>
<tr>
<td>6.Tens</td>
<td>0.90</td>
<td>0.2835</td>
<td>0.3050</td>
<td>-1.213</td>
</tr>
<tr>
<td>5.Day.Swing</td>
<td>0.80</td>
<td>0.2903</td>
<td>0.5500</td>
<td>-0.818</td>
</tr>
<tr>
<td>5.Day.Graveyard</td>
<td>0.70</td>
<td>0.5238</td>
<td>0.9687</td>
<td>-1.064</td>
</tr>
</tbody>
</table>

The strategy implemented represents one of incremental activity duration improvement. JANUS could be configured to pursue another strategy by changing the order in which expediting strategies are tried. It could also be configured to automatically order all defined tactics by some criteria, such as cost premium, duration improvement or cost slope multiplier.

5.2. OOP in KEE

Objects or frames are implemented in KEE as “units.” Object attributes are implemented as slots in those KEE units. Each slot has facets which restrict the value(s) that may be assigned to the slot. Object behavior is implemented in KEE as functions or “methods” assigned as the values of slots of the object. Message-passing can be performed in KEE either programmatically or through the user interface. Programmatic control is exercised by the use of demons – KEE “active values” – attached to object slots. These active values are procedures that are executed whenever the value of the slot to which they are attached is changed.
Chapter 6. Example Problem

A test case has been adapted from a problem used by Professor John Fondahl in his course at Stanford in project management. This project is often referred to as “Fondahl's Famous Pier Problem.” Professor Fondahl used this case to demonstrate his non-computer approach to time-cost tradeoff analysis to his students.

6.1 Project Network

Figure 10 shows the initial state of the pier project schedule used to demonstrate JANUS. The network provides a complex enough case to exercise JANUS effectively. The information shown in this schedule is supplemented in JANUS with additional estimating.

Figure 10: "Fondahl's Famous Pier Problem," adapted for JANUS

The project schedule is adapted from an example used by Professor John Fondahl to illustrate his non-computer approach to analyzing time-cost tradeoff problems to his project management classes. This figure shows the initial state of the schedule network before JANUS begins its analysis.
crew, expediting and participant data.

### 6.2 Project Estimate

Table 4 shows the initial state of the estimate for the example pier project. The manhour estimate and the material estimate were input and the other values were calculated by the system, based on the crews and tactics assigned to each activity.

<table>
<thead>
<tr>
<th>Act #</th>
<th>Description</th>
<th>Labor Cost</th>
<th>Labor Fringes</th>
<th>Material Cost</th>
<th>Equipment Cost</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Notice to Proceed</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$0</td>
</tr>
<tr>
<td>2</td>
<td>Mobilize</td>
<td>0</td>
<td>0</td>
<td>5,000</td>
<td>0</td>
<td>$5,000</td>
</tr>
<tr>
<td>3</td>
<td>Obtain Concrete Files</td>
<td>0</td>
<td>0</td>
<td>37,000</td>
<td>0</td>
<td>$37,000</td>
</tr>
<tr>
<td>4</td>
<td>Obtain Steel Piles</td>
<td>0</td>
<td>0</td>
<td>8,800</td>
<td>0</td>
<td>$8,800</td>
</tr>
<tr>
<td>5</td>
<td>Obtain Riprap</td>
<td>0</td>
<td>0</td>
<td>16,000</td>
<td>0</td>
<td>$16,000</td>
</tr>
<tr>
<td>6</td>
<td>Obtain Rubber Bumpers</td>
<td>0</td>
<td>0</td>
<td>11,000</td>
<td>0</td>
<td>$11,000</td>
</tr>
<tr>
<td>7</td>
<td>Shop &amp; Casting Yard</td>
<td>50</td>
<td>750</td>
<td>375</td>
<td>500</td>
<td>$1,625</td>
</tr>
<tr>
<td>8</td>
<td>Demolish Existing Pier</td>
<td>375</td>
<td>5,625</td>
<td>2,813</td>
<td>0</td>
<td>$12,188</td>
</tr>
<tr>
<td>9</td>
<td>Manufacture Precast Deck Units</td>
<td>675</td>
<td>13,500</td>
<td>6,750</td>
<td>1,500</td>
<td>$21,750</td>
</tr>
<tr>
<td>10</td>
<td>Manufacture Precast Caps</td>
<td>800</td>
<td>16,000</td>
<td>8,000</td>
<td>2,000</td>
<td>$26,000</td>
</tr>
<tr>
<td>11</td>
<td>Excavate for Thrust Beams</td>
<td>16</td>
<td>320</td>
<td>160</td>
<td>0</td>
<td>$880</td>
</tr>
<tr>
<td>12</td>
<td>Drive Concrete Piles</td>
<td>480</td>
<td>8,640</td>
<td>4,320</td>
<td>0</td>
<td>$23,360</td>
</tr>
<tr>
<td>13</td>
<td>Form &amp; Pour Thrust Beams</td>
<td>250</td>
<td>5,250</td>
<td>2,625</td>
<td>2,500</td>
<td>$10,375</td>
</tr>
<tr>
<td>14</td>
<td>Drive Fender Piles</td>
<td>72</td>
<td>1,296</td>
<td>648</td>
<td>0</td>
<td>$3,504</td>
</tr>
<tr>
<td>15</td>
<td>Backfill</td>
<td>16</td>
<td>320</td>
<td>160</td>
<td>0</td>
<td>$880</td>
</tr>
<tr>
<td>16</td>
<td>Place Riprap</td>
<td>160</td>
<td>3,200</td>
<td>1,600</td>
<td>0</td>
<td>$8,800</td>
</tr>
<tr>
<td>17</td>
<td>Install Girits</td>
<td>552</td>
<td>11,592</td>
<td>5,796</td>
<td>0</td>
<td>$21,988</td>
</tr>
<tr>
<td>18</td>
<td>Set Caps</td>
<td>250</td>
<td>5,250</td>
<td>2,625</td>
<td>0</td>
<td>$9,950</td>
</tr>
<tr>
<td>19</td>
<td>Set Deck Units</td>
<td>375</td>
<td>7,875</td>
<td>3,938</td>
<td>0</td>
<td>$14,938</td>
</tr>
<tr>
<td>20</td>
<td>Cast-in-Place Concrete</td>
<td>480</td>
<td>10,080</td>
<td>5,040</td>
<td>5,000</td>
<td>$20,120</td>
</tr>
<tr>
<td>21</td>
<td>Remove Girits</td>
<td>100</td>
<td>2,100</td>
<td>1,050</td>
<td>0</td>
<td>$3,150</td>
</tr>
<tr>
<td>22</td>
<td>Install Rubber Bumpers</td>
<td>240</td>
<td>5,040</td>
<td>2,520</td>
<td>0</td>
<td>$7,560</td>
</tr>
<tr>
<td>23</td>
<td>Install Deck Hardware</td>
<td>50</td>
<td>1,050</td>
<td>525</td>
<td>0</td>
<td>$1,575</td>
</tr>
<tr>
<td>24</td>
<td>Install Mechanical Services</td>
<td>120</td>
<td>2,880</td>
<td>1,440</td>
<td>0</td>
<td>$5,320</td>
</tr>
<tr>
<td>25</td>
<td>Final Clean-up</td>
<td>75</td>
<td>1,125</td>
<td>563</td>
<td>0</td>
<td>$1,688</td>
</tr>
<tr>
<td>26</td>
<td>Project Complete</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$0</td>
</tr>
</tbody>
</table>

| Project Totals | 5136 | 101,893 | 50,947 | 89,300 | 31,310 | $273,450 |

Table 4: Initial estimate data for the example pier project
Table 5 shows the initial durations and cost slopes calculated by JANUS for each activity in the pier project.

<table>
<thead>
<tr>
<th>Act #</th>
<th>Description</th>
<th>Initial Status</th>
<th>Final Durations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Notice to Proceed</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Mobilize</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Obtain Concrete Piles</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>Obtain Steel Piles</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>Obtain Riprap</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>Obtain Rubber Bumpers</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>Shop &amp; Casting Yard</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>Demolish Existing Pier</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>Manufacture Precast Deck Units</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>10</td>
<td>Manufacture Precast Caps</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>Excavate for Thrust Beams</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>Drive Concrete Piles</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>13</td>
<td>Form &amp; Pour Thrust Beams</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>Drive Fender Piles</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>Backfill</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>Place Riprap</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>Install Girts</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>18</td>
<td>Set Caps</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>19</td>
<td>Set Deck Units</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>20</td>
<td>Cast-in-Place Concrete</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>Remove Girts</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>Install Rubber Bumpers</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>23</td>
<td>Install Deck Hardware</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td>Install Mechanical Services</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>Final Clean-up</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>26</td>
<td>Project Complete</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5: Initial durations and cost slopes and final durations for two test scenarios for the example pier project.

Note that durations of critical activities are shown in boldfaced type.

6.3 Project Participants

Figure 5 shows the hierarchy of project participants defined for the sample pier project. The relationships among the project participants are shown in Figure 11. The organization represented by this structure is common: the owner hires a single prime contractor who is responsible for all construction work. The prime performs some work
with its own forces and subcontracts the remainder. Note that this chart gives no information regarding the types of contracts used by the various parties.

![Pier Project Organizational Chart](image)

**Figure 11: Employment relationships among the pier project participants**

This is a graph of the *Employ* relationship among the participants defined for the pier project. This organizational chart reflects a common practice in the construction industry in the United States today of the owner hiring a general contractor to manage all facets of the project construction.

### 6.4 How JANUS approaches the analysis

This section walks through the sample pier project and explains how JANUS performs its time-cost tradeoff analysis.

#### 6.4.1. Initialization

To begin with, all the various components are initialized. Participants and activities both require initialization.

*Participants/Organization*

Initialization of the project participants involves setting up all employer/employee relations among them. To begin with, for each participant, the user must define a set of employees and a contract type. That is, the user must fill in the *Employ* and *Contract* slots for each participant. JANUS then fills in each participant’s *Employed by* slot and its *Day.Value* slot, the former directly from other participants’ *Employ* slots, the latter based upon the value of its *Contract* slot. As currently implemented, JANUS assigns day value based upon this value. If a participant’s contract (with its employer) is a cost-plus contract,
its day value is that of its employer. If it is a lump sum contract, general contractors’ day values are set to their daily overhead rate and subcontractors’ day values are set to 0. The only demon associated with participant frames calculates its day value when its contract changes. The user must manually perform any other updates needed to keep the participants’ slots current.

Activities/Schedule Network

When activity frames are defined, the values of their MH. Estimate and Material. Estimate slots must be input by the user. Demons attached to those slots cause the other estimate slots to be calculated whenever changes are made to the manhour estimate or to the material estimate. The activity’s current cost, duration and cost per day; its normal cost, duration and cost per day; its next cost, duration and cost per day; and its cost slope are also calculated at that time.

Initialization of the schedule network first involves clearing the values of each activity’s CPM-related slots (early start, float, criticality, etc.) and its predecessors slot. Next, the first activity in the network is messaged to initialize the network. This network initialization involves notifying each of its successors to add it to their predecessors slot and then to continue initializing the network by so messaging their successors. The first activity in the network is then messaged to calculate the network, which causes all the CPM-related slots of each activity in the network to be calculated. After this initialization, a demon attached to the current duration slot of each activity causes the network to be recalculated whenever it is changed.

The network initialization also includes initialization of the project frame. That is, all activities in the project are collected and put into the Project. Activities slot and the project cost and duration are calculated. After this point, demons attached to various activity slots keep the project cost, duration, premium cost and time saved updated as individual activity slot values change.

6.4.2. Activity duration & cost

At this point, the project schedule is in its initial state, with each activity being performed in the most efficient way possible, i.e., by its preferred crew according to a straight-time work schedule. Assuming the project duration is unacceptable, the user may then instruct JANUS to expedite the project. The project is expedited by “crashing” individual activities. An activity is crashed by changing its current tactic to another that
shortens its duration. As currently implemented, JANUS can either crash just one activity or it can iteratively crash activities until no more activities can be crashed. JANUS stops when no critical activities can be further crashed without exceeding the day value of the activity’s responsible participant.

As an example, consider the activity Install.Girts. This activity is performed by the participant General.Contractor.1. That is, General.Contractor.1 is the value of the Responsible.Participant slot of the activity Install.Girts’. This activity has an initial estimate of 552 MH and $0 in material costs. Its preferred crew is a 3-man erection crew, whose average direct wage rate is $21.00, labor fringe rate is 50%, crew size is 3 men and equipment cost per crew hour is $25.00. Combining these factors gives the following values for Install.Girts’ slots:

- Labor.Estimate = $11,592
- Fringe.Estimate = $5,796
- CH.Estimate = 184
- Normal.Cost = $21,988
- Normal.Duration = 23

Install.Girts’ cost slope is also calculated. To begin with, its current tactic is Straight.Time.Only, as it is for every activity. Its Next.Tactic is 6.Eights, also as it is for every activity. Install.Girts’ cost slope is calculated from its next cost, its next duration (both based on its next tactic) and from its normal cost and normal duration. Install.Girts’ next cost is $24,259 and its next duration is 20 days, making its cost slope (757).

These calculations are done for every activity in the network. They are performed whenever an attribute that affects one of these factors changes, most notably the activity’s current tactic and its next tactic.

6.4.3. Reducing an activity’s duration

At each step of the analysis, JANUS selects one activity to crash. The activity it chooses is the critical activity with the smallest cost slope that does not exceed its responsible participant’s day value. Reducing that activity’s duration consists of:

- putting its next tactic into its current tactic slot
- updating its next tactic slot
- updating its current and next estimates and durations
- updating its delta cost, delta duration and cost slope
- recalculating the network and updating the project information.

Table 6 shows the cost and duration results of applying each of the tactics currently defined in the system to the activity Install.Girts.

<table>
<thead>
<tr>
<th>Expediting Tactic</th>
<th>Labor Estimate</th>
<th>Fringe Estimate</th>
<th>Equipment Estimate</th>
<th>Total Estimate</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight.Time.Only</td>
<td>11,592</td>
<td>5,796</td>
<td>4,600</td>
<td>21,988</td>
<td>23</td>
</tr>
<tr>
<td>6.Eights</td>
<td>13,218</td>
<td>5,796</td>
<td>5,245</td>
<td>24,259</td>
<td>20</td>
</tr>
<tr>
<td>5.Tens</td>
<td>13,422</td>
<td>5,796</td>
<td>5,326</td>
<td>24,544</td>
<td>19</td>
</tr>
<tr>
<td>6.Tens</td>
<td>14,878</td>
<td>5,796</td>
<td>5,904</td>
<td>26,578</td>
<td>17</td>
</tr>
<tr>
<td>5.Day.Swing</td>
<td>14,957</td>
<td>5,796</td>
<td>5,935</td>
<td>26,688</td>
<td>14</td>
</tr>
<tr>
<td>5.Day.Graveyard</td>
<td>17,664</td>
<td>5,796</td>
<td>7,009</td>
<td>30,469</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 6: Cost and duration impacts of the application of different tactics to the activity Install.Girts

JANUS' analysis terminates when there are no more activities to crash. The result of the analysis is an expedited schedule whose duration has been reduced within the cost parameters of the various participants defined for the project. Relations among the participants can be changed and JANUS re-run to reflect a different contracting strategy. The user can then use the overall cost and duration data to help evaluate the various contracting strategies available to him. This information has not previously been available to decision makers.

6.4.4. Results of two scenarios for the example pier project

Figure 10 shows the initial state of the pier project schedule. Table 4 shows the initial estimate for the pier project and Table 5 shows each activity’s initial duration and cost slope.

To demonstrate its analysis capabilities, JANUS was run twice: a cost-plus scenario, where all participants work under cost-plus contracts and the owner’s day value of $2,000; and a lump sum scenario, where all participants work under lump sum contracts and the general contractor’s daily overhead rate of $700. The strategy actually implemented in JANUS was to select expediting tactics in order of increasing duration reduction, not in order of increasing cost or efficiency. That strategy could be changed to consider tactics in order of increasing cost or some other factor as well.
In the cost-plus scenario, JANUS initially selected activity *Buy.PC.Piles* to crash. It selected this activity for 3 reasons:

1) it was critical
2) its cost slope (300) was less than its responsible participant’s (*Piling.Sub.1*)
   day value (2,000).

Crashing this activity consisted of changing its current tactic (*Straight.Time.Only*) to its next tactic (*Expedite.Purchase*). This change fired demons attached to the current tactic slot which recomputed its current duration (15 days) and current cost ($38,500). When JANUS changed the current duration, a demon attached to that slot caused the project network to be recomputed, reducing the overall project duration from 147 days to 143 and changing the critical path, making the activities *Mobilization, Demolition* and *Excavation* critical and making the activity *Buy.PC.Piles* no longer critical. Another demon recalculated the activity’s delta duration. The demons attached to the activity’s current cost recomputed the activity’s cost slope, its delta cost and the project’s cost.

The next activity JANUS selected to crash was *Install.Bumpers* because its cost slope (354) was then the lowest among the crashable activities. Its duration was progressively reduced from 15 days to 9 by successively applying tactics with increasing duration improvement characteristics. The next two tactics resulted in new cost slopes that were still less than the next cheapest alternative, so JANUS continued to crash this activity. After applying the 6*Tens* tactic, the activity *Install.Bumpers* became non-critical. JANUS then selected the activity *Remove.Girts*, which had just become critical, for expediting and began to crash it.

JANUS continued in this fashion until all crashable activities had been crashed to the limits allowed by the tactics defined. The result was an overall project duration of 79 days and an overall project cost of $321,074. In effect, $47,626 was paid to reduce the project duration 68 days. Table 5 shows the final durations of all activities in the network when the cost-plus scenario was finished. The durations shown in bold face are those for critical activities.

After running this cost-plus scenario, the participants’ contracts and day values were changed to the lump sum scenario and JANUS was run again. This time, JANUS first selected the activity *Install.Bumpers* because the piling subcontractor was no longer
willing to spend the $300 per day necessary to reduce the delivery time of the precast concrete piles. That is, its day value was 0, so its activities were not considered for crashing.

JANUS also stopped crashing activities earlier than under the cost-plus scenario. The last activity crashed in that situation was Set.PC.Caps, whose initial cost slope was (1,027). Under the lump sum scenario, JANUS would not consider that activity because its responsible participant, the general contractor, was unwilling to spend more than $700 a day. JANUS also did not crash a number of other activities because, like buying the precast concrete piles, their responsible participants’ day values were 0.

In this situation, the final state of the project was a duration of 130 days at a cost of $282,468. In other words, $9,020 was paid to reduce the project duration 17 days.
7.1. Contributions

The work described in this thesis and embodied in the prototype system JANUS makes several contributions to the state of the art in current project management practice.

- It provides a simulation of observed time-cost tradeoff behavior as opposed to the computation of a theoretically optimal time-cost tradeoff.
- It models organizational and contractual variables in a coherent representation and reasoning scheme to support the time-cost tradeoff analysis.
- It provides some insight into the effects of multiple participants on time-cost tradeoff analysis along with a concrete approach toward modeling and simulating those effects.

7.2. Suggested Extensions

The prototype system JANUS is really just a starting point for studying organizational strategies in the context of projects. It provides a coherent architecture for representing many of the factors needed to assess schedule impacts. It can, however, use some improvement in several areas.

The CPM calculations as currently implemented are not particularly efficient. The network that these calculations may run on is also restricted to a single start node and a single end node. We have not attempted to make the CPM calculations efficient, but the overall system would benefit from improvements in this area.

As currently implemented, crews are the smallest production unit defined in JANUS. A fascinating area for future work is in the development and definition of crew composition. A great deal of work could be done on the fundamentals of various resources, what it is about different resources that makes them work well with other resources and what combinations of resources – labor, tools and equipment – go best together and why. JANUS could be easily extended to include separate frame representations for resources and to reason about how those resources should be combined into crews, taking into consideration their cost, their degree of specialization, their availability, etc. Reasoning at this level would also allow analysis of resource-constrained schedules with consideration
of such things as the amount of resource or availability of a resource on the site or in the region.

The way estimating is implemented in JANUS is somewhat awkward. Construction estimators generally do not work from the level of crews when developing estimates. They usually work from a more detailed set of resources, i.e., individual craftsmen, separate pieces of equipment, etc. The estimating portion of the model would benefit from the addition of resources as separate objects. Cost data could then be associated with the resources where they belong rather than with the crew, which should really be just a summary of its members’ costs.

In its current state, JANUS does very little reasoning about contract clauses with respect to their effect on project participants’ behavior. This is another interesting area for future work. JANUS’ representation of contracts could be extended to include specific contract clauses, e.g., a liquidated damages clause, and its reasoning could then include consideration of the presence or absence of such clauses.

The expediting tactic representation incorporated in the current version of JANUS contains a rather restricted set of tactics. The abstraction hierarchy for tactics outlined in [Barber 88] provides a more general approach to defining tactics. JANUS’ tactic representation could be structured along similar lines to allow better reasoning about the feasibility of expediting activities and to allow evaluation of a more complete set of feasible tactics for each activity.

Among the tactics that are defined, no provision is made for decreasing tactic efficiency over time. Studies have shown [BRT 80] that efficiency decreases dramatically as overtime is worked beyond a month or so at a time. JANUS would certainly benefit from an extension to incorporate this idea.

All costs included in JANUS’ project cost total are direct production costs. JANUS’ project cost estimating functions currently do not include time-related costs such as jobsite overhead. JANUS does use participants’ time-related indirect costs to determine whether and how to expedite an activity. Depending on the type of contract, indirect costs may or may not affect contract and project costs. JANUS would benefit from an extension to its contract reasoning to account for these indirect costs in its project cost calculations.

JANUS’ approach to day value could also be extended. As currently implemented, no consideration is made of the number of activities or participants working at a given time.
Each participant’s day value is a static number that does not vary for the entire project. It would be useful to allow more dynamic reasoning about participant day value within the more detailed context of the project, e.g., considering the number of activities a participant has going on at a given time.

The purpose of this research project was not to implement a commercial project management system. Its purpose was to prove the concept of implementing the ideas developed during the course of the research in a computer program. While that goal has been achieved, a substantial amount of work would have to be done to make the system generally usable. As currently implemented, JANUS contains no good graphical representation of the project network; it only uses the slot graphing capability available in KEE. A good graphical network representation is a needed extension to the work.

Control of the analysis demonstrated in JANUS is currently handled by the user executing commands in the proper order. Editing of project components is done strictly through the KEE interface. Another extension needed to make the system more generally useful is to implement all commonly required functions and operations as menu commands.
References


A Knowledge-Based Model of Time-Cost Tradeoff Analysis for Construction Schedules

(Part II-Appendices)

Alan T. Axworthy

TECHNICAL REPORT
Number 21 B

January, 1990

Stanford University
Preface

This technical report supplements that entitled *A Knowledge-Based Model of Time-Cost Tradeoff Analysis for Construction Schedules*. It provides computer code and frame descriptions for the objects utilized in the model. The prototype model, JANUS, was implemented in IntelliCorp's knowledge engineering environment, *KEE*. Objects in the model were implemented as *KEE* units. Methods attached to those units were written in Common Lisp within *KEE*. 
Table of Contents

Preface .................................................................................................................................................. iii
Table of Contents ................................................................................................................................ iv
Appendix 1. *Frames*: The JANUS Knowledge Base ................................................................. 1
Appendix 2. *Code*: Lisp methods used in JANUS ................................................................. 33
Appendix 3. *Demons*: Active values used in JANUS ............................................................. 73
Appendix 1.  Frames: The JANUS Knowledge Base

Activities.................................................................................................................................2
Purchasing Activities.............................................................................................................14
Contracts ...............................................................................................................................20
Crews......................................................................................................................................21
Participants............................................................................................................................23
Projects ...................................................................................................................................26
Tactics .....................................................................................................................................29
Unit: ACTIVITIES

Unit Comment: The class of schedule activities.
Superclasses: ENTITIES
Member of: CLASSES
Subclasses: LABOR.DRIVEN.ACTIVITIES MISC.ACTIVITIES PURCHASING.ACTIVITIES EQUIPMENT.DRIVEN.ACTIVITIES DUMMY.ACTIVITIES

Member slot: ADD.PREDECESSOR
Comment: A method to add a predecessor to an activity.
Inheritance: METHOD
Valueclass: METHOD
Values: ADD-PREDECESSOR-FN

Member slot: ALTERNATE.CREW
Comment: The crew that can perform the activity in the fastest possible time, regardless of the cost.
Inheritance: OVERRIDE.VALUES
Valueclass: CREWS
Cardinality: Max 1
Values: 1.MAN.DUMMY.CREW

Member slot: BACKWARD.FLAG
Comment: The slot to indicate whether the activity's backward pass calculations have been correctly performed yet.
Inheritance: OVERRIDE.VALUES
Valueclass: (ONE.OF TRUE FALSE UNDETERMINED)
Cardinality: Max 1
Values: TRUE

Member slot: BACKWARD.PASS
Comment: The method controlling the updating of the activity's late finish and start dates and its float.
Inheritance: METHOD
Valueclass: METHOD
Values: BACKWARD-PASS-FN

Member slot: CALCULATE.CH.ESTIMATE
Comment: The method to calculate the activity's crew hour estimate from its manhour estimate and its current crew.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-CH-ESTIMATE-FN

Member slot: CALCULATE.COST
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-COST-FN
Member slot: CALCULATE.COST.SLOPE  
Comment: The method to calculate the activity's cost slope.  
Inheritance: METHOD  
Valueclass: METHOD  
Values: CALCULATE-COST-SLOPE-FN

Member slot: CALCULATE.CURRENT_COST.PER.DAY  
Comment: The method to calculate the activity's current cost per day.  
Inheritance: METHOD  
Valueclass: METHOD  
Values: CALCULATE-CURRENT-COST-PER-DAY-FN

Member slot: CALCULATE.DELTA.COST  
Comment: The method to calculate the difference between the activity's current cost and its normal cost.  
Inheritance: METHOD  
Valueclass: METHOD  
Values: CALCULATE-DELTA-COST-FN

Member slot: CALCULATE.DELTA.DURATION  
Comment: The method to calculate the difference between the activity's current duration and its normal duration.  
Inheritance: METHOD  
Valueclass: METHOD  
Values: CALCULATE-DELTA-DURATION-FN

Member slot: CALCULATE.DURATION  
Comment: The method that calculates an activity's duration from its MH.Estimate and the value of the Crew.Size slot of its Preferred.Crew.  
Inheritance: METHOD  
Valueclass: METHOD  
Values: CALCULATE-CURRENT-DURATION-FN

Member slot: CALCULATE.EF  
Comment: A method to calculate the value of the activity's early finish slot.  
Inheritance: METHOD  
Valueclass: METHOD  
Values: CALCULATE-EF-FN

Member slot: CALCULATE.EQUIPMENT  
Inheritance: METHOD  
Valueclass: METHOD  
Values: CALCULATE-EQUIPMENT-FN

Member slot: CALCULATE.ES  
Comment: A method to calculate the value of the activity's early start slot.  
Inheritance: METHOD  
Valueclass: METHOD  
Values: CALCULATE-ES-FN
Member slot: CALCULATE.ESTIMATE
Comment: A method to perform the calculations to determine the labor, fringes, equipment and total estimates for an activity.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-ESTIMATE-FN

Member slot: CALCULATE.FLOAT
Comment: A method to calculate the value of the activity's float slot.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-FLOAT-FN

Member slot: CALCULATE.FRINGES
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-FRINGES-FN

Member slot: CALCULATE.LABOR
Comment: The method to calculate the value of the activity's direct labor cost. It multiplies the value of its MH.Estimate slot by the value of the Direct.Cost.per.MH slot of its Preferred.Crew.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-LABOR-FN

Member slot: CALCULATE.LF
Comment: A method to calculate the value of the activity's late finish slot.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-LF-FN

Member slot: CALCULATE.LS
Comment: A method to calculate the value of the activity's late start slot.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-LS-FN

Member slot: CALCULATE.NETWORK
Comment: A method to control all the network calculations.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-NETWORK-FN

Member slot: CALCULATE.NEXT.COST
Comment: The method to calculate the activity's total estimated cost based upon its next tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-NEXT-COST-FN
Appendix 1: Activities

Member slot: CALCULATE.NEXT.COST.PER.DAY
Comment: The method to calculate the activity's cost per day based on its next tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-NEXT-COST-PER-DAY-FN

Member slot: CALCULATE.NEXT.DURATION
Comment: The method to calculate an activity's duration based on its next tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-NEXT-DURATION-FN

Member slot: CALCULATE.NEXT.EQUIPMENT
Comment: The method to calculate the activity's equipment estimate based upon the activity's next tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-NEXT-EQUIPMENT-FN

Member slot: CALCULATE.NEXT_ESTIMATE
Comment: The method to calculate the activity's total cost estimate based on its next tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-NEXT-ESTIMATE-FN

Member slot: CALCULATE.NEXT.LABOR
Comment: The method to calculate the activity's labor estimate based upon its next tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-NEXT-LABOR-FN

Member slot: CALCULATE.NORMAL.COST.PER.DAY
Comment: The method to calculate the activity's normal cost per day
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-NORMAL-COST-PER-DAY-FN

Member slot: CH.ESTIMATE
Comment: The current estimated number of crew hours required to perform the activity. This value comes from the activity's MH.Estimate divided by the value of the Crew.Size slot of its Preferred.Crew.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0
Member slot: CHANGE CURRENT TACTIC
Comment: The method to update the activity's current tactic and its next tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CHANGE CURRENT TACTIC FN

Member slot: CHECK PRED EFS
Comment: The method to check if predecessors have calculated their forward passes yet.
Inheritance: METHOD
Valueclass: METHOD
Values: CHECK PRED EFS FN

Member slot: CHECK SUCC LSS
Comment: The method to check that an activity's successors' Late Start dates have been correctly calculated.
Inheritance: METHOD
Valueclass: METHOD
Values: CHECK SUCC LSS FN

Member slot: CLEAR ESTIMATE
Comment: The method to clear all the estimate slots of the activity.
Inheritance: METHOD
Valueclass: METHOD
Values: CLEAR ESTIMATE FN

Member slot: CLEAR PRED EFS
Comment: A method to clear all the values of its Pred.EFs slot.
Inheritance: METHOD
Valueclass: METHOD
Values: CLEAR PRED EFS FN

Member slot: CLEAR PREDECESSORS
Comment: A method to clear all values from the activity's Predecessors slot.
Inheritance: METHOD
Valueclass: METHOD
Values: CLEAR PREDECESSORS FN

Member slot: CLEAR SUCC LSS
Comment: A method to clear the activity's successor early start dates slot, Succ.LSs.
Inheritance: METHOD
Valueclass: METHOD
Values: CLEAR SUCC LSS FN

Member slot: COLLECT PRED EFS
Comment: A method to collect the early finish dates of all an activity's predecessors.
Inheritance: METHOD
Valueclass: METHOD
Values: COLLECT PRED EFS FN
Member slot: COLLECT.SUCC.LSS
Comment: A method to collect the late start dates of all the activity's successors.
Inheritance: METHOD
Valueclass: METHOD
Values: COLLECT-SUCC-LSS-FN

Member slot: COST.SLOPE
Comment: The activity's cost slope.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
Values: 999999.9

Member slot: CRITICAL
Comment: A slot that is true if the activity is critical, i.e., if it has no float.
Inheritance: OVERRIDE.VALUES
Valueclass: (ONE.OF TRUE FALSE UNDETERMINED)
Cardinality: Max 1
Values: UNDETERMINED

Member slot: CURRENT.COST
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
ActiveValues: CALC.PROJECT EstIMATE.AV CALC.COST.SLOPE.AV
Values: 0

Member slot: CURRENT.COST.PER.DAY
Comment: The current activity cost per day
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
ActiveValues: CALC.COST.SLOPE.AV
Values: 0

Member slot: CURRENT.CREW
Comment: The crew currently assigned to the activity by the system. The value of this slot may change as a result of reasoning by the system. It must be either the activity's Preferred.Crew or its Alternate.Crew.
Inheritance: OVERRIDE.VALUES
Valueclass: CREWS
Cardinality: Max 1
ActiveValues: CALC.DURATION.AV CALC.ESTIMATE.AV
Values: 1.MAN.DUMMY.CREW
Member slot: CURRENT.DURATION
Comment: The current duration of the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
ActiveValues: CALC.NETWORK.AV CALC.COST.SLOPE.AV
Values: 0

Member slot: CURRENT.TACTIC
Comment: The value of the expediting tactic currently being applied to the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: TACTICS
Cardinality: Max 1
ActiveValues: CALC.ESTIMATE.AV
Values: STRAIGHT.TIME.ONLY

Member slot: DELTA.COST
Comment: The difference between the activity's current cost and its normal cost.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: DELTA.DURATION
Comment: The difference between the activity's current duration and its normal duration.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: DESCRIPTION
Comment: A more verbose description of the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: WORD
Cardinality: Max 1
Values: Unknown

Member slot: DETERMINE.CURRENT.CREW
Comment: The method that determines the value of the activity's Current.Crew slot.
Inheritance: METHOD
Valueclass: METHOD
Values: Unknown

Member slot: EARLY.FINISH
Comment: Traditional CPM early finish date.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0
Member slot: EARLY.START
Comment: Traditional CPM early start date.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: EQUIPMENT_ESTIMATE
Comment: The estimated equipment cost for the activity, in dollars.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: FLOAT
Comment: Traditional CPM total float.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: FORWARD.FLAG
Comment: True when the forward pass has been properly calculated, otherwise false.
Inheritance: OVERRIDE.VALUES
Valueclass: (ONE.OF TRUE FALSE UNDETERMINED)
Cardinality: Max 1
Values: TRUE

Member slot: FORWARD.PASS
Comment: The method controlling the updating of the activity's early start and finish dates.
Inheritance: METHOD
Valueclass: METHOD
Values: FORWARD-PASS-FN

Member slot: FRINGE_ESTIMATE
Comment: The estimated labor fringe cost for the activity, in dollars.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: INITIALIZE.ACTIVITY
Comment: A method to set the Early.Start, Last.EF, Late.Finish and Last.LS slots to 0.
Inheritance: METHOD
Valueclass: METHOD
Values: INITIALIZE-ACTIVITY-FN

Member slot: INITIALIZE.BACKWARD.FLAGS
Comment: The method to propagate the setting of the Backward.Flag slot of all activities to False.
Inheritance: METHOD
Valueclass: METHOD
Values: INITIALIZE-BACKWARD-FLAGS-FN
Member slot: INITIALIZE.FORWARD.FLAGS
Comment: The method to propagate the setting of the Forward.Flag slot of all affected activities to False.
Inheritance: METHOD
Valueclass: METHOD
Values: INITIALIZE-FORWARD-FLAGS-FN

Member slot: INITIALIZE.NETWORK
Comment: A method to clear the Predecessor slot of all activities in the network.
Inheritance: METHOD
Valueclass: METHOD
Values: INITIALIZE-NETWORK-FN

Member slot: INPUT.estimate
Inheritance: METHOD
Valueclass: METHOD
Values: INPUT-ESTIMATE-FN

Member slot: LABOR.estimate
Comment: The estimated direct labor cost for the activity, in dollars.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: LATE.FINISH
Comment: Traditional CPM late finish date.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: LATE.START
Comment: Traditional CPM late start date.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: MATERIAL.estimate
Comment: The estimated material cost for the activity, in dollars.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
ActiveValues: CALC.estimate.Av
Values: 0
Appendix 1: Activities

Member slot: MH.ESTIMATE
Comment: The estimated number of manhours required for the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
ActiveValues: CALC.DURATION.AV CALC.ESTIMATE.AV
Values: 0

Member slot: NEXT.COST
Comment: The activity's total estimated cost based on its next tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: NEXT.COST.PER.DAY
Comment: The activity's cost per day based on its next cost and next duration.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
Values: 0.0

Member slot: NEXT.DURATION
Comment: The activity's estimated duration based on its next tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: NEXT.EQUIPMENT.ESTIMATE
Comment: The activity's estimated equipment cost based on its next tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: NEXT.LABOR.ESTIMATE
Comment: The estimate of direct labor costs based upon the activity's next tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: NEXT.TACTIC
Comment: The next tactic to be employed in expediting the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: TACTICS
Cardinality: Max 1
ActiveValues: CALC.NEXT.ESTIMATE.AV
Values: 6.EIGHTS
Member slot: NORMAL.COST
Comment: The normal cost of the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: NORMAL.COST.PER.DAY
Comment: The value of the activity's normal cost divided by its normal duration.
Inheritance: OVERRIDE.VALUES
Valueclass: NUMBER
Cardinality: Max 1
Values: 0

Member slot: NORMAL.DURATION
Comment: The normal duration of the activity, i.e., the activity's duration using its preferred.crew.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: NOTIFY.SUCCESSORS
Comment: A method to notify all an activity's successors to add the activity to their Predecessors slot.
Inheritance: METHOD
Valueclass: METHOD
Values: NOTIFY-SUCCESSORS-FN

Member slot: PRED.EFS
Comment: A list of the early finish dates of all the activity's predecessors.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Values: 0

Member slot: PREDECESSORS
Comment: The list of all immediate predecessor activities of the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: ACTIVITIES
Values: Unknown

Member slot: PREFERRED.CREW
Comment: The preferred crew for the activity. This crew should be that which is able to perform the activity most efficiently, i.e., at least cost.
Inheritance: OVERRIDE.VALUES
Valueclass: CREWS
Cardinality: Max 1
Values: 1.MAN.DUMMY.CREW
Appendix 1: Activities

Member slot: RESET.ACTIVITY
Comment: The method to reset an activity's estimate and tactics to its original state.
Inheritance: METHOD
Valueclass: METHOD
Values: RESET-ACTIVITY-FN

Member slot: RESPONSIBLE.PARTICIPANT
Comment: The project participant responsible for performing this activity.
Inheritance: OVERRIDE.VALUES
Valueclass: PARTICIPANTS
Cardinality: Max 1
Values: GENERAL.CONTRACTOR.1

Member slot: SET.BACKWARD.FLAG
Comment: The method to set the value of an activity's Backward.Flag to False.
Inheritance: METHOD
Valueclass: METHOD
Values: SET-BACKWARD-FLAG-FN

Member slot: SET.FLAGS
Comment: The method to set all affected activities' Forward.Flag and Backward.Flag slots to False.
Inheritance: METHOD
Valueclass: METHOD
Values: SET-FLAGS-FN

Member slot: SET.FORWARD.FLAG
Comment: The method to set the Forward.Flag slot of an activity.
Inheritance: METHOD
Valueclass: METHOD
Values: SET-FORWARD-FLAG-FN

Member slot: SUCC.LSS
Comment: A list of the late start dates of all the activity's successors.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Values: 0

Member slot: SUCCESSORS
Comment: The list of all immediate successors of the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: ACTIVITIES
Values: Unknown
Unit: PURCHASING.ACTIVITIES

Superclasses: ACTIVITIES
Member of: CLASSES
Subclasses: COMMODITY.PURCHASING.ACTIVITIES
            CUSTOM.PURCHASING.ACTIVITIES

Member slot: BACKWARDFLAG
Comment: The slot to indicate whether the activity's backward pass
         calculations have been correctly performed yet.
Inheritance: OVERRIDE.VALUES
Valueclass: (ONE.OF TRUE FALSE UNDETERMINED)
Cardinality: Max 1
Values: TRUE

Member slot: CALCULATE.COST
Comment: The method to calculate the value of the activity's
         Total.Activity.Estimate slot. It sums the values of its
         and Equipment.Estimate slots.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-PURCHASING-COST-FN

Member slot: CALCULATE.DURATION
Comment: The method that calculates an activity's duration from
         its MH.Estimate and the value of the Crew.Size slot of
         its Preferred.Crew.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-PURCHASING-DURATION-FN

Member slot: CALCULATE.ESTIMATE
Comment: A method to perform the calculations to determine the
         labor, fringes, equipment and total estimates for an
         activity.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-PURCHASING-ESTIMATE-FN

Member slot: CALCULATE.MATERIAL
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-PURCHASING-MATERIAL-FN

Member slot: CALCULATE.NEXT.COST
Comment: The method to calculate the activity's total estimated
         cost based upon its next tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-NEXT-PURCHASING-COST-FN

Member slot: CALCULATE.NEXT.DURATION
Comment: The method to calculate an activity's duration based on
         its next tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-NEXT-PURCHASING-DURATION-FN
Member slot: CH.ESTIMATE
Comment: The current estimated number of crew hours required to perform the activity. This value comes from the activity's MH.Estimate divided by the value of the Crew.Size slot of its Preferred.Crew.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: CHANGE.CURRENT.TACTIC
Comment: The method to update the activity's current tactic and its next tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CHANGE-PURCHASING-TACTIC-FN

Member slot: CLEAR.ESTIMATE
Comment: The method to clear all the estimate slots of the activity.
Inheritance: METHOD
Valueclass: METHOD
Values: CLEAR-PURCHASING-ESTIMATE-FN

Member slot: COST.SLOPE
Comment: The activity's cost slope.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
Values: 999999.9

Member slot: CRITICAL
Comment: A slot that is true if the activity is critical, i.e., if it has no float.
Inheritance: OVERRIDE.VALUES
Valueclass: (ONE.OF TRUE FALSE UNDETERMINED)
Cardinality: Max 1
Values: TRUE

Member slot: CURRENT.COST
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
ActiveValues: CALC.PROJECT.ESTIMATE.AV CALC.COST.SLOPE.AV
Values: 0

Member slot: CURRENT.COST.PER.DAY
Comment: The current activity cost per day
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
ActiveValues: CALC.COST.SLOPE.AV
Values: 0
Member slot: CURRENT.CREW
Comment: The crew currently assigned to the activity by the system. The value of this slot may change as a result of reasoning by the system. It must be either the activity's Preferred.Crew or its Alternate.Crew.
Inheritance: OVERRIDE.VALUES
Valueclass: CREWS
Cardinality: Max 1
ActiveValues: CALC.DURATION.AV CALC.ESTIMATE.AV
Values: 1.MAN.DUMMY.CREW

Member slot: CURRENT.DURATION
Comment: The current duration of the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
ActiveValues: CALC.NETWORK.AV CALC.COST.SLOPE.AV
Values: 0

Member slot: CURRENT.TACTIC
Comment: The value of the expediting tactic currently being applied to the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: TACTICS
Cardinality: Max 1
ActiveValues: CALC.ESTIMATE.AV
Values: STRAIGHT.TIME.ONLY

Member slot: EARLY.FINISH
Comment: Traditional CPM early finish date.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: EARLY.START
Comment: Traditional CPM early start date.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: EQUIPMENT.ESTIMATE
Comment: The estimated equipment cost for the activity, in dollars.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: EXPEDITED.DURATION
Comment: The duration a purchasing activity may be expedited to by spending some money.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0
Member slot: FLOAT
Comment: Traditional CPM total float.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: FORWARD.FLAG
Comment: True when the forward pass has been properly calculated, otherwise false.
Inheritance: OVERRIDE.VALUES
Valueclass: (ONE.OF TRUE FALSE UNDETERMINED)
Cardinality: Max 1
Values: TRUE

Member slot: FRINGE.ESTIMATE
Comment: The estimated labor fringe cost for the activity, in dollars.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: INPUT.ESTIMATE
Inheritance: METHOD
Valueclass: METHOD
Values: INPUT-PURCHASING-ESTIMATE-FN

Member slot: LABOR.ESTIMATE
Comment: The estimated direct labor cost for the activity, in dollars.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: LATE.FINISH
Comment: Traditional CPM late finish date.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: MATERIAL.ESTIMATE
Comment: The estimated material cost for the activity, in dollars.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
ActiveValues: CALC. ESTIMATE.AV
Values: 0
Appendix 1: Purchasing Activities

Member slot: MH.ESTIMATE
Comment: The estimated number of manhours required for the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
ActiveValues: CALC.DURATION.AV CALC.ESTIMATE.AV
Values: 0

Member slot: NEXT.COST
Comment: The activity's total estimated cost based on its next tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: NEXT.COST.PER.DAY
Comment: The activity's cost per day based on its next cost and next duration.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
Values: 0.0

Member slot: NEXT.DURATION
Comment: The activity's estimated duration based on its next tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: NEXT.EQUIPMENT.ESTIMATE
Comment: The activity's estimated equipment cost based on its next tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: NEXT.LABOR.ESTIMATE
Comment: The estimate of direct labor costs based upon the activity's next tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: NEXT.MATERIAL.COST
Comment: The cost of expediting a purchasing activity to its expedited duration.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0
Appendix 1: Purchasing Activities

Member slot: NEXT.TACTIC
Comment: The next tactic to be employed in expediting the activity.
Inheritance: OVERRIDE.VALUES
Valueclass: TACTICS
Cardinality: Max 1
ActiveValues: CALC.NEXT_ESTIMATE_AV
Values: EXPEDITE_PURCHASE

Member slot: PRED.EFS
Comment: A list of the early finish dates of all the activity's predecessors.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Values: 0

Member slot: REGULAR.COST
Comment: The regular cost of the purchased item, corresponding to its normal delivery time.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: SUCC.LSS
Comment: A list of the late start dates of all the activity's successors.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Values: 0
Unit: CONTRACTS

Unit Comment: The class of contracts.
Member of: CLASSES
Subclasses: LUMP.SUM.CONTRACTS COST.PLUS.CONTRACTS
Unit: CREWS

Unit Comment: The class of all work crews available to perform activities.
Superclasses: ENTITIES
Member of: CLASSES
Subclasses: DUMMY.CREWS EARTHWORK.CREWS PILING.CREWS MECHANICAL.CREWS LABOR.CREWS CARPENTRY.CREWS CONCRETE.CREWS

Member slot: CALCULATE.COMPOSITE.CREW.RATE
Comment: The method to calculate the value of the slot Composite.Crew.Rate.
Inheritance: METHOD
Valueclass: METHOD
Values: Unknown

Member slot: COMPOSITE.CREW.RATE
Comment: The hourly rate for the entire crew, including direct labor, labor fringes and equipment.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
Values: 0.0

Member slot: CREW.MEMBERS
Comment: A list of the crafts of each of the crew members
Inheritance: OVERRIDE.VALUES
Valueclass: WORD
Values: Unknown

Member slot: CREW.SIZE
Comment: The number of individuals in the crew.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 1

Member slot: DIRECT.COST.PER.MH
Comment: The average hourly wage rate of the crew members.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
Values: 0.0

Member slot: EQUIPMENT
Comment: A list of all equipment utilized by the crew.
Inheritance: OVERRIDE.VALUES
Valueclass: WORD
Values: Unknown

Member slot: EQUIPMENT.COST.PER.CH
Comment: The hourly rate of the equipment utilized by the crew.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
Values: 0.0
Appendix 1: Crews

Member slot: LABOR.FRINGE.RATE
Comment: The estimated cost of fringe benefits, expressed as a percentage of direct labor costs.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
Values: 0.5
Unit: PARTICIPANTS

Unit Comment: The class of all project participants.
Superclasses: ENTITIES
Member of: CLASSES
Subclasses: SUPPLIERS OWNERS CONTRACTORS

Member slot: ANNUAL.SALES
Comment: The value of this slot is the participant's approximate annual sales volume in dollars. It is used to give us an idea of the size of the participant relative to the other participants.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: Unknown

Member slot: CLEAR.EMPLOYEES
Comment: This slot contains a method to message all its employees to clear their Employed.by slots.
Inheritance: METHOD
Valueclass: METHOD
Values: CLEAR-EMPLOYEES-FN

Member slot: CLEAR.EMPLOYER
Comment: The method contained in this slot clears the value from the Employed.by slot of the participant.
Inheritance: METHOD
Valueclass: METHOD
Values: CLEAR-EMPLOYER-FN

Member slot: CONTRACT
Comment: This slot points to the contract unit for the contract under which the participant works.
Inheritance: OVERRIDE.VALUES
Valueclass: CONTRACTS
Cardinality: Max 1
ActiveValues: CALC.DAY.VALUE.AV
Values: Unknown

Member slot: CONTRACT.AMOUNT
Comment: This slot is used in conjunction with the Annual.Sales slot to determine how important this project is to the participant.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: Unknown

Member slot: DAY.VALUE
Comment: This slot holds the participant's opinion of what one day is worth in dollars.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: Unknown
Member slot: DETERMINE.CONTRACT.TYPE
Comment: The method to set subcontractors' and suppliers' contracts to the same type as the general contractor's.
Inheritance: METHOD
Valueclass: METHOD
Values: DETERMINE-CONTRACT-TYPE-FN

Member slot: DETERMINE.DAY.VALUE
Comment: This method will determine the participant's perceived value of one day and put that value into its Day.Value slot.
Inheritance: METHOD
Valueclass: METHOD
Values: DETERMINE-DAY-VALUE-FN

Member slot: EMPLOYED.BY
Comment: This slot contains the participant for whom this participant works.
Inheritance: OVERRIDE.VALUES
Valueclass: (MEMBER.OF PARTICIPANTS)
Cardinality: Max 1
ActiveValues: CALC.DAY.VALUE.AV
Values: Unknown

Member slot: EMPLOYS
Comment: This slot contains the participant(s) this participant employs.
Inheritance: OVERRIDE.VALUES
Valueclass: (MEMBER.OF PARTICIPANTS)
ActiveValues: EMPLOYER.AV
Values: Unknown

Member slot: INITIALIZE.ORG.CHART
Comment: The method to initialize the organizational chart. It reconnects employees with their employers and updates their day values.
Inheritance: METHOD
Valueclass: METHOD
Values: INITIALIZE-ORG-CHART-FN

Member slot: INITIALIZE.PARTICIPANT
Comment: This method initializes the Contract.Importance, Day.Value and Employer.Relationship slots of the participant frame.
Inheritance: METHOD
Valueclass: METHOD
Values: INIT-PARTICIPANT-FN

Member slot: NOTIFY.EMPLOYEES
Comment: This slot contains a message to send a message to its employees, notifying them of its new status as their employer.
Inheritance: METHOD
Valueclass: METHOD
Values: NOTIFY-EMPLOYEES-FN
Member slot: SET.EMPLOYER
Comment: This slot contains a method to fill in its employed by
        slot when it receives this message from its employer. It
        requires 2 arguments: itself and the employer unit.
Inheritance: METHOD
Valueclass: METHOD
Values: SET-EMPLOYER-FN
Unit: PROJECTS

Superclasses: ENTITIES
Member of: CLASSES
Members: PIER.PROJECT

Member slot: ADDED.COST
Comment: The premium cost over and above the project's normal cost.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: CALCULATE.ADDED.COST
Comment: The method to calculate the difference between the project's current cost and its normal cost.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-ADDED-COST-FN

Member slot: CALCULATE.PROJECT.DURATION
Comment: The method that calculates the overall duration of the project, in days.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-PROJECT-DURATION-FN

Member slot: CALCULATE.PROJECT.ESTIMATE
Comment: The method that calculates the estimated labor, material and equipment costs and estimated manhours for the project.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-PROJECT-ESTIMATE-FN

Member slot: CALCULATE.REDUCED.DURATION
Comment: The method to calculate the difference between the project's current duration and its normal duration.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-REDUCED-DURATION-FN

Member slot: CRASH.ACTIVITY
Comment: The method to crash one single activity.
Inheritance: METHOD
Valueclass: METHOD
Values: CRASH-ACTIVITY-FN

Member slot: CRASH.PROJECT
Comment: The method to crash a project as far as the constraints on the activities allow.
Inheritance: METHOD
Valueclass: METHOD
Values: CRASH-PROJECT-FN
Member slot: DETERMINE.NEXT.TO.CRASH
Comment: The method to select the next activity in the project to crash.
Inheritance: METHOD
Valueclass: METHOD
Values: DETERMINE-NEXT-TO-CRASH-FN

Member slot: INITIALIZE.PROJECT
Comment: The method to initialize the project.
Inheritance: METHOD
Valueclass: METHOD
Values: INITIALIZE-PROJECT-FN

Member slot: NEXT.TO.CRASH
Comment: The next activity to crash in order to further reduce the project duration.
Inheritance: OVERRIDE.VALUES
Valueclass: ACTIVITIES
Cardinality: Max 1
Values: Unknown

Member slot: PROJECT.ACTIVITIES
Comment: The list of all activities contained in this project.
Inheritance: OVERRIDE.VALUES
Valueclass: (LIST.OF ACTIVITIES)
Values: Unknown

Member slot: PROJECT.DURATION
Comment: The total duration of the project.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: Unknown

Member slot: PROJECT.EQUIPMENT.TOTAL
Comment: The total estimated cost of equipment for the project.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: Unknown

Member slot: PROJECT.FRINGE.TOTAL
Comment: The total estimated cost of labor fringes for the project.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: Unknown

Member slot: PROJECT.LABOR.TOTAL
Comment: The total estimated cost of labor for the project.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: Unknown
Member slot: PROJECT.MATERIAL.TOTAL
Comment: The total estimated cost of material for the project.
Inheritance: OVERRIDE.VALUE
Valueclass: INTEGER
Cardinality: Max 1
Values: Unknown

Member slot: PROJECT.MH.TOTAL
Comment: The total number of estimated manhours for the project.
Inheritance: OVERRIDE.VALUE
Valueclass: INTEGER
Cardinality: Max 1
Values: Unknown

Member slot: PROJECT.NORMAL.COST
Comment: The sum of the project's activities' normal costs.
Inheritance: OVERRIDE.VALUE
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: PROJECT.NORMAL.DURATION
Comment: The duration of the project with all activities in their normal states.
Inheritance: OVERRIDE.VALUE
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: PROJECT.TOTAL.ESTIMATE
Comment: The total estimated cost of labor, material and equipment for the project.
Inheritance: OVERRIDE.VALUE
Valueclass: INTEGER
Cardinality: Max 1
Values: Unknown

Member slot: REDUCED.DURATION
Comment: The amount of time that the duration of the project has been reduced below its normal duration.
Inheritance: OVERRIDE.VALUE
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: UPDATE PROJECT
Comment: The method to update a project's cost, duration, added cost and reduced duration.
Inheritance: METHOD
Valueclass: METHOD
Values: UPDATE-PROJECT-FN
Unit: TACTICS

Unit Comment: The class of activity expediting tactics.
Superclasses: ENTITIES
Member of: CLASSES
Members: EXPEDITE_PURCHASE 6.EIGHTS 5.DAY.GRAVEYARD 5.TENS 5.DAY.SWING STRAIGHT.TIME.ONLY

Member slot: CALCULATE.COST.PREMIUM
Comment: The method to calculate the direct labor cost premium of this tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-COST-PREMIUM-FN

Member slot: CALCULATE.COST.SLOPE.MULTIPLIER
Comment: The method to calculate the direct labor cost slope multiplier for the tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-COST-SLOPE-MULTIPLIER-FN

Member slot: CALCULATE.DOUBLE.TIME
Comment: The method to calculate the number of double time hours worked by the tactic in a day.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-DOUBLE-TIME-FN

Member slot: CALCULATE.DURATION.IMPROVEMENT
Comment: The method to calculate the duration improvement achieved with this tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-DURATION-IMPROVEMENT-FN

Member slot: CALCULATE.EFFECTIVE.HOURS
Comment: The method to calculate the effective number of crew hours accomplished each day using the tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-EFFECTIVE-HOURS-FN

Member slot: CALCULATE.HOURS.PAID
Comment: The method to calculate the equivalent number of straight time hours the tactic costs for a day.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-HOURS-PAID-FN

Member slot: CALCULATE.HOURS.WORKED
Comment: The method to calculate the actual number of hours worked by the tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-HOURS-WORKED-FN
Appendix 1: Tactics

Member slot: CALCULATE.OVERTIME.PREMIUM
Comment: The method to calculate the equivalent number of straight-time hours for the overtime hours.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-OVERTIME-PREMIUM-FN

Member slot: CALCULATE.SHIFT.FACTORS
Comment: The method to calculate straight time hours and shift premium for the tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-SHIFT-FACTORS-FN

Member slot: CALCULATE.TIME.AND.A.HALF
Comment: The method to calculate the number of time-and-a-half hours the tactic works in a day.
Inheritance: METHOD
Valueclass: METHOD
Values: CALCULATE-TIME-AND-A-HALF-FN

Member slot: COST.PREMIUM
Comment: The additional direct labor cost for using this tactic over a standard 5-day, 8-hour work week. This factor is expressed as a percentage and includes the effect of overtime premiums and inefficiency of the tactic. This factor considers only the direct labor portion of the activity cost.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
ActiveValues: TACTIC.COST.SLOPE.MULTIPLIER.AV
Values: 0.0

Member slot: COST.SLOPE.MULTIPLIER
Comment: The value of the tactic's cost per day multiplier, i.e., the number by which to multiply an activity's daily cost to determine its cost slope using the tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
Values: 0.0

Member slot: DAYS.PER.WEEK
Comment: The number of days worked in one week for this tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER {[Interval: [5 7]]}
Cardinality: Max 1
ActiveValues: TACTIC.PREMIUM.HOURS.AV
Values: 5
Appendix 1: Tactics

Member slot: DURATION.IMPROVEMENT
Comment: The reduction of activity duration achieved as a result of using this tactic, expressed as a percentage of the standard 5-day, 8-hour work week.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
ActiveValues: TACTIC.COST.SLOPE.MULTIPLIER.AV
Values: -1.0

Member slot: EFFICIENCY
Comment: The efficiency of the tactic for performing an activity.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
ActiveValues: TACTIC.EFFECTIVE.HOURS.AV
Values: 1.0

Member slot: HOURS.1.5
Comment: The number of hours worked at time-and-a-half in one day for this tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
ActiveValues: TACTIC.HOURS.WORKED.AV TACTIC.OT.PREMIUM.AV
Values: 0

Member slot: HOURS.2
Comment: The number of hours worked at double-time in one day for this tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER
Cardinality: Max 1
Values: 0

Member slot: HOURS.EFFECTIVE
Comment: The effective number of crew hours worked in a day using this tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
ActiveValues: TACTIC.COST.PREMIUM.AV TACTIC.DURATION.IMPROVEMENT.AV
Values: 8.0

Member slot: HOURS.OT.PREMIUM
Comment: The equivalent number of straight-time hours paid for overtime work in one day.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
ActiveValues: TACTIC.HOURS.PAID.AV
Values: 0.0
Member slot: HOURS.PAIRED
Comment: The equivalent number of straight time hours that the tactic costs for one day.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
ActiveValues: TACTIC.COST.PREMIUM.AV
Values: 8.0

Member slot: HOURS.PER.SHIFT
Comment: The number of hours worked in each shift for this tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER #[Interval: [8 12]]
Cardinality: Max 1
ActiveValues: TACTIC.PREMIUM.HOURS.AV
Values: 8

Member slot: HOURS.SHIFT.PREMIUM
Comment: The total number of hours paid for in one day using this tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
Values: 0.0

Member slot: HOURS.STRAIGHT
Comment: The number of straight time hours worked in one day using this tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
ActiveValues: TACTIC.HOURS.PAIRED.AV TACTIC.HOURS.WORKED.AV
Values: 8.0

Member slot: HOURS.WORKED
Comment: The equivalent number of straight-time hours lost to inefficiency in one day.
Inheritance: OVERRIDE.VALUES
Valueclass: FLOAT
Cardinality: Max 1
ActiveValues: TACTIC.EFFECTIVE.HOURS.AV
Values: 8.0

Member slot: INITIALIZE.TACTIC.FACTORS
Comment: The method to calculate all the factors relating to the tactic.
Inheritance: METHOD
Valueclass: METHOD
Values: INITIALIZE-TACTIC-FACTORS-FN

Member slot: SHIFTS.PER.DAY
Comment: The number of shifts worked each day for this tactic.
Inheritance: OVERRIDE.VALUES
Valueclass: INTEGER #[Interval: [1 3]]
Cardinality: Max 1
ActiveValues: TACTIC.SHIFT.FACTORS.AV
Values: 1
Appendix 2. Code: Lisp methods used in JANUS

JANUSACT.LISP ................................................................. 34
JANUSA VS.LISP ............................................................... 43
JANUSCPM.LISP ............................................................... 49
JANUSCRW.LISP ............................................................... 57
JANUSPAR.LISP ............................................................... 58
JANUSPRO.LISP ............................................................... 62
JANUSTAC.LISP ............................................................... 67
;; -*- Syntax: Common-lisp; Package: K; Mode: LISP -*-
(in-package 'kee)

;;;; This file contains functions pertaining to calculation of
;;;; activity factors in JANUS. It operates in conjunction
;;;; with the knowledge base 'janus.u' and with the function
;;;; files 'janusfnv.lisp' and 'januscpm.lisp.'

;;;; CALCULATE-CH-ESTIMATE-FN is the value of the Calculate.CH.
;;;; Estimate slot of an activity. It divides the activity's

(defun CALCULATE-CH-ESTIMATE-FN (thisunit)
  (let* ((crew (unit (get.value thisunit 'Current.Crew)))
         (crewSize (get.value crew 'Crew.Size))
         (mh (get.value thisunit 'MH.Estimate))
         (put.value thisunit 'CH.Estimate
                   (/ mh crewSize))))

;;;; CALCULATE-LABOR-FN is the value of the Calculate.Labor
;;;; slot for an activity. It multiplies the value of the
;;;; activity's MH.Estimate slot by the value of the
;;;; Direct.Cost.per.MH slot of its Current.Crew, all multiplied
;;;; by 1 plus its Current.Tactic's Cost.Premium.

(defun CALCULATE-LABOR-FN (thisunit)
  (let* ((crew (unit (get.value thisunit 'Current.Crew)))
         (tactic (unit (get.value thisunit 'Current.Tactic)))
         (laborRate (get.value crew 'Direct.Cost.per.MH))
         (costPremium (get.value tactic 'Cost.Premium))
         (MH (get.value thisunit 'MH.Estimate))
         (put.value thisunit 'Labor.Estimate
               (* MH laborRate (1+ costPremium)))))

;;;; CALCULATE-NEXT-LABOR-FN does the same thing using the
;;;; activity's Next.Tactic.

(defun CALCULATE-NEXT-LABOR-FN (thisunit)
  (let* ((crew (unit (get.value thisunit 'Current.Crew)))
         (tactic (unit (get.value thisunit 'Next.Tactic)))
         (laborRate (get.value crew 'Direct.Cost.per.MH))
         (costPremium (get.value tactic 'Cost.Premium))
         (MH (get.value thisunit 'MH.Estimate))
         (put.value thisunit 'Next.Labor.Estimate
               (* MH laborRate (1+ costPremium)))))
;;; CALCULATE-FRINGES-FN is the value of the Calculate.Fringes
;;; slot for an activity. It multiplies the value of its
;;; Labor.Estimate slot by the value of the Labor.Fringe.Rate
;;; slot of its Current.Crew.

(defun CALCULATE-FRINGES-FN (thisunit)
  (let* ((crew (unit (get.value thisunit 'Current.Crew)))
         (fringeRate (get.value crew 'Labor.Fringe.Rate))
         (directLabor (* (get.value thisunit 'MH.Estimate)
                         (get.value crew 'Direct.Cost.per.MH))))
    (put.value thisunit 'Fringe.Estimate
                (* fringeRate directLabor))))

;;; CALCULATE-EQUIPMENT-FN is the value of the Calculate.Equipment
;;; slot for an activity. It determines the number of crew hours
;;; needed for the activity from the values of its MH.Estimate
;;; slot and the Crew.Size slot of its Current.Crew. It then
;;; figures the equipment cost from the number of crew hours and
;;; the value of the Equipment.Cost.per.CH slot of its
;;; Current.Crew.

(defun CALCULATE-EQUIPMENT-FN (thisunit)
  (let* ((crew (unit (get.value thisunit 'Current.Crew)))
         (tactic (unit (get.value thisunit 'Current.Tactic)))
         (costPremium (get.value tactic 'Cost.Premium))
         (equipRate (get.value crew 'Equipment.Cost.per.CH))
         (CH (get.value thisunit 'CH.Estimate)))
    (put.value thisunit 'Equipment.Estimate
                (*
                 CH
                 equipRate
                 (1+ costPremium)))))

;;; CALCULATE-NEXT-EQUIPMENT-FN is the same thing, using the
;;; activity's Next.Tactic.

(defun CALCULATE-NEXT-EQUIPMENT-FN (thisunit)
  (let* ((crew (unit (get.value thisunit 'Current.Crew)))
         (tactic (unit (get.value thisunit 'Next.Tactic)))
         (costPremium (get.value tactic 'Cost.Premium))
         (equipRate (get.value crew 'Equipment.Cost.per.CH))
         (CH (get.value thisunit 'CH.Estimate)))
    (put.value thisunit 'Next.Equipment.Estimate
                (*
                 CH
                 equipRate
                 (1+ costPremium)))))
;;; CALCULATE-PURCHASING-MATERIAL-FN is the value of the Calculate.
;;; Material slot of purchasing activities.
(defun CALCULATE-PURCHASING-MATERIAL-FN (thisunit)
  (let ((currentTactic (get.value thisunit 'Current.Tactic)))
    (cond ((eql currentTactic unit 'Expedite.Purchase)
           (put.value thisunit 'Material.Estimate
                        (get.value thisunit 'Next.Material.Cost)))
      (t (put.value thisunit 'Material.Estimate
                        (get.value thisunit 'Regular.Cost))))))

;;; CALCULATE-COST-FN is the value of the Calculate.Cost slot
;;; for an activity. It sums the values of its Labor.Estimate,
;;; slots.
(defun CALCULATE-COST-FN (thisunit av.off?)
  (let ((labor (get.value thisunit 'Labor.Estimate))
        (fringes (get.value thisunit 'Fringe.Estimate))
        (material (get.value thisunit 'Material.Estimate))
        (equipment (get.value thisunit 'Equipment.Estimate))
        (put.value thisunit 'Current.Cost
                   (+ labor fringes material equipment)
                   nil nil :no.av av.off?)))

;;; CALCULATE-NEXT-COST-FN is the same thing, using the activity's
;;; estimate information calculated from its Next.Tactic.
(defun CALCULATE-NEXT-COST-FN (thisunit)
  (let ((labor (get.value thisunit 'Next.Labor.Estimate))
        (fringes (get.value thisunit 'Fringe.Estimate))
        (material (get.value thisunit 'Material.Estimate))
        (equipment (get.value thisunit 'Equipment.Estimate))
        (put.value thisunit 'Next.Cost
                   (+ labor fringes material equipment))))

;;; CALCULATE-PURCHASING-COST-FN is the value of the Calculate.Cost
;;; slot of a purchasing activity.
(defun CALCULATE-PURCHASING-COST-FN (thisunit av.off?)
  (put.value thisunit 'Current.Cost
             (get.value thisunit 'Material.Estimate)
             nil nil :no.av av.off?))

;;; CALCULATE-NEXT-PURCHASING-COST-FN is the value of the Calculate.
;;; Next.Cost slot of a purchasing activity.
(defun CALCULATE-NEXT-PURCHASING-COST-FN (thisunit)
  (let ((expeditedCost (get.value thisunit 'Next.Material.Cost)))
    (put.value thisunit 'Next.Cost expeditedCost)))
;;; CLEAR-ESTIMATE-FN is the value of the Clear.Estimate slot
;;; for an activity. It clears all the calculated estimate values
;;; of the activity, i.e., it sets them to 0.

(defun CLEAR-ESTIMATE-FN (thisunit)
  (remove.all.values thisunit 'Labor.Estimate)
  (remove.all.values thisunit 'Fringe.Estimate)
  (remove.all.values thisunit 'Equipment.Estimate)
  (remove.all.values thisunit 'Current.Cost.per.Day)
  (remove.all.values thisunit 'Current.Tactic)
  (remove.all.values thisunit 'Next.Labor.Estimate)
  (remove.all.values thisunit 'Next.Equipment.Estimate)
  (remove.all.values thisunit 'Next.Cost.per.Day)
  (remove.all.values thisunit 'Next.Tactic)
  (remove.all.values thisunit 'Cost.Slope)
)

;;; CLEAR-PURCHASING-ESTIMATE-FN is the value of the Clear.Estimate
;;; slot of purchasing activities.

(defun CLEAR-PURCHASING-ESTIMATE-FN (thisunit)
  (remove.all.values thisunit 'Current.Cost.per.Day)
  (remove.all.values thisunit 'Current.Tactic)
  (remove.all.values thisunit 'Next.Cost.per.Day)
  (remove.all.values thisunit 'Next.Tactic)
  (remove.all.values thisunit 'Cost.Slope)
)

;;; CALCULATE-ESTIMATE-FN is the value of the Calculate.Estimate
;;; slot for an activity. It calculates labor, fringes, equipment
;;; and the total for an activity.

(defun CALCULATE-ESTIMATE-FN (thisunit av.off?)
  (cond ((eql (get.value thisunit 'Current.Duration)
  (get.value thisunit 'Next.Duration))
  nil)
  t
  (unitmsg thisunit 'Calculate.Labor)
  (unitmsg thisunit 'Calculate.Fringes)
  (unitmsg thisunit 'Calculate.Equipment)
  (unitmsg thisunit 'Calculate.Cost av.off?)
  (unitmsg thisunit 'Calculate.Duration av.off?)
  (unitmsg thisunit 'Calculate.Current.Cost.per.Day))
))

;;; CALCULATE-NEXT-ESTIMATE-FN does the same thing for the estimate
;;; calculated from the activity's Next.Tactic.

(defun CALCULATE-NEXT-ESTIMATE-FN (thisunit)
  (unitmsg thisunit 'Calculate.Next.Labor)
  (unitmsg thisunit 'Calculate.Fringes)
  (unitmsg thisunit 'Calculate.Next.Equipment)
  (unitmsg thisunit 'Calculate.Next.Cost)
  (unitmsg thisunit 'Calculate.Next.Duration)
  (unitmsg thisunit 'Calculate.Next.Cost.per.Day)
  (unitmsg thisunit 'Calculate.Cost.Slope))
)
;;; CALCULATE-PURCHASING-ESTIMATE-FN is the value of the Calculate.
;;; Estimate slot of purchasing activities.

(defun CALCULATE-PURCHASING-ESTIMATE-FN (thisunit av.off?)
  (unitmsg thisunit 'Calculate.Material)
  (unitmsg thisunit 'Calculate.Cost av.off?)
  (unitmsg thisunit 'Calculate.Duration av.off?)
  (unitmsg thisunit 'Calculate.Current.Cost.per.Day))

;;; INPUT-ESTIMATE-FN is the value of the Input.Estimate slot
;;; for an activity. It prompts the user for values for an
;;; activity's MH.Estimate and Material.Estimate slots. It
;;; then initializes the activity's normal, current and next
;;; costs, durations and costs per day along with its cost
;;; slope and delta cost and duration slots.

(defun INPUT-ESTIMATE-FN (thisunit)
  (unitmsg thisunit 'Clear.Estimate)
  (format t
    "-% Please type the manhour estimate for ~a.-%" thisunit)
  (let ((MH (read)))
    (format t
      "-% Please type its estimated material cost.-%")
    (let ((material (read)))
      (put.value thisunit 'MH.Estimate MH)
      (put.value thisunit 'Material.Estimate material)
      (unitmsg thisunit 'Reset.Activity)))))

;;; INPUT-PURCHASING-ESTIMATE-FN is the value of the Input.
;;; Estimate slot of a purchasing activity.

(defun INPUT-PURCHASING-ESTIMATE-FN (thisunit)
  (unitmsg thisunit 'Clear.Estimate)
  (format t
    "-% Please type the normal material cost of ~a.-%" thisunit)
  (let ((material (read)))
    (format t
      "-% What is its normal delivery time?-%")
    (let ((duration (read)))
      (format t
        "-% What is its expedited cost?-%")
      (let ((expeditedCost (read)))
        (format t
          "-% What is its expedited delivery time?-%")
        (let ((expeditedDuration (read)))
          (put.value thisunit 'Regular.Cost material)
          (put.value thisunit 'Normal.Duration duration)
          (put.value thisunit 'Next.Material.Cost expeditedCost)
          (put.value thisunit 'Expedited.Duration expeditedDuration)))
        (unitmsg thisunit 'Reset.Activity))
;;; CALCULATE-CURRENT-DURATION-FN is the value of the Calculate.
;;; Duration slot for an activity. It sets the value of its
;;; Current.Duration to the result of dividing its CH.Estimate
;;; by the value of its Current.Tactic's Hours.Effective slot.

(defun CALCULATE-CURRENT-DURATION-FN (thisunit av.off?)
  (unitmsg thisunit 'Calculate.CH.Estimate)
  (let* ((CH (get.value thisunit 'CH.Estimate))
         (tactic (get.value thisunit 'Current.Tactic))
         (effectiveHours (get.value tactic 'Hours.Effective))
         (duration (/ CH effectiveHours)))
  (cond (> duration 0)
    (put.value thisunit 'Current.Duration
duration nil nil :no.av av.off?))
  (t))

;;; CALCULATE-NEXT-DURATION-FN is the same thing, using the activity's
;;; Next.Tactic.

(defun CALCULATE-NEXT-DURATION-FN (thisunit)
  (unitmsg thisunit 'Calculate.CH.Estimate)
  (let* ((CH (get.value thisunit 'CH.Estimate))
         (tactic (get.value thisunit 'Next.Tactic))
         (effectiveHours (get.value tactic 'Hours.Effective)))
  (put.value thisunit 'Next.Duration
   (/ CH effectiveHours)))

;;; CALCULATE-PURCHASING-DURATION-FN is the value of the Calculate.
;;; Duration slot of purchasing activities. It sets the activity's
;;; Current.Duration to either its normal duration or to its
;;; expedited duration, based upon whether or not its Expedite.This.
;;; Activity slot is set.

(defun CALCULATE-PURCHASING-DURATION-FN (thisunit av.off?)
  (let* ((currentTactic (get.value thisunit 'Current.Tactic))
         (cond (eql currentTactic (unit 'Expedit.Purchase))
       (put.value thisunit 'Current.Duration
get.value thisunit 'Next.Duration
nil nil :no.av av.off?))
      (t (put.value thisunit 'Current.Duration
get.value thisunit 'Normal.Duration
nil nil :no.av av.off?))))

;;; CALCULATE-NEXT-PURCHASING-DURATION-FN is the value of the
;;; Calculate.Next. Duration slot of purchasing activities. It sets
;;; the activity's next duration to its expedited duration.

(defun CALCULATE-NEXT-PURCHASING-DURATION-FN (thisunit)
  (put.value thisunit 'Next.Duration
   (get.value thisunit 'Expedit.Duration)))
;;; CALCULATE-DURATION-FN is the value of the Calculate.Duration
;;; slot for an activity. It sets the value of its Duration
;;; slot to the result of dividing its MH.Estimate by the value of
;;; the Crew.Size slot of its Preferred.Crew and then by 8 (hours
;;; per day).

;;; This function was retired on 7/30/89 when I got the tactic
;;; and crew hierarchies defined to support the more proper
;;; calculation.
;;; -- See CALCULATE-CURRENT-DURATION-FN above --

(defun CALCULATE-DURATION-FN (thisunit)
  (let* ((crew (get.value thisunit 'Preferred.Crew))
         (crewSize (get.value crew 'Crew.Size))
         (MH (get.value thisunit 'MH.Estimate))
         (cond ((member (unit '(Dummy.Activities JANUS))
                (unit.ancestors thisunit 'member))
                (put.value thisunit 'Normal.Duration 0))
               (t (put.value thisunit 'Normal.Duration
                          (/* MH crewSize 8)))))

;;; CALCULATE-NORMAL-COST-PER-DAY-FN is the value of the Calculate.
;;; Normal.Cost.per.Day slot of an Activity. It sets the value
;;; of the activity's Normal.Cost.per.Day slot to the result of

(defun CALCULATE-NORMAL-COST-PER-DAY-FN (thisunit)
  (let ((cost (get.value thisunit 'Normal.Cost))
        (duration (get.value thisunit 'Normal.Duration)))
    (put.value thisunit 'Normal.Cost.per.Day
       (/* cost duration))))

;;; CALCULATE-CURRENT-COST-PER-DAY-FN is the value of the Calculate.
;;; Current.Cost.per.Day slot of an Activity. It sets the value of
;;; the activity's Current.Cost.per.Day slot to the result of
;;; dividing its Current.Cost by the value of its Current.Duration.

(defun CALCULATE-CURRENT-COST-PER-DAY-FN (thisunit)
  (let ((cost (get.value thisunit 'Current.Cost))
        (duration (get.value thisunit 'Current.Duration)))
    (cond ((eq 0 duration) (put.value thisunit 'Current.Cost.per.Day cost))
          (t (put.value thisunit 'Current.Cost.per.Day
            (/* cost duration))))))

;;; CALCULATE-NEXT-COST-PER-DAY-FN is the same thing, using the
;;; activity's Next.Tactic.

(defun CALCULATE-NEXT-COST-PER-DAY-FN (thisunit)
  (let ((cost (get.value thisunit 'Next.Cost))
        (duration (get.value thisunit 'Next.Duration)))
    (cond ((eq 0 duration) (put.value thisunit 'Next.Cost.per.Day cost))
          (t (put.value thisunit 'Next.Cost.per.Day
            (/* cost duration))))))
;;; CALCULATE-COST-SLOPE-FN is the value of the Calculate.Cost.Slope slot of an Activity. It sets the value of the activity's Cost.Slope to the result of dividing the difference between its next and normal costs by the difference between its next and normal durations. Note that this calculation is also hard-wired to the most expensive expediting tactic currently identified - 5.Day.Greyward.

(defun CALCULATE-COST-SLOPE-FN (thisunit)
  (let ((nextCost (get.value thisunit 'Next.Cost))
        (normalCost (get.value thisunit 'Normal.Cost))
        (nextDuration (get.value thisunit 'Next.Duration))
        (normalDuration (get.value thisunit 'Normal.Duration))
        (currentTactic (get.value thisunit 'Current.Tactic)))
    (cond ((or
            (eql nextDuration normalDuration)
            (eql currentTactic (unit '5.Day.Greyward)))
            (put.value thisunit 'Cost.Slope 999999.9))
          (t (put.value thisunit 'Cost.Slope
               (/ (- nextCost normalCost)
                  (- nextDuration normalDuration))))))

;;; CALCULATE-DELTA-COST-FN is the value of the Calculate.Delta.Cost slot of an activity. It sets the value of its Delta.Cost slot to the difference between its current cost and its normal cost.

(defun CALCULATE-DELTA-COST-FN (thisunit)
  (let ((currentCost (get.value thisunit 'Current.Cost))
        (normalCost (get.value thisunit 'Normal.Cost))
        (put.value thisunit 'Delta.Cost
               (- currentCost normalCost))))

;;; CALCULATE-DELTA-DURATION-FN is the value of the Calculate.Delta.Duration slot of an activity. It sets the value of its Delta.Duration slot to the difference between its current duration and its normal duration.

(defun CALCULATE-DELTA-DURATION-FN (thisunit)
  (let ((currentDuration (get.value thisunit 'Current.Duration))
        (normalDuration (get.value thisunit 'Normal.Duration))
        (put.value thisunit 'Delta.Duration
               (- currentDuration normalDuration))))
;;; CHANGE-CURRENT-TACTIC-FN is the value of the Change.Current.
;;; Tactic slot of an activity. It moves the activity’s next tactic to
;;; its current tactic slot and puts the next most duration-saving
;;; tactic into its next tactic slot.

(defun CHANGE-CURRENT-TACTIC-FN (thisunit)
  (let (((currentTactic (get.value thisunit 'Current.Tactic)))
    (cond
      ((eq currentTactic (unit 'Straight.Time.Only))
       (put.value thisunit 'Next.Tactic '5.Tens)
       (put.value thisunit 'Current.Tactic '6.Eights))
      ((eq currentTactic (unit '6.Eights))
       (put.value thisunit 'Next.Tactic '5.Day.Swing)
       (put.value thisunit 'Current.Tactic '5.Tens))
      ((eq currentTactic (unit '5.Tens))
       (put.value thisunit 'Next.Tactic '5.Day.Swing)
       (put.value thisunit 'Current.Tactic '6.Eights))
      ((eq currentTactic (unit '5.Eight))
       (put.value thisunit 'Next.Tactic '5.Day.Graveyard)
       (put.value thisunit 'Current.Tactic '5.Day.Swing))
      (t (put.value thisunit 'Current.Tactic '5.Day.Graveyard)))))

;;; CHANGE-PURCHASING-TACTIC-FN is the value of the Change.Current.
;;; Tactic slot of an activity.

(defun CHANGE-PURCHASING-TACTIC-FN (thisunit)
  (put.value thisunit 'Current.Tactic 'Expedite.Purchase)
  (put.value thisunit 'Cost.Slope 999999.9))

;;; RESET-ACTIVITY-FN is the value of the Reset.Activity slot of
;;; an activity. It resets the activity's estimate and tactic
;;; parameters to where they need to be to begin the analysis.

(defun RESET-ACTIVITY-FN (thisunit)
  (unimsg thisunit 'Clear.Estimate)
  (unimsg thisunit 'Calculate.Estimate t)
  (let (((currentCost (get.value thisunit 'Current.Cost))
          (currentDuration (get.value thisunit 'Current.Duration))
          (currentCostperDay (get.value thisunit 'Current.Cost.per.Day))
          (put.value thisunit 'Normal.Cost currentCost)
          (put.value thisunit 'Normal.Duration currentDuration)
          (put.value thisunit 'Normal.Cost.per.Day currentCostperDay))
    (unimsg thisunit 'Calculate.Next.Estimate)
    (unimsg thisunit 'Calculate.Delta.Cost)
    (unimsg thisunit 'Calculate.Delta.Duration))

Page 42
;;; **-- Syntax: Common-lisp; Package: K; Mode: LISP **--
;;; (in-package 'kee)

;;;; This file contains all the active value functions used in
;;;; JANUS.

;;;; AVPUT functions all need the following arguments:
;;;;
;;;;   self: the unit containing the messaged method
;;;;   slot: the slot to which the active value is attached
;;;;   newvalues: a list of new values being put into the slot
;;;;   oldvalues: a list of the values which were previously in the slot
;;;;   unit: the unit containing the attached slot
;;;;   slottype: the type of the attached slot: 'member or 'own
;;;;
;;;;   and the following optional arguments:
;;;;
;;;;   world: the KEEworld in which the active value is fired
;;;;   world.inherit.flg ?????.

;;;; ******************************
;;;; The following functions work on PARTICIPANTS frames
;;;; ******************************

;;;; EMPLOYER-AVPUT-FN is the active value function attached to the Employs
;;;; slot of all participants. It insures that each participant's Employed.by
;;;; slot gets updated when its Employs slot changes.

(defun EMPLOYER-AVPUT-FN (self slot newvalues oldvalues unit slottype
  &optional world world.inherit.flg)
  (unitmsg unit 'Clear.Employees oldvalues)
  (unitmsg unit 'Notify.Employees
            newvalues))

;;;; CALC-DAY-VALUE-AVPUT-FN is the value of the 'avput' slot in the active
;;;; value unit Calc.Day.Value.AV attached to the Contract and Employed.by
;;;; slots of a participant.

(defun CALC-DAY-VALUE-AVPUT-FN (self slot nValues oValues unit slottype
  &optional world world.inherit.flg)
  (unitmsg unit 'Determine.Day.Value
            nValues))
;;; The following functions work on ACTIVITIES frames
;;; **********************************************

;;; CALC-NETWORK-AVPUT-FN is the value of the 'avput' slot in the active
;;; value unit Calc.Network.AV attached to the Current.Duration slot of
;;; an activity. It sends the Calculate.Network message to the activity
;;; whenever its Current.Duration value is changed.

(defun CALC-NETWORK-AVPUT-FN (self slot newvals
oldvals unit slottype
optional world world.inherit.flg)
(unitmsg unit 'Calculate.Network
newvals)

;;; CALC-DURATION-AVPUT-FN is the value of the 'avput' slot in
;;; CALC.DURATION.AV, the active value unit attached to several slots
;;; of an activity. It insures that the activity's current duration
;;; gets recalculated whenever any of its inputs change, including:
;;; - its MH.Estimate
;;; - its Current.Crew
;;; - its Current.Tactic

(defun CALC-DURATION-AVPUT-FN (self slot newvals oldvals unit slottype
optional world world.inherit.flg)
(unitmsg unit 'Calculate.Duration nil)
newvals)

;;; CALC-ESTIMATE-AVPUT-FN is the value of the 'avput' slot in
;;; CALC.ESTIMATE.AV, the active value unit attached to several slots
;;; of an activity. It insures that the activity's estimate gets
;;; updated whenever any of its inputs change, including:
;;; - its MH.Estimate
;;; - its Material.Estimate
;;; - its Current.Crew.
;;; The following slots may also affect an activity's estimate,
;;; but they are one step removed from the activity & will need
;;; another active value (not implemented as of 7/31):
;;; - its Current.Crew's Direct.Cost.per.MH
;;; - its Current.Crew's Labor.Fringe.Rate
;;; - its Current.Crew's Crew.Size
;;; - its Current.Crew's Equipment.Cost.per.CH

(defun CALC-ESTIMATE-AVPUT-FN (self slot newvals oldvals unit slottype
optional world world.inherit.flg)
(unitmsg unit 'Calculate.Estimate nil)
newvals)
;;; CALC-NEXT-ESTIMATE-AVPUT-FN is the value of the 'avput' slot
;;; in CALC-NEXT-ESTIMATE-AV, the active value unit attached to
;;; several slots of an activity, including:
;;;   - its Current.Tactic

(defun CALC-NEXT-ESTIMATE-AVPUT-FN (self slot newvals oldvals
    unit slottype
    &optional world world.inherit.flg)

  (unitmsg unit 'Calculate.Next.Estimate)
  newvals)

;;; CALC-COST-SLOPE-AVPUT-FN is the value of the 'avput' slot in
;;; CALC-COST-SLOPE-AV, the active value unit attached to the:
;;;   Current.Cost.per.Day and
;;;   Current.Tactic
;;; slots of an activity. It insures that the activity's Current.
;;; Cost.Slope gets updated whenever a change is made to its

(defun CALC-COST-SLOPE-AVPUT-FN (self slot newvalues oldvalues
    unit slottype
    &optional world world.inherit.flg)

  (unitmsg unit 'Calculate.Cost.Slope)
  newvalues)

;;; CALC-COST-PER-DAY-AVPUT-FN is the value of the 'avput' slot in
;;; CALC-COST-PER-DAY-AV, the active value unit attached to the
;;; Current.Cost slot of an activity. It insures that the activity's
;;; Current.Cost.per.Day gets updated whenever a change is made to
;;; its Current.Cost.

(defun CALC-COST-PER-DAY-AVPUT-FN (self slot nvalues ovalues
    unit slottype
    &optional world world.inherit.flg)

  (unitmsg unit 'Calculate.Current.Cost.per.Day)
  nvalues)

;;; CALC-PROJECT-ESTIMATE-AVPUT-FN is the value of the 'avput' slot
;;; in CALC-PROJECT-AV, the active value attached to several slots
;;; of an activity. It insures that the project duration and estimate
;;; get updated whenever there is a change to any activity durations
;;; or estimates. The activity slots to which this active value are
;;; bound include:
;;;   - Current.Duration
;;;   - Current.Cost.
;;; Note that the project, Pier.Project, is hard-wired. Another way of
;;; dealing with this must be developed to handle multiple projects.

(defun CALC-PROJECT-ESTIMATE-AVPUT-FN (self slot nvalues
    ovalues unit slottype
    &optional world world.inherit.flg)

  (unitmsg 'Pier.Project 'Calculate.Project.Estimate)
  nvalues)
;;; ***********************************************
;;; The following function is really a PROJECT function, but actually
;;; goes with the PROJECT.COMPLETE activity frame.
;;; ***********************************************

;;; CALC-PROJECT-DURATION-AVPUT-FN is the value of the 'avput slot
;;; in CALC.PROJECT.DURATION.AV, the active value attached to the
;;; Late.Finish slot of the PROJECT.COMPLETE unit. It insures that
;;; the project PIER.PROJECT gets its duration updated whenever it
;;; changes.

(defun CALC-PROJECT-DURATION-AVPUT-FN (self slot nvals
  ovals unit slottype
  &optional world
  world.inherit.flg)
  (unitmsg 'Pier.Project 'Calculate.Project.Duration)
nvals)

;;; ***********************************************
;;; The following functions work with TACTICS frames
;;; ***********************************************

;;; TACTIC-SHIFT-FACTOR-AVPUT-FN is the value of the avput slot
;;; in the Tactic.Shift.Factors.AV unit, attached to the Shifts.per.Day
;;; slot of a tactic. It updates the tactic's Hours.Straight and its
;;; Hours.Shift.Premium.

(defun TACTIC-SHIFT-FACTOR-AVPUT-FN (self slot newValues oldValues
  unit slottype
  &optional world world.inherit.flg)
  (unitmsg unit 'Calculate.Shift.Factors)
nNewValues)

;;; TACTIC-PREMIUM-HOURS-AVPUT-FN is the value of the avput slot
;;; in the Tactic.Premium.Hours.AV unit, attached to the
;;; Hours.per.Shift and Days.per.Week slots of a tactic. It
;;; updates the tactic's Hours.1.5 and Hours.2 slots.

(defun TACTIC-PREMIUM-HOURS-AVPUT-FN (self slot nValues oValues
  unit slottype
  &optional world world.inherit.flg)
  (unitmsg unit 'Calculate.Time.and.a.Half)
  (unitmsg unit 'Calculate.Double.Time)
nNewValues)

;;; TACTIC-HOURS-PAID-AVPUT-FN is the value of the avput slot in the
;;; Tactic.Hours.Paid.AV unit, attached to the Hours.Straight and
;;; Hours.1.5 slots of a tactic. It updates the tactic's Hours.Paid slot.

(defun TACTIC-HOURS-PAID-AVPUT-FN (self slot newVals
  oldVals unit slottype
  &optional world world.inherit.flg)
  (unitmsg unit 'Calculate.Hours.Paid)
nNewVals)
;;; TACTIC-HOURS-WORKED-AVPUT-FN is the value of the avput slot in
;;; the Tactic.Hours.Worked.AV unit, attached to the Hours.Straight
;;; and Hours.1.5 slots of a tactic. It updates the tactic's
;;; Hours.Worked slot.

(defun TACTIC-HOURS-WORKED-AVPUT-FN (self slot nVals oVals
    unit slottype
    &optional world world.inherit.flg)
    (unitmsg unit 'Calculate.Hours.Worked) nVals)

;;; TACTIC-OT-PREMIUM-AVPUT-FN is the value of the avput slot in the
;;; Tactic.OT.Premium.AV unit, attached to the Hours.1.5 slot of a
;;; tactic. It updates the tactic's Hours.OT.Premium slot.

(defun TACTIC-OT-PREMIUM-AVPUT-FN (self slot newVs oldVs
    unit slottype
    &optional world world.inherit.flg)
    (unitmsg unit 'Calculate.Overtime.Premium) newVs)

;;; TACTIC-EFFECTIVE-HOURS-AVPUT-FN is the value of the avput slot in
;;; the Tactic.Effective.Hours.AV unit, attached to the Hours.Worked
;;; and Efficiency slots of a tactic. It updates the tactic's
;;; Hours.Effective slot.

(defun TACTIC-EFFECTIVE-HOURS-AVPUT-FN (self slot nVs oVs
    unit slottype
    &optional world world.inherit.flg)
    (unitmsg unit 'Calculate.Effective.Hours) nVs)

;;; TACTIC-DURATION-IMPROVEMENT-AVPUT-FN is the value of the avput slot
;;; in the Tactic.Duration.Improvement.AV unit, attached to the
;;; Hours.Effective slot of a tactic. It updates the tactic's
;;; Duration.Improvement slot.

(defun TACTIC-DURATION-IMPROVEMENT-AVPUT-FN (self slot nwValues odValues
    unit slottype
    &optional world world.inherit.flg)
    (unitmsg unit 'Calculate.Duration.Improvement) nwValues)

;;; TACTIC-COST-PREMIUM-AVPUT-FN is the value of the avput slot in
;;; the Tactic.Cost.Premium.AV unit, attached to the Hours.Effective
;;; and Hours.Paid slots of a tactic. It updates the tactic's
;;; Cost.Premium slot.

(defun TACTIC-COST-PREMIUM-AVPUT-FN (self slot nwVals oDVals
    unit slottype
    &optional world world.inherit.flg)
    (unitmsg unit 'Calculate.Cost.Premium) nwVals)
;;; TACTIC-COST-SLOPE-MULTIPLIER-AVPUT-FN is the value of the avput
;;; slot in the Tactic.Cost.Slope.Multiplier.AV unit, attached to the
;;; Duration.Improvement and Cost.Premium slots of a tactic. It
;;; updates the tactic's Cost.Slope.Multiplier slot.

(defun TACTIC-COST-SLOPE-MULTIPLIER-AVPUT-FN (self slot nWVs
    odWVs unit slottype
    &optional world world.inherit.fig)
  (unitmsg unit 'Calculate.Cost.Slope.Multiplier)
  nWVs)
;;; -*- Syntax: Common-lisp; Package: K; Mode: LISP -*-
(in-package 'kee)

;;;; This file contains all the functions necessary for JANUS to
;;;; perform critical path method (CPM) calculations. It goes with
;;;; the knowledge base 'janus.u' and the ancillary functions
;;;; contained in the file 'janusfn.s.lisp.' The functions in this
;;;; file operate on the units in the class of ACTIVITIES.

;;;; Here is the procedure for using these functions:
;;;; 1. First load them into the lisp environment.
;;;; 2. Next, message the first activity in the network to
;;;;    initialize the network. This clears all the dates
;;;;    from each activity's slots and correctly hooks up its
;;;;    Predecessors slot to match its predecessors' successors.
;;;; 3. Finally, message the first activity to calculate the network.
;;;; After doing this, changes to durations will propagate through
;;;; the network automatically, due to the active value attached
;;;; to each activity's Current.Duration slot.

;;;; LCPM is just a function to load this file. It makes reloading
;;;; after making changes easier. Note the pathname. Be sure to change
;;;; this when moving to another machine or moving the folder elsewhere.

(defun LCPM ()
  (load "297ml1:hd:alan:januscpm.lisp"))

;;;; CLEAR-PREDECESSORS-FN is the value of the Clear.Predecessors slot
;;;; for each activity. It removes all values of the Predecessors slot
;;;; of the activity.

(defun CLEAR-PREDECESSORS-FN (thisunit)
  (remove.all.values thisunit 'Predecessors))

;;;; CLEAR-LAST-PREDECESSORS-FN is the value of the Clear.Predecessors
;;;; slot of the last activity in the network.

(defun CLEAR-LAST-PREDECESSORS-FN (thisunit)
  (remove.all.values thisunit 'Predecessors))

;;;; ADD-PREDECESSOR-FN is the value of the Add.Predecessor slot
;;;; for each activity. It adds a value to the Predecessors slot of the
;;;; activity in response to receiving a message from a predecessor.

(defun ADD-PREDECESSOR-FN (thisunit predecessorunit)
  (add.value thisunit 'Predecessors predecessorunit))
;;; NOTIFICATION-SUCCESSORS-FN is the value of the Notify.Successors slot
;;; for each activity. It sends a message to each of the activities in
;;; its Successors slot to first add it to their Predecessors slot,
;;; then notify its successors.

(defun NOTIFICATION-SUCCESSORS-FN (thisunit)
  (let ((successorList (get.values thisunit 'Successors)))
    (dolist (successor successorList)
      (unitmsg successor 'Add.Predecessor thisunit)
      (unitmsg successor 'Notify.Successors)))

;;; NOTIFICATION-LAST-SUCCESSORS-FN is the value of the Notify.Successors
;;; slot of the last activity in the network.

(defun NOTIFICATION-LAST-SUCCESSORS-FN (thisunit)
  t)

;;; INITIALIZE-ACTIVITY-FN is the value of the Initialize.Activity
;;; slot for each activity. It sets an activity's Early.Start,
;;; Last.EF, Late.Finish and Last.LS slots to 0. This is necessary
;;; initially to avoid inheritance problems when arithmetic is
;;; performed using 'unknown' values as arguments.

;;; (It also sets its Current.Crew to its Preferred.Crew, its Current.
;;; Cost to its Normal.Cost and finally its Current.Duration to its
;;; Normal.Duration.) -- Removed --

(defun INITIALIZE-ACTIVITY-FN (thisunit)
  (put.value thisunit 'Early.Start 0)
  (put.value thisunit 'Early.Finish 0)
  (put.value thisunit 'Late.Start 0)
  (put.value thisunit 'Late.Finish 0)
  (put.value thisunit 'Critical 'Undetermined)
  (put.value thisunit 'Current.Crew
    (get.value thisunit 'Preferred.Crew))
  (remove.all.values thisunit 'Current.Tactic))

; (let ((successors (get.values thisunit 'Successors)))
;   (dolist (successor successors)
;     (unitmsg successor 'Initialize.Activity))))
;;; INITIALIZE-NETWORK-FN is the value of the Initialize.Network slot
;;; for each activity. It messages all activities in the project
;;; to clear their Predecessors slot, put 0 into their ES, EF, LF, and
;;; LS slots, then notify all successors to hook up their predecessors.
;;; 6/10/89: It also includes the estimate input and initial duration
;;; calculations for the network. This requires functions
;;; which are defined in the file 'janusdur.lisp.'
;;; 7/8/89: I commented out the estimate input & duration calcs for
;;; debugging purposes.
;;; 7/31/89: I added a message to each activity to calculate its
;;; duration. This function is defined in the file
;;; janusdur.lisp, which must be loaded along with this one.

(defun INITIALIZE-NETWORK-FN (thisunit)
  (unitmsg thisunit 'Clear.Predecessors)
  (unitmsg thisunit 'Initialize.Activity)
  (unitmsg thisunit 'Reset.Activity) ; from JANUSACT.LISP
  (let ((successors (get.values thisunit 'Successors)))
    (dolist (successor successors)
      (unitmsg successor 'Initialize.Network)))
  ;(unitmsg thisunit 'Notify.Successors))

;;; CALCULATE-EF-FN is the value of the Calculate.EF slot for each
;;; activity. It calculates an activity's early finish date from
;;; its early start date and its current duration.

(defun CALCULATE-EF-FN (thisunit)
  (let ((EF (+ (get.value thisunit 'Early.Start)
               (get.value thisunit 'Current.Duration)))
        (put.value thisunit 'Early.Finish EF)))

;;; CALCULATE-LS-FN is the value of the Calculate.LS slot for each
;;; activity. It calculates an activity's late start date from its
;;; late finish date and its current duration.

(defun CALCULATE-LS-FN (thisunit)
  (let ((oldLS (get.value thisunit 'Late.Start))
        (newLS (- (get.value thisunit 'Late.Finish)
                  (get.value thisunit 'Current.Duration)))
        (cond ((equal oldLS newLS) oldLS)
              (t (put.value thisunit 'Last.LS oldLS)
                  (put.value thisunit 'Late.Start newLS)))))

;;; CLEAR-PRED-EFS-FN is the value of the Clear.Pred.EFs slot
;;; for each activity. It clears all values from its Pred.EFs slot.

(defun CLEAR-PRED-EFS-FN (thisunit)
  (remove.all.values thisunit 'Pred.EFs))
;;; COLLECT-PRED-EFS-FN is the value of the Collect.Pred.EFs slot
;;; for each activity. It gets the early finish date of each of
;;; its predecessors, and puts those values into its Pred.EFs slot.

(defun COLLECT-PRED-EFS-FN (thisunit)
  (let ((predList (get.values thisunit 'Predecessors))
         (cond ((null predList) (put.value thisunit 'Pred.EFs 0))
                (t (dolist (pred predList)
                        (add.values thisunit 'Pred.EFs
                                     (list (get.value pred 'Early.Finish))))))

;;; CLEAR-SUCC-LS-LS-FN is the value of the Clear.Succ.LSs slot for
;;; each activity. It clears all values from its Succ.LSs slot.

(defun CLEAR-SUCC-LS-LS-FN (thisunit)
  (remove.all.values thisunit 'Succ.LSs)

;;; COLLECT-SUCC-LS-LS-FN is the value of the Collect.Succ.LSs slot
;;; for each activity. It gets the late start date of each of its
;;; successors and puts those values into its Succ.LSs slot.

(defun COLLECT-SUCC-LS-LS-FN (thisunit)
  (let ((succList (get.values thisunit 'Successors))
         (cond ((null succList)
                  ; If there are no successors, set the value of
                  ; the slot to the activity's own early finish date.
                  (put.value thisunit 'Succ.LSs
                              (get.value thisunit 'Early.Finish)))
               ; Otherwise, clear the slot and collect the values
               ; from its successors.
               (t (dolist (succ succList)
                       (add.values thisunit 'Succ.LSs
                                    (list (get.value succ 'Late.Start)))))))

;;; LIST-MAX is a function to calculate the maximum value of a list
;;; of numbers. The LISP 'max' function requires separate numbers as
;;; arguments. LIST-MAX takes just a single list as an argument and
;;; returns the maximum number in the list.

(defun LIST-MAX (numList maxSoFar)
  (let ((number (car numList))
         (restOfList (cdr numList)))
    (cond ((null numList) maxSoFar)
          (> number maxSoFar) (LIST-MAX restOfList number))
          (t (LIST-MAX restOfList maxSoFar))))
;;; LIST-MIN is a function to calculate the minimum value of a list
;;; of numbers. The LISP 'min' function requires separate numbers as
;;; arguments. LIST-MIN takes just a single list as an argument and
;;; returns the minimum number in the list.

(defun LIST-MIN (numList minSoFar)
  (let ((number (car numList))
        (restOfList (cdr numList)))
    (cond ((null numList) minSoFar)
          ((< number minSoFar) (LIST-MIN restOfList number))
          (t (LIST-MIN restOfList minSoFar))))

;;; CHECK-PRED-EFS-FN is the value of the Check.Pred.EFs slot of
;;; each activity. It looks at the Forward.Flag of each of its
;;; predecessors. If any are False, it sends that predecessor
;;; the Calculate.ES message. If not, it simply returns.

(defun CHECK-PRED-EFS-FN (thisunit)
  (let ((predList (get.values thisunit 'Predecessors))
        (dolist (pred predList)
          (cond ((equal 'False (get.value pred 'Forward.Flag))
                (unitmsg pred 'Calculate.ES))
                (t t)))))

;;; CALCULATE-ES-FN is the value of the Calculate.ES slot of each
;;; activity. It sets its Early.Start slot equal to the maximum
;;; value in its Pred.EFs slot.

(defun CALCULATE-ES-FN (thisunit)
  (unitmsg thisunit 'Clear.Pred.EFs)
  (unitmsg thisunit 'Collect.Pred.EFs)
  (put.value thisunit 'Early.Start
              (LIST-MAX (get.values thisunit 'Pred.EFs) 0)))

;;; CALCULATE-FIRST-ES-FN is the value of the Calculate.ES slot
;;; of the first activity in the network.

(defun CALCULATE-FIRST-ES-FN (thisunit)
  (put.value thisunit 'Early.Start 0))

;;; CHECK-SUCC-LS-FN is the value of the Check.Succ.LFs slot of
;;; each activity. It looks at the Backward.Flag of each of its
;;; successors. If any are False, it sends that successor
;;; the Calculate.LF message. If not, it simply returns.

(defun CHECK-SUCC-LS-FN (thisunit)
  (let ((succList (get.values thisunit 'Successors))
        (dolist (suc succList)
          (cond ((equal 'False (get.value suc 'Backward.Flag))
                (unitmsg suc 'Calculate.LF))
                (t t)))))

Page 53
;;;; CALCULATE-LF-FN is the value of the Calculate.LF slot of each
;;;; activity. It sets its Early.Start slot equal to the minimum
;;;; value in its Succ.LSs slot.
(defun CALCULATE-LF-FN (thisunit)
  (unimsg thisunit 'Clear.Succ.LSs)
  (unimsg thisunit 'Check.Succ.LSs)
  (unimsg thisunit 'Collect.Succ.LSs)
  (put.value thisunit 'Late.Finish
    (LIST-MIN (get.values thisunit 'Succ.LSs) 99999))
)

;;;; CALCULATE-LAST-LF-FN is the value of the Calculate.LF slot of
;;;; the last activity in the network.
(defun CALCULATE-LAST-LF-FN (thisunit)
  (put.value thisunit 'Late.Finish
    (get.value thisunit 'Early.Finish))
)

;;;; CALCULATE-FLOAT-FN is the value of the Calculate.Float slot of
;;;; each activity. It sets its float slot equal to its Late.Start
;;;; minus its Early.Start. If Float is 0, it also sets its Critical
;;;; slot to True.
(defun CALCULATE-FLOAT-FN (thisunit)
  (let ((newFloat (+ (get.value thisunit 'Late.Start)
                     (get.value thisunit 'Early.Start) 1))
    (put.value thisunit 'Float newFloat)
    (cond ((equal newFloat 0) (put.value thisunit 'Critical 'True))
      (t (put.value thisunit 'Critical 'False))))
)

;;;; SET-FORWARD-FLAG-FN sets the Forward.Flag of an activity to
;;;; False.
(defun SET-FORWARD-FLAG-FN (thisunit)
  (put.value thisunit 'Forward.Flag 'False)
  (let ((successors (get.values thisunit 'Successors))
    (dolist (successor successors)
      (unimsg successor 'Set.Forward.Flag))))

;;;; SET-LAST-FORWARD-FLAG-FN is the value of the Set.Forward.Flag
;;;; slot of the last activity in the network.
(defun SET-LAST-FORWARD-FLAG-FN (thisunit)
  (put.value thisunit 'Forward.Flag 'False)
  (unimsg thisunit 'Set.Backward.Flag))
;;; SET-BACKWARD-FLAG-FN sets the Backward.Flag of an activity to
;;; False.
(defun SET-BACKWARD-FLAG-FN (thisunit)
  (put.value thisunit 'Backward.Flag 'False)
  (let ((predecessors (get.values thisunit 'Predecessors)))
    (dolist (predecessor predecessors)
      (unimsg predecessor 'Set.Backward.Flag)))))

;;; SET-FIRST-BACKWARD-FLAG-FN is the value of the Set.Backward.
;;; Flag slot of the first activity in the network.
(defun SET-FIRST-BACKWARD-FLAG-FN (thisunit)
  (put.value thisunit 'Backward.Flag 'False))

;;;;* INITIALIZE-FORWARD-FLAGS-FN sets the Forward.Flag of an
;;;;* activity and all activities downstream from it in the network
;;;;* to False.
(defun INITIALIZE-FORWARD-FLAGS-FN (thisunit)
  (let ((successors (get.values thisunit 'Successors)))
    (unimsg thisunit 'Set.Forward.Flag)
    (cond ((null successors) nil)
          (t (dolist successor successors)
             (unimsg successor 'Initialize.Forward.Flags))))))

;;;;* INITIALIZE-BACKWARD-FLAGS-FN sets the Backward.Flag of an
;;;;* activity and all activities upstream from it in the network
;;;;* to False.
(defun INITIALIZE-BACKWARD-FLAGS-FN (thisunit)
  (let ((predecessors (get.values thisunit 'Predecessors)))
    (unimsg thisunit 'Set.Backward.Flag)
    (cond ((null predecessors) nil)
          (t (dolist predecessor predecessors)
             (unimsg predecessor 'Initialize.Backward.Flags))))))

;;;;* SET-FLAGS-FN sets the Forward.Flag and Backward.Flag of all
;;;;* activities affected by a change in the network. Note that
;;;;* the 'backward pass' of this procedure is hard-wired to the
;;;;* activity Project.Complete.
(defun SET-FLAGS-FN (thisunit)
  (unimsg thisunit 'Set.Forward.Flag)
  (let ((successors (get.value thisunit 'Successors)))
    (dolist (successor successors)
      (unimsg successor 'Set.Flags))))
;;; FORWARD-PASS-FN calculates its own early start and early finish
;;; dates.
(defun FORWARD-PASS-FN (thisunit)
  (unitmsg thisunit 'Calculate.ES)
  (unitmsg thisunit 'Calculate.EF)
  (put.value thisunit 'Forward.Flag 'True)
  (let ((successors (get.values thisunit 'Successors))
      (dolist (successor successors)
        (unitmsg successor 'Forward.Pass))))

;;; LAST-FORWARD-PASS-FN is the value of the Forward.Pass slot of
;;; the last activity in the network.
(defun LAST-FORWARD-PASS-FN (thisunit)
  (unitmsg thisunit 'Calculate.ES)
  (unitmsg thisunit 'Calculate.EF)
  (put.value thisunit 'Forward.Flag 'True)
  (unitmsg thisunit 'Backward.Pass))

;;; BACKWARD-PASS-FN calculates its own late finish and late start
;;; dates, its float and its criticality.
(defun BACKWARD-PASS-FN (thisunit)
  (unitmsg thisunit 'Calculate.LF)
  (unitmsg thisunit 'Calculate.LS)
  (unitmsg thisunit 'Calculate.Float)
  (put.value thisunit 'Backward.Flag 'True)
  (let ((predecessors (get.values thisunit 'Predecessors))
      (dolist (predecessor predecessors)
        (unitmsg predecessor 'Backward.Pass))))

;;; FIRST-BACKWARD-PASS-FN is the value of the Backward.Pass slot
;;; of the first activity in the network.
(defun FIRST-BACKWARD-PASS-FN (thisunit)
  (unitmsg thisunit 'Calculate.LF)
  (unitmsg thisunit 'Calculate.LS)
  (unitmsg thisunit 'Calculate.Float)
  (put.value thisunit 'Backward.Flag 'True))

;;; CALCULATE-NETWORK-FN puts it all together. It starts with a
;;; message to one activity. It then messages that activity's
;;; successors to perform their forward pass calculations. It continues
;;; until it reaches the last activity in the network, whereupon it
;;; begins messaging predecessor activities to perform their
;;; backward pass calculations. It stops when it reaches the first
;;; activity in the network.
(defun CALCULATE-NETWORK-FN (thisunit)
  (unitmsg thisunit 'Set.Forward.Flag)
  (unitmsg thisunit 'Forward.Pass))
;;; -*- Syntax: Common-lisp; Package: K; Mode: LISP -*-
(in-package 'kee)

;;;; This file contains all functions associated with Crew frames
;;;; in the knowledge base JANUS.

;;;; CALCULATE-COMPOSITE-CREW-RATE-FN is the value of the Calculate.
;;;; Composite.Crew.Rate slot in Crews frames. It sets the value of
;;;; the slot Composite.Crew.Rate to the sum of its Direct.Cost.per.MH
;;;; times 1 + its Labor.Fringe.Rate (getting the direct labor cost
;;;; plus fringes) and its Equipment.Cost.per.CH.
(defun CALCULATE-COMPOSITE-CREW-RATE-FN (thisunit)
  (let ((directRate (get.value thisunit 'Direct.Cost.per.MH))
        (fringeRate (get.value thisunit 'Labor.Fringe.Rate))
        (equipRate (get.value thisunit 'Equipment.Cost.per.CH)))
    (put.value thisunit 'Composite.Crew.Rate
        (+ (* directRate (1+ fringeRate)) equipRate))))
;;; -*- Syntax: Common-lisp; Package: K; Mode: LISP -*-
(in-package 'kee)

;;; This is the functions file for JANUS. It operates in conjunction
;;; with the knowledge base 'janus.u'. This file contains functions for
;;; manipulating members of the class of PARTICIPANTS.

;;; 5/17/89: Added a before wrapper to the load.kb slot of the janus
;;; unit in the system kb knowledgebases to load this file
;;; automatically when the janus kb is loaded.

;;; 9/18/89: Changed the name of this file from janusfns to januspar
;;; and removed the loading functions.

;;; This file contains the following function definitions:

;;; INIT-PARTICIPANT-FN is the value of the Initialize.Participant slot of
;;; participant frames. The function clears the results of prior
;;; reasoning regarding the slots Contract.Importance, Day.Value and
;;; Employer.Relationship. That is, it sets the value of each of those
;;; slots for the frame 'thisunit' to NIL.
;;; 9/18/89: Commented out the contract importance and employer
;;; relationship stuff. I will leave this to a future
;;; extension.

(defun INIT-PARTICIPANT-FN (thisunit)
  (remove.all.values thisunit 'Contract.Importance)
  (remove.all.values thisunit 'Day.Value)
  (remove.all.values thisunit 'Employer.Relationship))

;;; CLEAR-EMPLOYER-FN is the value of the Clear.Employer slot for each
;;; participant. It removes any value of the Employed.by slot of the
;;; participant.

(defun CLEAR-EMPLOYER-FN (thisunit)
  (remove.all.values thisunit 'Employed.by))

;;; SET-EMPLOYER-FN is the value of the Set.Employer slot for each
;;; participant. It sets the Employed.by slot of the participant in
;;; response to receiving a message from its employer.

(defun SET-EMPLOYER-FN (thisunit employerunit)
  (put.value thisunit 'Employed.by employerunit))
;;; CLEAR-EMPLOYEES-FN is the value of the Clear.Employees slot for each
;;; participant. It sends a message to each of the participants in its
;;; Employs slot to clear the current value of its Employed.by slot.
;;; The optional argument allows the function to message the old values of
;;; its Employs slot when triggered by an active value function.

(defun CLEAR-EMPLOYEES-FN (thisunit &optional oldValues)
  (cond ((null oldValues)
   (let ((employeeList (get.values thisunit 'Employs)))
     (dolist (employee employeeList)
       (unitmsg employee 'Clear.Employer))))
   (t (dolist (oldEmployee oldValues)
       (unitmsg oldEmployee 'Clear.Employer))))))

;;; NOTIFY-EMPLOYEES-FN is the value of the Notify.Employees slot for each
;;; participant. It sends a message to each of the participants in its
;;; Employs slot to set its Employed.by slot.

(defun NOTIFY-EMPLOYEES-FN (thisunit)
  (let ((employeeList (get.values thisunit 'Employs)))
    (dolist (employee employeeList)
      (unitmsg employee 'Set.Employer thisunit))))
;;; DETERMINE-DAY-VALUE-FN is the value of the Determine.Day.Value slot
;;; of a participant. It gives participants the following day values:
;;; If the participant is an owner, do nothing, otherwise
;;; If the participant's contract is a lump sum contract, then:
;;;   if the participant is a General.Contractor, then
;;;     its day value equals its daily overhead rate, otherwise
;;;   its day value is 0.
;;; If the participant's contract is a cost plus contract, then:
;;;   its day value equals that of its employer.
;;; This function can be easily extended.

(defun DETERMINE-DAY-VALUE-FN (thisunit)
  (let ((employer (get.value thisunit 'Employed.by))
         (contract (get.value thisunit 'Contract))
         (owner
          (car
           (member thisunit
                    (unit.descendants 'Owners 'member))))
         (commoditySupplier
          (car
           (member thisunit
                    (unit.descendants 'Commodity.Suppliers 'member))))
         (gContractor
          (car
           (member thisunit
                    (unit.descendants 'General.Contractors 'member))))

; More variables may be defined here to allow more factors to
; be considered in the determination of day value.

  (cond ((null employer) nil)
          (owner
           (commoditySupplier (put.value thisunit 'Day.Value
                                          (get.value employer 'Day.Value)))
           (member contract (unit.descendants 'Lump.Sum.Contracts 'member))
           (cond (gContractor (put.value thisunit 'Day.Value
                                            (get.value thisunit 'Daily.OH.Rate)))
                  (t (put.value thisunit 'Day.Value 0))))
          ((member contract (unit.descendants 'Cost.Plus.Contracts 'member))
           (put.value thisunit 'Day.Value
                        (get.value employer 'Day.Value))
           (t nil))))

;;; DISPLAY-ORG-CHART is a function to display the project organizational
;;; chart in a vertical fashion.

(defun DISPLAY-ORG-CHART ()
  (graph-slot 'janus 'owner.1 'Employed
              :tangled-p t
              :horizontal-p nil
              :title "Pier Project Organizational Chart")
;;; DETERMINE-CONTRACT-TYPE-FN is the value of the Determine.Contract.Type
;;; slot of a participant. The idea here is that a general contractor's
;;; contract type should carry to its employees.

(defun DETERMINE-CONTRACT-TYPE-FN (thisunit)
  (let ((owner
         (car
          (member thisunit
           (unit.descendants 'Owners 'member)))))
    (gContractor
     (car
      (member thisunit
       (unit.descendants 'General.Contractors 'member)))
     (employer (get.value thisunit 'Employed.by))
     (cond (owner)
           (gContractor)
           (t (put.value thisunit 'Contract
                           (get.value employer 'Contract))))))

;;; INITIALIZE-ORG-CHART-FN is the value of the Initialize.Org.Chart slot
;;; of a participant. It clears the Employed.by slots of all participants
;;; downstream from the messaged participant, then re-sets them, then
;;; messages each downstream participant to propagate the initialization.

(defun INITIALIZE-ORG-CHART-FN (thisunit)
  (let ((employeeList (get.values thisunit 'Employs))
        (unitmsg thisunit 'Clear.Employees)
        (unitmsg thisunit 'Notify.Employees)
        (unitmsg thisunit 'Determine.Contract.Type)
        (unitmsg thisunit 'Determine.Day.Value)
        (dolist (employee employeeList)
                 (unitmsg employee 'Initialize.Org.Chart)))))
;;; -- Syntax: Common-lisp; Package: K; Mode: LISP --
(in-package 'kee)

;;; This file contains the functions to calculate project
;;; schedule and cost data on the knowledge base JANUS.

;;; CALCULATE-PROJECT-DURATION-FN is the value of the Calculate.
;;; Project.Duration slot in the unit Project. It determines
;;; the overall duration of the project and puts the result in
;;; the Project.Duration slot of the Project unit. It requires
;;; the functions in the file 'januscpm' in order to work.
;;; Note that this function is hard-wired to the unit
;;; Project.Complete.

(defun CALCULATE-PROJECT-DURATION-FN (thisunit)
  (put.value thisunit 'Project.Duration
    (get.value 'Project.Complete 'Late.Finish)))

;;; COLLECT-PROJECT-ACTIVITIES is an ancillary function that
;;; traverses the project network and constructs a list of
;;; all the activities in the network.

(defun COLLECT-PROJECT-ACTIVITIES () ;(activityList)
  (REMOVE-NIL
    (remove-duplicates
      (flatten
        (COLLECT-ACTIVITY-LISTS
          (list (unit 'Notice.to.Proceed))))))
    ; (let ((activity (car activityList))
    ;       (otherActivities (cdr activityList)))
    ;    (cond ((null activity) nil)
    ;           ((null otherActivities)
    ;            (cons
    ;             activity
    ;             (COLLECT-PROJECT-ACTIVITIES
    ;              (get.values activity 'Successors))))
    ;     (t
    ;      (cons
    ;       activity
    ;       (COLLECT-PROJECT-ACTIVITIES otherActivities))))))

;;; REMOVE-NIL is an ancillary function that removes nil from an
;;; input list.

(defun REMOVE-NIL (inputList)
  (let ((atom (car inputList))
         (restOfList (cdr inputList)))
    (cond ((null atom) nil)
          (t (cons atom (REMOVE-NIL restOfList)))))
;;; GET-ACTIVITY-SUCCESSOR-LIST returns the list of successors of
;;; the activity fed in as an argument.

(defun GET-ACTIVITY-SUCCESSOR-LIST (activity)
  (get.values activity 'Successors))

;;; COLLECT-ACTIVITY-LISTS is an ancillary function that
;;; traverses the network and builds a list of successor lists
;;; of all the activities in the network.

(defun COLLECT-ACTIVITY-LISTS (activityList)
  (let ((activity (car activityList))
        (otherActivities (cdr activityList)))
    (cond ((null activity) nil)
          (t (cons activity
              (cons (COLLECT-ACTIVITY-LISTS
                      (GET-ACTIVITY-SUCCESSOR-LIST activity))
                   (COLLECT-ACTIVITY-LISTS otherActivities)))))

;;; CALCULATE-PROJECT-ESTIMATE-FN is the value of the Calculate.
;;; Project.Estimate slot in the unit Projects. It sums the
;;; Equipment.Estimate, Labor.Estimate, Material.Estimate and
;;; MH.Estimate slots of all activities in the network and puts
;;; the results in the Project.Estimate, Project.Labor.
;;; Estimate, Project.Material.Estimate and Project.MH.Estimate
;;; slots of the unit in the class of Projects. It then sums all
;;; these slots and puts that value in its Project.Total.Estimate
;;; slot.

(defun CALCULATE-PROJECT-ESTIMATE-FN (thisunit)
  (let ((activityList (get.value thisunit 'Project.Activities)))
    (setq equipment 0
          labor 0
          fringes 0
          material 0
          mh 0)
    (dolist (activity activityList)
      (setq equipment (+ equipment (get.value activity 'Equipment.Estimate)))
      (setq labor (+ labor (get.value activity 'Labor.Estimate)))
      (setq fringes (+ fringes (get.value activity 'Fringe.Estimate)))
      (setq material (+ material (get.value activity 'Material.Estimate)))
      (setq mh (+ mh (get.value activity 'MH.Estimate)))
      (put.value thisunit 'Project.Estimate (total equipment)
          (put.value thisunit 'Project.Labor.Total labor)
          (put.value thisunit 'Project.Fringe.Total fringes)
          (put.value thisunit 'Project.Material.Total material)
          (put.value thisunit 'Project.MH.Total mh)
          (put.value thisunit 'Project.Total.Estimate
                        (+ equipment labor fringes material))))
;;; INITIALIZE-PROJECT-FN is the value of the Initialize.Project slot of a project. It initializes the network, collects the activities that comprise the project and calculates all the project's initial and normal factors.

(defun INITIALIZE-PROJECT-FN (thisunit)
  (unimsg 'Notice.to.Proceed 'Initialize.Project)
  (unimsg 'Notice.to.Proceed 'Notify.Successors)
  (unimsg 'Notice.to.Proceed 'Calculate.Network)
  (put.value thisunit 'Project.Activities (COLLECT-PROJECT-ACTIVITIES))
  (unimsg thisunit 'Calculate.Project.Estimate)
  (unimsg thisunit 'Calculate.Project.Duration)
  (put.value thisunit 'Project.Normal.Cost (get.value thisunit 'Project.Total.Estimate))
  (put.value thisunit 'Project.Normal.Duration (get.value thisunit 'Project.Duration))
  (unimsg thisunit 'Calculate.Added.Cost)
  (unimsg thisunit 'CalculateReduced.Duration)
  (unimsg thisunit 'Determine.Next.to.Crash)
  (unimsg thisunit 'Get.Activity.Data))

;;; CALCULATE-ADDED-COST-FN is the value of the Calculate.Added.Cost of a project. It determines the difference between the current cost of the project and its normal cost.

(defun CALCULATE-ADDED-COST-FN (thisunit)
  (let ((currentCost (get.value thisunit 'Project.Total.Estimate))
        (normalCost (get.value thisunit 'Project.Normal.Cost)))
    (put.value thisunit 'Added.Cost (- currentCost normalCost))))

;;; CALCULATE-REDUCED-DURATION-FN is the value of the Calculate.Reduced.Duration slot of a project. It determines the difference between the current project duration and its normal duration.

(defun CALCULATE-REDUCED-DURATION-FN (thisunit)
  (let ((currentDuration (get.value thisunit 'Project.Duration))
        (normalDuration (get.value thisunit 'Project.Normal.Duration)))
    (put.value thisunit 'Reduced.Duration (- currentDuration normalDuration))))

;;; UPDATE-PROJECT-FN is the value of the Update.Project slot of a project. It updates the project's estimate, its duration, its added cost and its reduced duration slots.

(defun UPDATE-PROJECT-FN (thisunit)
  (unimsg thisunit 'Calculate.Project.Estimate)
  (unimsg thisunit 'Calculate.Project.Duration)
  (unimsg thisunit 'Calculate.Added.Cost)
  (unimsg thisunit 'CalculateReduced.Duration))
(defun DETERMINE-NEXT-TO-CRASH-FN (thisunit)
  (let ((activities (get.value thisunit 'Project.Activities)))
    (setq nextActivity 'Notice.to.Proceed)
    (dolist (activity activities)
      (let* ((costSlope (abs (get.value activity 'Cost.Slope)))
             (criticality (get.value activity 'Critical))
             (owner (get.value activity 'Responsible.Participant))
             (dayValue (get.value owner 'Day.Value))
             (duration (get.value activity 'Current.Duration)))
        (cond ((and (eql criticality 'True)
                        (>= duration 1)
                        (<= costSlope dayValue))
                (cond ((< costSlope duration)
                        (setq nextActivity activity)
                        (t)))
                (t))))
  (put.value thisunit 'Next.to.Crash nextActivity))

;;; CRASH-ACTIVITY-FN is the value of the Crash.Activity slot of
;;; a project. It crashes a single activity - its next activity -
;;; recalculates the network, updates its own cost and duration
;;; information and selects the next activity to crash. That is,
;;; it performs the duration reduction one step at a time.

(defun CRASH-ACTIVITY-FN (thisunit)
  (let ((activity (get.value thisunit 'Next.to.Crash))
        (unitmsg activity 'Change.Current.Tactic)
        (unitmsg thisunit 'Update.Project)
        (unitmsg thisunit 'Determine.Next.to.Crash)
        (unitmsg thisunit 'Get.Activity.Data))

;;; CRASH-PROJECT-FN is the value of the Crash.Project slot of a
;;; project. It applies the stepwise activity crashing until there
;;; are no further crashable activities.

(defun CRASH-PROJECT-FN (thisunit)
  (let ((nextActivity (get.value thisunit 'Next.to.Crash))
        (cond ((= nextActivity (unit 'Notice.to.Proceed)) nil)
              (t (unitmsg thisunit 'Crash.Activity)
                  (unitmsg thisunit 'Crash.Project))))

  (put.value thisunit 'Next.to.Crash nextActivity))
;;; GET-ACTIVITY-DATA-FN is the value of the Get.Activity.Data slot
;;; of a project. It simply gets the current tactic of the project's
;;; next activity to crash.

(defun GET-ACTIVITY-DATA-FN (thisunit)
  (let* ((activity (get.value thisunit 'Next.to.Crash))
         (tactic (get.value activity 'Current.Tactic))
         (duration (get.value activity 'Current.Duration))
         (costSlope (get.value activity 'Cost.Slope)))
    (put.value thisunit 'Next.Activity.Tactic tactic)
    (put.value thisunit 'Next.Activity.Duration duration)
    (put.value thisunit 'Next.Activity.Cost.Slope costSlope)))
;; **- Syntax: Common-lisp; Package: K; Mode: LISP **-
(in-package 'kee)

;;; This file contains the functions needed by the Tactics units
;;; in the JANUS knowledge base in order to determine their own
;;; characteristics.

;;; CALCULATE-EFFECTIVE-HOURS-FN is the value of the Calculate.
;;; Effective.Hours slot of a tactic. It determines the effective
;;; number of crew hours a tactic accomplishes and puts this value
;;; into the Hours.Effective slot of the tactic.
(defun CALCULATE-EFFECTIVE-HOURS-FN (thisunit)
  (let ((hrsWorked (get.value thisunit 'Hours.Worked))
        (efficiency (get.value thisunit 'Efficiency)))
    (put.value thisunit 'Hours.Effective
      (* hrsWorked efficiency))))

;;; CALCULATE-HOURS-WORKED-FN is the value of the Calculate.Hours.
;;; Worked slot of a tactic. It determines the number of hours worked
;;; using the tactic: the sum of the straight-time, 1.5-time and
;;; double-time hours less the shift premium.
(defun CALCULATE-HOURS-WORKED-FN (thisunit)
  (let ((straightTime (get.value thisunit 'Hours.Straight))
        (timeAndAHalf (get.value thisunit 'Hours.1.5))
        (doubleTime (get.value thisunit 'Hours.2))
        (shiftPremium (get.value thisunit 'Hours.Shift.Premium)))
    (put.value thisunit 'Hours.Worked
      (+
        (- straightTime shiftPremium)
        timeAndAHalf
        doubleTime))))

;;; CALCULATE-HOURS-PAID-FN is the value of the Calculate.Hours.Paid
;;; slot of a tactic. It determines the equivalent number of straight
;;; time hours paid for a day's work using this tactic.
(defun CALCULATE-HOURS-PAID-FN (thisunit)
  (let ((straightTime (get.value thisunit 'Hours.Straight))
        (timeAndAHalf (get.value thisunit 'Hours.1.5))
        (doubleTime (get.value thisunit 'Hours.2))
        (overtimePremium (get.value thisunit 'Hours.OTPremium)))
    (put.value thisunit 'Hours.Paid
      (+ straightTime timeAndAHalf doubleTime overtimePremium))))
;;; CALCULATE-SHIFT-FACTORS-FN is the value of the Calculate.Shift.Factors
;;; slot of a tactic. It determines the number of straight time hours
;;; performed by the tactic and its shift premium, based on the number
;;; of shifts the tactic uses in a day.

(defun CALCULATE-SHIFT-FACTORS-FN (thisunit)
  (let ((numberOfShifts (get.value thisunit 'Shifts.per.Day)))
    (cond ((= numberOfShifts 1) (put.value thisunit 'Hours.Straight 0)
           (put.value thisunit 'Hours.Shift.Premium 0))
          ((= numberOfShifts 2) (put.value thisunit 'Hours.Straight 16)
           (put.value thisunit 'Hours.Shift.Premium 0.5))
          (t (put.value thisunit 'Hours.Straight 24)
             (put.value thisunit 'Hours.Shift.Premium 1.5))))))
;;; CALCULATE-DUOUBLE-TIME-FN is the value of the Calculate.Double.Time
;;; slot of a tactic. It determines the equivalent number of 5-day-week
;;; double-time hours for the tactic.

(defun CALCULATE-DUOUBLE-TIME-FN (thisunit)
  (let ((daysPerWeek (get.value thisunit 'Days.per.Week))
        (hrsPerShift (get.value thisunit 'Hours.per.Shift)))
    (cond ((= daysPerWeek 5) ;Monday - Friday
           (cond (<= hrsPerShift 10) ;Nothing for time <= 10 hr M-F
                  (put.value thisunit 'Hours.2 0))
                  (t          ;Double time for time over 10 hr M-F
                  (put.value thisunit 'Hours.2
                                   (- hrsPerShift 10))))
        ((= daysPerWeek 6) ;Saturday
         (cond (<= hrsPerShift 8) ;Nothing for time <= 8 hr on Saturday
                (put.value thisunit 'Hours.2 0))
                (<= hrsPerShift 10)
                (put.value thisunit 'Hours.2
                                   (+
                                        (- hrsPerShift 8) ;Saturday's time over 8 hr...
                                        5)))))         ;...prorated over M-F.
                (t
                (put.value thisunit 'Hours.2
                                   (+
                                        (- hrsPerShift 10) ;M-F's time over 10 hr...
                                        (+
                                             (- hrsPerShift 8);...plus Saturday's time...
                                             5)))))) ;...over 8 hours, prorated.
            (t ;Sunday
            (cond (<= hrsPerShift 8) ;Double time all day Su, prorated.
                  (put.value thisunit 'Hours.2
                                   (+
                                        (- hrsPerShift 5) ;Sunday's time all day...
                                        (+
                                             (- hrsPerShift 8);...plus Saturday's time...
                                             5))))) ;...prorated.
                                (t
                                (put.value thisunit 'Hours.2
                                   (+
                                        (- hrsPerShift 5) ;Sunday's time all day...
                                        (+
                                             (- hrsPerShift 8);...over 8 hours, both...
                                             5) ;...prorated plus M-F's...
                                             (- hrsPerShift 10)))))))));...time > 10 hr.
;;; CALCULATE-TIME-AND-A-HALF-FN is the value of the Calculate.Time.and. a-Half slot of a tactic. It determines the equivalent number of 5-day-week time-and-a-half hours for the tactic.

(defun CALCULATE-TIME-AND-A-HALF-FN (thisunit)
  (let ((daysPerWeek (get.value thisunit 'Days.per.Week))
         (hrsPerShift (get.value thisunit 'Hours.per.Shift)))
    (cond ((<= daysPerWeek 5) ; Monday - Friday
            (cond ((<= hrsPerShift 8) ; Nothing for time <= 8 hr M-F
                   (put.value thisunit 'Hours.1.5 0))
                  ((< hrsPerShift 10) ; M-F's time over 8 hr...
                     (put.value thisunit 'Hours.1.5 (- hrsPerShift 8)))
                  (t ; ...up to a maximum of 2 hours.
                     (put.value thisunit 'Hours.1.5 2))))
    (t ; Saturday and Sunday get the same treatment
     (cond ((<= hrsPerShift 8) ; Saturday's first 8 hours, prorated.
            (put.value thisunit 'Hours.1.5 (/ hrsPerShift 5))
            (cond ((< hrsPerShift 10)
                   (put.value thisunit 'Hours.1.5 (+ ; Saturday's first 8 hours...
                     (/ 8 5) ; ...prorated, plus M-F's...
                     (- hrsPerShift 8))) ; ...time over 8 hours.
                   (t ; Saturday's first 8 hours...
                     (put.value thisunit 'Hours.1.5 (+ ; ...prorated, plus M-F's...
                       (/ 8 5)
                       2))))))))))

;;; CALCULATE-OVERTIME-PREMIUM-FN is the value of the Calculate. Overtime.Premium slot of a tactic. It determines the equivalent number of straight time hours represented by the overtime hours worked for the tactic.

(defun CALCULATE-OVERTIME-PREMIUM-FN (thisunit)
  (let ((timeAndAHalf (get.value thisunit 'Hours.1.5))
         (doubleTime (get.value thisunit 'Hours.OT.Premium))
         (+ (* timeAndAHalf .5)
            doubleTime))))
;;; CALCULATE-COST-PREMIUM-FN is the value of the Calculate.Cost.
;;; Premium slot of a tactic. It determines the cost premium of the
;;; tactic with respect to direct labor cost only. The cost premium
;;; is expressed as a percentage of the hours required to perform the
;;; activity on a straight time only basis. It takes into account
;;; the costs of inefficiency and of overtime and shift premiums.

(defun CALCULATE-COST-PREMIUM-FN (thisunit)
  (let ((hoursPaid (get.value thisunit 'Hours.Paid))
        (effectiveHours (get.value thisunit 'Hours.Effective)))
    (cond ((eql effectiveHours 0.0)
           (put.value thisunit 'Cost.Premium
                        (1- hoursPaid)))
          (t (put.value thisunit 'Cost.Premium
                          (1- (/ hoursPaid effectiveHours)))))))

;;; CALCULATE-DURATION-IMPROVEMENT-FN is the value of the Calculate.
;;; Duration.Improvement slot of a tactic. It determines the duration
;;; improvement of the tactic with respect to straight time only work.

(defun CALCULATE-DURATION-IMPROVEMENT-FN (thisunit)
  (let ((effectiveHours (get.value thisunit 'Hours.Effective)))
    (put.value thisunit 'Duration.Improvement
               (/ (- effectiveHours 8)
                   8))))

;;; CALCULATE-COST-SLOPE-MULTIPLIER-FN is the value of the Calculate.
;;; Cost.Slope.Multiplier slot of a tactic. It determines the cost
;;; slope multiplier of the tactic, based on direct labor costs only.
;;; This factor is used to order selection of tactics to use. Another
;;; factor - based on direct labor, labor fringes and equipment
;;; costs - is calculated for each activity for the actual time-cost
;;; tradeoff calculations.

(defun CALCULATE-COST-SLOPE-MULTIPLIER-FN (thisunit)
  (let ((costPremium (get.value thisunit 'Cost.Premium))
        (durationImprovement (get.value thisunit 'Duration.Improvement)))
    (cond ((eql durationImprovement 0.0) 999999)
          (t (put.value thisunit 'Cost.Slope.Multiplier
                          (/
                           (*
                            costPremium
                            (1+ durationImprovement))
                           (-
                            0
                            durationImprovement))))))))
;;; INITIALIZE-TACTIC-FACTORS-FN is the value of the Initialize.
;;; Tactic.Factors slot of a tactic. It must be run whenever a new
;;; tactic is defined and whenever any of the following slots of
;;; a tactic are changed:
;;;   - Straight.Hours
;;;   - 1.5x.Hours
;;;   - 2x.Hours
;;;   - Efficiency.

(defun INITIALIZE-TACTIC-FACTORS-FN (thisunit)
  (unitmag thisunit 'Calculate.Shift.Factors)
  (unitmag thisunit 'Calculate.Time.and.a.Half)
  (unitmag thisunit 'Calculate.Double.Time)
  (unitmag thisunit 'Calculate.Overtime.Premium)
  (unitmag thisunit 'Calculate.Hours.Worked)
  (unitmag thisunit 'Calculate.Hours.Paid)
  (unitmag thisunit 'Calculate.Effective.Hours)
  (unitmag thisunit 'Calculate.Cost.Premium)
  (unitmag thisunit 'Calculate.Duration.Improvement)
  (unitmag thisunit 'Calculate.Cost.Slope.Multiplier))
Appendix 3. *Demons*: Active values used in JANUS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALC.COST.SLOPE.AV</td>
<td>74</td>
</tr>
<tr>
<td>CALC.DAY.VALUE.AV</td>
<td>74</td>
</tr>
<tr>
<td>CALC.DURATION.AV</td>
<td>74</td>
</tr>
<tr>
<td>CALC_ESTIMATE.AV</td>
<td>74</td>
</tr>
<tr>
<td>CALC.NETWORK.AV</td>
<td>75</td>
</tr>
<tr>
<td>CALC.NEXT_ESTIMATE.AV</td>
<td>75</td>
</tr>
<tr>
<td>CALC.PROJECT.DURATION.AV</td>
<td>75</td>
</tr>
<tr>
<td>CALC.PROJECT_ESTIMATE.AV</td>
<td>75</td>
</tr>
<tr>
<td>EMPLOYER.AV</td>
<td>76</td>
</tr>
<tr>
<td>TACTIC.COST.PREMIUM.AV</td>
<td>76</td>
</tr>
<tr>
<td>TACTIC.COST.SLOPE.MULTIPLIER.AV</td>
<td>76</td>
</tr>
<tr>
<td>TACTIC.DURATION.IMPROVEMENT.AV</td>
<td>77</td>
</tr>
<tr>
<td>TACTIC.EFFECTIVE.HOURS.AV</td>
<td>77</td>
</tr>
<tr>
<td>TACTIC.HOURS.PAID.AV</td>
<td>77</td>
</tr>
<tr>
<td>TACTIC.HOURS.WORKED.AV</td>
<td>78</td>
</tr>
<tr>
<td>TACTIC.OT.PREMIUM.AV</td>
<td>78</td>
</tr>
<tr>
<td>TACTIC.SHIFT.FACTORS.AV</td>
<td>78</td>
</tr>
</tbody>
</table>
Unit: CALC.COST.SLOPE.AV
;Attached to: Current.Cost in ACTIVITIES
              Current.Cost.per.Day in ACTIVITIES

Superclasses:ACTIVEVALUE
Member of:     CLASSES

Member slot:  AVPUT
Inheritance:  METHOD
Valueclass:   METHOD
Values:       CALC-COST-SLOPE-AVPUT-FN

Unit: CALC.DAY.VALUE.AV ;Attached to: Employed.by in PARTICIPANTS
                          Contract in PARTICIPANTS

Superclasses:ACTIVEVALUE
Member of:     ACTIVEVALUE CLASSES

Member slot:  AVPUT
Inheritance:  METHOD
Valueclass:   METHOD
Values:       CALC-DAY-VALUE-AVPUT-FN

Unit: CALC.DURATION.AV ;Attached to: Current.Cost in ACTIVITIES
                          Current.Crew in ACTIVITIES
                          Current.Tactic in ACTIVITIES

Superclasses:ACTIVEVALUE
Member of:     CLASSES

Member slot:  AVPUT
Inheritance:  METHOD
Valueclass:   METHOD
Values:       CALC-DURATION-AVPUT-FN

Unit: CALC.ESTIMATE.AV ;Attached to: Current.Crew in ACTIVITIES
                          Current.Tactic in ACTIVITIES
                          Material.Estimate in ACTIVITIES
                          MH.Estimate in ACTIVITIES

Superclasses:ACTIVEVALUE
Member of:     CLASSES

Member slot:  AVPUT
Inheritance:  METHOD
Valueclass:   METHOD
Values:       CALC-ESTIMATE-AVPUT-FN
Unit: CALC.NETWORK.AV ;Attached to: Current.Duration in ACTIVITIES

  Superclasses: ACTIVEVALUE
  Member of: CLASSES
  Member slot: AVPUT
  Inheritance: METHOD
  Valueclass: METHOD
  Values: CALC-NETWORK-AVPUT-FN

Unit: CALC.NEXT_ESTIMATE.AV
;Attached to: Next.Tactic in ACTIVITIES

  Superclasses: ACTIVEVALUE
  Member of: CLASSES
  Member slot: AVPUT
  Inheritance: METHOD
  Valueclass: METHOD
  Values: CALC-NEXT-ESTIMATE-AVPUT-FN

Unit: CALC.PROJECT.DURATION.AV
;Attached to: Current.Duration in ACTIVITIES

  Superclasses: ACTIVEVALUE
  Member of: CLASSES
  Member slot: AVPUT
  Inheritance: METHOD
  Valueclass: METHOD
  Values: CALC-PROJECT-DURATION-AVPUT-FN

Unit: CALC.PROJECT_ESTIMATE.AV
;Attached to: Current.Cost in ACTIVITIES

  Superclasses: ACTIVEVALUE
  Member of: CLASSES
  Member slot: AVPUT
  Inheritance: METHOD
  Valueclass: METHOD
  Values: CALC-PROJECT-ESTIMATE-AVPUT-FN
Unit: EMPLOYER.AV
;Attached to: Employs in PARTICIPANTS

Superclasses: ACTIVEVALUE
Member of: ACTIVEVALUE CLASSES

Member slot: AVPUT
Inheritance: METHOD
Valueclass: METHOD
Values: EMPLOYER-AVPUT-FN

Unit: TACTIC.COST.PREMIUM.AV
;Attached to: Hours.Effective in TACTICS
; Hours.Paid in TACTICS

Unit Comment: The active value unit to insure that each tactic's
Hours.per.Day slot is correctly updated whenever a new
tactic is added or an existing tactic's Efficiency,
Shifts.per.Day or Hours.per.Shift slots change.

Superclasses: ACTIVEVALUE
Member of: CLASSES

Member slot: AVPUT
Inheritance: METHOD
Valueclass: METHOD
Values: TACTIC-COST-PREMIUM-AVPUT-FN

Unit: TACTIC.COST.SLOPE.MULTIPLIER.AV
;Attached to: Cost.Premium in TACTICS
; Duration.Improvement in TACTICS

Unit Comment: The active value unit to insure that each tactic's
Hours.per.Day slot is correctly updated whenever a new
tactic is added or an existing tactic's Efficiency,
Shifts.per.Day or Hours.per.Shift slots change.

Superclasses: ACTIVEVALUE
Member of: CLASSES

Member slot: AVPUT
Inheritance: METHOD
Valueclass: METHOD
Values: TACTIC-COST-SLOPE-MULTIPLIER-AVPUT-FN
Unit: TACTIC.DURATION.IMPROVEMENT.AV
;Attached to: Hours.Effective in TACTICS

Unit Comment: The active value unit to insure that each tactic's
Hours.per.Day slot is correctly updated whenever a new
tactic is added or an existing tactic's Efficiency,
Shifts.per.Day or Hours.per.Shift slots change.

Superclasses: ACTIVEVALUE
Member of: CLASSES

Member slot: AVPUT
Inheritance: METHOD
Valueclass: METHOD
Values: TACTIC-DURATION-IMPROVEMENT-AVPUT-FN

Unit: TACTIC.EFFECTIVE.HOURS.AV
;Attached to: Efficiency in TACTICS
; Hours.Worked in TACTICS

Unit Comment: The active value unit to insure that each tactic's
Hours.per.Day slot is correctly updated whenever a new
tactic is added or an existing tactic's Efficiency,
Shifts.per.Day or Hours.per.Shift slots change.

Superclasses: ACTIVEVALUE
Member of: CLASSES

Member slot: AVPUT
Inheritance: METHOD
Valueclass: METHOD
Values: TACTIC-EFFECTIVE-HOURS-AVPUT-FN

Unit: TACTIC.HOURS.PAID.AV
;Attached to: Hours.OT.Premium in TACTICS
; Hours.Straight in TACTICS

Unit Comment: The active value unit to insure that each tactic's
Hours.per.Day slot is correctly updated whenever a new
tactic is added or an existing tactic's Efficiency,
Shifts.per.Day or Hours.per.Shift slots change.

Superclasses: ACTIVEVALUE
Member of: CLASSES

Member slot: AVPUT
Inheritance: METHOD
Valueclass: METHOD
Values: TACTIC-HOURS-PAID-AVPUT-FN
Unit: TACTIC.HOURS.WORKED.AV
;Attached to: Hours.Straight in TACTICS

Unit Comment:The active value unit to insure that each tactic's
Hours.per.Day slot is correctly updated whenever a new
tactic is added or an existing tactic's Efficiency,
Shifts.per.Day or Hours.per.Shift slots change.

Superclasses:ACTIVEVALUE
Member of: CLASSES

Member slot: AVPUT
Inheritance: METHOD
Valueclass: METHOD
Values: TACTIC-HOURS-WORKED-AVPUT-FN

Unit: TACTIC.OT.PREMIUM.AV
;Attached to: Hours.1.5 in TACTICS

Unit Comment:The active value unit to insure that each tactic's
Hours.per.Day slot is correctly updated whenever a new
tactic is added or an existing tactic's Efficiency,
Shifts.per.Day or Hours.per.Shift slots change.

Superclasses:ACTIVEVALUE
Member of: CLASSES

Member slot: AVPUT
Inheritance: METHOD
Valueclass: METHOD
Values: TACTIC-OT-PREMIUM-AVPUT-FN

Unit: TACTIC.SHIFT.FACTORS.AV
;Attached to: Shifts.per.Day in TACTICS

Unit Comment:The active value unit to insure that each tactic's
Hours.per.Day slot is correctly updated whenever a new
tactic is added or an existing tactic's Efficiency,
Shifts.per.Day or Hours.per.Shift slots change.

Superclasses:ACTIVEVALUE
Member of: CLASSES

Member slot: AVPUT
Inheritance: METHOD
Valueclass: METHOD
Values: TACTIC-SHIFT-FACTORS-AVPUT-FN