Simulation of Information Flow in Design

by

Andrew Dean Christian

B.S.E. (M.E.) University of Michigan, Ann Arbor
B.S.E. (E.S.) University of Michigan, Ann Arbor
(1987)

M.S.E. (M.E.) Massachusetts Institute of Technology
(1989)

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Mechanical Engineering

at the
Massachusetts Institute of Technology
September 1995

Signature of Author_____

Department of Mechanical Engineering
August 29, 1995

Certified by_____

Warren P. Seering
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by_____

Ain A. Sonin
Chairman, Departmental Graduate Committee

Simulation of Information Flow in Design

by

Andrew Dean Christian

Submitted to the Department of Mechanical Engineering on August 11, 1995 in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Mechanical Engineering.

Abstract

Efficient communication between engineering team members is essential to project success. Engineers must coordinate their activities, exchange timely information, gather consensus, and participate in making group decisions. As a part of this research, an information-flow model of the engineering design process was created. The model explicitly represents the complexity of communication between interdependent design tasks. It has been implemented in a computer simulation that can be used to predict project performance.

The information-flow model specifies work and communication requirements for each task in a design project. Users of the model must specify three types of information: tasks, dependencies, and roles. Tasks are components of the project that represent work to be accomplished by an individual or through group discussion. Dependencies specify the need for information transfer between tasks. Roles specify responsibilities of individuals for each task.

In the computer simulation, individuals are represented by computer agents who work in a virtual office environment. The agents work on assigned tasks, engage in conversations, exchange information, and schedule and attend meetings. The project model, office layout, communication equipment, and agent behavior are specified by the experimenter.

The information-flow model and computer simulation have been validated against engineering projects conducted by design engineers. Results from a set of experimental methodologies addressing common management issues (e.g., task criticality, task interdependency, and team size) are also presented.

Thesis Committee: Professor Warren P. Seering, Chairperson
Professor Don Clausing
Professor Steven D. Eppinger

To Divya

Contents

List of Figures

List of Tables

List of Subroutines

Preface

Six years ago I cheerfully started the Ph.D. process at MIT, excited about studying new subjects and eager to switch from robotics to design research. The following years I deepened my understanding of mechanical engineering, mastered new aspects of computer science, and studied management theory. I also nearly quit several times out of shear frustration. Was it worth it? Ask me again in a few years.

The topic of this thesis, simulating the engineering design process, is about the last thesis topic I would have ever guessed that I would embark upon. It arose out of conversations my advisor, Warren Seering, and I had over the first three years of my Ph.D. as I took courses and alternately tried different thesis topics and approaches.

The impasse was broken the summer I spent visiting three companies that are participants in the Leaders for Manufacturing program. For three and half months I talked to engineers at Polaroid, Chrysler, and General Motors. Nearly a hundred engineers and managers spent a few hours of their time telling me about their jobs, how their company did design work, and what they thought of the process. It was a rewarding and useful summer in that it caused me to re-evaluate current methods of modeling the design process.

The result of the summer's activities was a renewed sense of purpose and clear aim in the modeling process. Above all, the engineers at the companies were concerned with the exchange of *information*. Everyone's job depended on communicating and coordinating with co-workers.

Armed with the idea of explicitly representing information exchange and group coordination, I returned to the drawing board and created a tentative method of modeling the design process. That model has been refined over the years by discussions with my committee and by the hardest master of them all: the computer. There is no method better suited to illuminating weak logic and faulty assumptions than the discipline of creating a computer program.

Has the model succeeded in capturing the design process as I observed it? Dozens of thorny problems and possible solutions come to my mind as I contemplate that question. Despite those reservations, I do believe that the model has merit. The reader is left to judge the accuracy of that statement for him/herself.

It has been my great pleasure to spend the Ph.D. years working at the MIT Artificial Intelligence Laboratory. Many people have participated in the thesis process, too many to list here. I would like to thank a few individuals: my committee, Don Clausing and Steven Eppinger, for their patience and advice over the past few years, Lukas Ruecker, who was always willing to debate computer programming, and Brian Avery, for listening to me carry on for hours and then coming up with good suggestions long after everyone else was asleep.

Finally, I'd like to thank my thesis advisor, Warren Seering, for supporting my interests all of these years and for generating enthusiasm and ideas. I'd also like to thank my parents and extended family, who have spent *far* longer than any of us would have guessed delicately avoiding asking the dreaded "so when will you finish?", and especially my wife, Divya, who has listened, patiently followed my thesis progress, and ensured that I didn't go *too* crazy.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. This material is based upon work supported under a National Science Foundation Graduate Fellowship and by the MIT Leaders for Manufacturing Program.

Chapter 1

Introduction

Efficient communication between engineering team members is essential to project success. Engineers must coordinate their activities, exchange timely information, and come to group consensus and decisions. This information exchange and coordination is fundamental to the success of the design and represents a large percentage of the time that engineers spend working on a project. For example, Hales [36] records that over 75% of the time spent on one particular design project was spent in meetings or exchanging information.

I have developed a method for modeling the design process that takes into account the complexity of communication between interdependent design tasks. The model specifies work and communication requirements for each task in a design project, the nature of the dependencies between the tasks, and roles of the individuals assigned to the tasks. To make predictions based on the information flow model, I have written a computer program that uses computer agents to represent the engineers. The computer agents interact with other agents within a virtual office environment. They exchange information, attend meetings, and work on their assigned tasks.

This thesis describes the information-flow model, the computer simulation, validation studies, and a sample of descriptive results. This chapter discusses two related bodies of research: methods of modeling the design process and methods of modeling organizations. The discussion of related research is intended to set the stage for the thesis model. Later on, specific modeling choices and related research will be discussed. This chapter concludes with a short overview of the thesis model and simulation.

1.1 Modeling

Two types of modeling methods are relevant to this thesis: design process models and organization models. Design process models specify the steps that should be taken in the design process. These steps can be either generic stages in the design process or concrete tasks to be accomplished in a particular design. Organizational models are models of what happens in a company. They focus on the interactions between individuals working at the company and how work is accomplished.

This thesis combines aspects of both design process models and organizational models. The next two sections describe relevant models and the approach selected for this thesis.

1.1.1 Design Process Models

The first problem encountered in this thesis was selecting an appropriate method of modeling the design process. There are two standard approaches: *descriptive* models and *precedence network* models.

Descriptive models specify the steps that should be undertaken in the design process. The Design Activity Model [65] is an example of a relatively simple descriptive model that shows the relationships between requirements, resources, and techniques employed to create a product. More recent textbooks on the design process specify the steps of the design process to be undertaken to insure high quality results (for example, Ullman [77] or Ulrich and Eppinger [78]). The most detailed example of *prescriptive* design I know of is Pahl and Beitz [59], who have systematically specified and organized the precise steps to be taken in the design process.

Descriptive models are well suited to leading engineers through the design process and helping them understand the process better. However, they are poorly suited for our goal of assisting managers working with small design teams. Descriptive models specify what should be done, not how effective different approaches will be.

Precedence network models of the design process do not prescribe the activities to be performed to accomplish the design, but rather take a set of tasks specified by the user and make predictions based on them. Precedence network models have three assumptions: First, the design project can be divided into a well-defined collection of tasks. Second, the time and resources required for each of these tasks can be estimated. Third, the tasks are partially ordered. That is, one can specify that certain tasks must occur before other tasks. These three assumptions allow a project planner/scheduler to reason about how long the design project will take and how many resources will be required. Note that the project planner/scheduler does not know the purpose of a task, but only how that task is ordered with respect to other tasks.

A variety of network models have been created. Each employs the standard three assumptions and augments them with assumptions appropriate for the desired purpose. Two of the oldest models are PERT and CPM [83]. The Program Evaluation and Review Technique (PERT) assumes probabilistic distributions of task duration. It calculates the probable duration of the project and can be used to locate tasks that fall on the *critical path*, which is defined to be the longest path through the network. The Critical Path Method (CPM) examines the tradeoff between overall project duration and the costs of individual tasks. The relationships between task cost and task duration and project cost and project duration are assumed to be known. The cost of reducing the duration of a single task is paid for by the money saved by reducing the duration of the project.

A variation of the precedence network approach was selected for the information-flow model presented in this thesis. The experimenter models a design process by specifying a set of tasks and dependencies between the tasks.

Interdependent Tasks

Both the PERT and CPM method assume that tasks are not interdependent. That is, dependencies cannot be arranged so that a sequence of tasks forms a loop (for example, $A \rightarrow B \rightarrow C \rightarrow A$). However, in the design process it is quite common to divide a project into a set of tasks and discover that the tasks are interdependent; that is, loops occur in the dependencies between tasks. It is also common in the design process to specify that two tasks may overlap in time; for example, $A \rightarrow B$, but B can begin once A is 50% complete. This overlap may simply require one task to lag another task by at least a certain amount, or it may reflect an underlying loop between the tasks.

Three approaches are available to represent interdependent and overlapping tasks using a network model. The first is to generalize the dependencies between tasks, but still require that a deterministic set of task start and stop times that satisfy the dependencies can be found. The second is to model dependency loops as iteration or rework. The third is to allow limited overlapping between tasks and then specify performance tradeoffs associated with this overlapping.

The first approach is to generalize the relationships between tasks. Generalized Precedence Relations (GPRs; see Elmaghraby and Kamburowski [24]) order two tasks by relations between task start times, finish times, and durations. For example, one can specify that task B can start as early as two days after task A, but must finish within five days of task A's finish and can

take no longer than 10 days total. A correctly formulated GPR model is deterministic because sets of start and stop times can be found that satisfy all of the dependencies.

The second approach to dependency loops is consider the network model to be a *flow* diagram (for example, Lesh [45]). Individual tasks in the network have functions that translate work done on a task into results and rework. Forward dependencies send the results to new tasks and allow them to start work, whereas backward dependencies represent rework channels that force additional work in completed tasks. The advantage of flow diagrams is that rework is logically incorporated into the project model. The disadvantage of flow diagrams is that the total amount of work required to accomplish a task is not readily apparent.

A variant on the flow diagram approach is to model the entire design process as a matrix iteration. Smith [70, 71] starts with a vector representing work to be done in a set of tasks. Each iteration in the design cycle is modeled by the multiplication of a matrix representing task interdependencies and the vector representing remaining work. The matrix is a variant of the Design Structure Matrix [25, 26, 27]. Eigenvalues and eigenvectors of the DSM then identify the slowest converging task loops.

The third approach to dependency loops specifies performance tradeoffs between overlapping tasks. This tradeoff can be as simple as specifying a function relating the size of the dependent task to how early it starts. Krishnan [39] uses a more complex formulation that requires rework in dependent tasks as a function of the progress made in the driving task. Once appropriate task start times have been selected, the total project duration can be calculated.

The three approaches solve the dependency loop problem in fundamentally different ways. The first refines the project dependency definition so that deterministic solutions may be found that satisfy the dependencies. The second approach views the network model as a flow of information and results, with dependency loops representing feedback. The third approach trades off relationships between overlapping tasks and their durations.

Selected Approach

After careful consideration of the available alternatives, I selected a variant of the first approach, generalized precedence relationships, to be used in the design model. This decision was ultimately motivated by the realization that selecting the first approach allowed us to address the other two approaches as well.

Dependencies between tasks in the design process normally represent a need for information. In fact, the timely exchange of information significantly affects design progress [5, 36]. A precedence approach to network models defines *when* that information is needed. A flow approach to network models can specifies *how* incorrect or faulty information results in changes in the product. This iteration is not a judgment of the quality of the initial work put out by a task; rather, it is a reflection of the fact that often in the design process an engineer must simply try an approach without being able to tell whether or not it will be optimal or even correct.

An advantage of the precedence network approach is that any flow model may be transformed into a dual precedence model. To do so, it is only necessary to calculate what work will be accomplished in what order and the total amount of work that is done in each task. These quantities can be transferred to a precedence model. The converse is not true because precedence networks specify a flexibility in when work on tasks may be accomplished; in other words there is *slack* available in a precedence network.

Another advantage of the precedence model is that it more closely matches the data recorded on a company's timesheets. In corporate project analysis, the size of a task in a precedence network can be easily estimated as the sum of all of the hours logged working on that task. On the other hand, the duration of tasks in the flow model is a function of the nominal size of the task and the strength of the feedback loop. Hence when working from recorded corporate data, it is difficult to estimate the size of task in a flow model.

Based on the perceived advantages and disadvantages of the flow and precedence models, we selected the precedence model as the foundation of our design model. Dependencies between tasks represent a need for information transfer from one task to the other. The dependencies

are generalized; they specify not only what information is needed, but when it is needed. Small iteration loops among a few tasks are represented by estimating total task durations (including necessary iteration) and specifying appropriate dependencies between the tasks. Large scale iteration loops (for example, multiple prototypes) are represented by unrolling the dependencies; that is, if you are planning on creating three prototypes, then each prototype is represented by a set of appropriate tasks and dependencies (rather than representing all prototype creating as a single set of tasks and counting on feedback to create three iterations).

The final problem is how to incorporate tradeoffs between task start times and task duration. To implement this in a precedence network requires specifying functions that relate the task duration to what types of information are available. This issue may be addressed without explicit incorporation into the precedence network by comparing the behavior of networks that vary the task sizes and precedence requirements.

1.1.2 Organizational Models

In conjunction with selecting an approach of modeling a design project I looked into methods of modeling the design *process* in an organizational context. A variety of types and approaches to organizational modeling exist in the literature. For the purpose of this thesis, I examined different models to look for ones that represented aspects we considered important: the coordination of teams of engineers working on tasks in a precedence network, the transfer of information between engineers, and the bounded rationality of human behavior. No models that met our requirements were found, but a large number of them contributed interesting ideas or approaches.

Types of Models

Organizational models can be loosely divided into those that represent individuals and those that do not. Organizational models that do not represent individuals are not set up to deal with the complex interdependencies of design projects, but they do often deal with coordination, personalities, and bounded rationality. For example, Abdel-Hamid's model of software project dynamics [2] is a system dynamics model [29] that explicitly deals with human resources, team experience, individual productivity, rework, and management controls. But it does not represent the nature and types of tasks to be accomplished. This is a limitation of the system dynamics approach of representing things as continuous streams of information; it is appropriate at macroscales in corporations, but at microscales the behavior of individuals becomes important.

Organization models that represent individuals have slowly appeared over the years mainly due to the presence of better and faster computer equipment. As computers have grown faster, models have grown more complex and tried to capture more realistic human behaviors.

In the early 70's Cohen [19] created the garbage can model of organizational choice. The garbage can model consists of a stream of problems and choices coming into an organization, and a set of agents that direct energy towards selecting solutions to those problems. Agent choices are based on interactions with other agents in the organization. The original simulation is quite simple; around this an entire body of literature developed [54].

Malone [52] and Crowston [20] have worked on the problem of coordination within an organization by modeling the effects of different management hierarchies. Lin [51] has looked into measures of organizational design. These researchers consider the effects of agent coordination structure, but not the behavior of the agents themselves.

A number of researchers have worked on applying distributed artificial intelligence to models of organizations. These generally take the form of having the separate agents coordinate their activities in order to accomplish either a common goal or to accomplish individual goals [80, 12, 79]. An advantage of this approach to organizational models is that it directly confronts the problem of human bounded rationality; the concept that humans do not have the mental capacity necessary to forecast precisely the outcome of their choices and hence select activities suboptimally [53, 68]. Carley [13] provides a nice description of two approaches to this problem; one dealing with agents working in a warehouse filling orders, and one examining team structure.

The model of organizational behavior that comes closest to the intent of this thesis is Cohen and Levitt's Virtual Design Team (VDT) [18, 17, 47, 48]. This is a model of teams of engineers working on a large project with managerial control. It represents the project as a collection of tasks arranged in a PERT-like network. Each task has an assigned team of engineers. Managers of the project come into play as exception handlers [30]; they must resolve problems that periodically occur in the working teams. The VDT provides a rich representation of communication structures and management hierarchy. However, it does not provide for the information flow between the teams working on the tasks in the project, nor does it provide for interconnected tasks or loops in the dependency chains.

Selected Approach

After considering the various approaches used in organizational models, I selected a distributed artificial intelligence approach with explicit information transfer between agents. Individuals in the organization are modeled by computer agents that select what to do and when to do it. They may work on tasks or transfer information to or from other agents in the simulation.

There are three advantages of using intelligent computer agents: First, it is easy to incorporate bounded rationality of humans. As long as the agents are restricted to selecting activities based upon what they could reasonably know, they are by nature bounded in their rationality. Second, the use of computer agents makes it simple to account for communication time; agents must transfer information before other agents may use it and this transference takes time. Third, the model is inherently expandable. As more complex behaviors are required only the agent's method of selecting actions needs to be changed.

1.2 Thesis Overview

This thesis consists of four parts: a method of modeling design projects as tasks interconnected by information requirements, a simulation of engineers working on a design project, corporate validation studies of the thesis model, and a demonstration of how the model and simulation may be used to help manage design projects.

The method of modeling design projects is referred to as the "information-flow model". This method is based on three ideas: First, that a design project can be modeled as a collection of tasks that represent work to be done either by an individual or by a team of engineers engaged in a group discussion. Second, that dependencies between these tasks represent a need for information. Parameters associated with a dependency specify *when* information transfer must occur and *how much* information is needed. Third, individual engineers are modeled as separate entities with private knowledge and the ability to do only one thing at a time. An engineer responsible for several tasks must decide what he/she will do at any given time: work on a task, participate in a group discussion, or exchange information with another engineer.

The simulation, *DiFS* or "Design Information-Flow Simulation", is a computer implementation of the information-flow model.[1] Engineers and managers are represented by computer agents working within a simulated office environment. They each have assigned offices to work in, tasks to work on, and communications equipment they use to communicate with other agents. During the simulation agents decide what to do: they work on tasks, discuss tasks with other agents, schedule and hold meetings, travel around the office, and write and receive messages. Agents select their actions based upon their private knowledge of the design project, knowledge of other agents, recent events, and personal work strategies. Figure 1.1 is a screen capture of the *DiFS* simulation software running a simple design project composed of 17 interdependent tasks and 5 agents working within a simulated office building.

The final chapters of this thesis discuss corporate validation studies and how the model and simulation may be used to assist a manager of a design project. The largest validation

[1] Why is the "i" in *DiFS* in lowercase? In the Macintosh OS, applications are identified by a unique registered four-letter signature. The signature of *DiFS* is "DiFS"; hence the spelling.

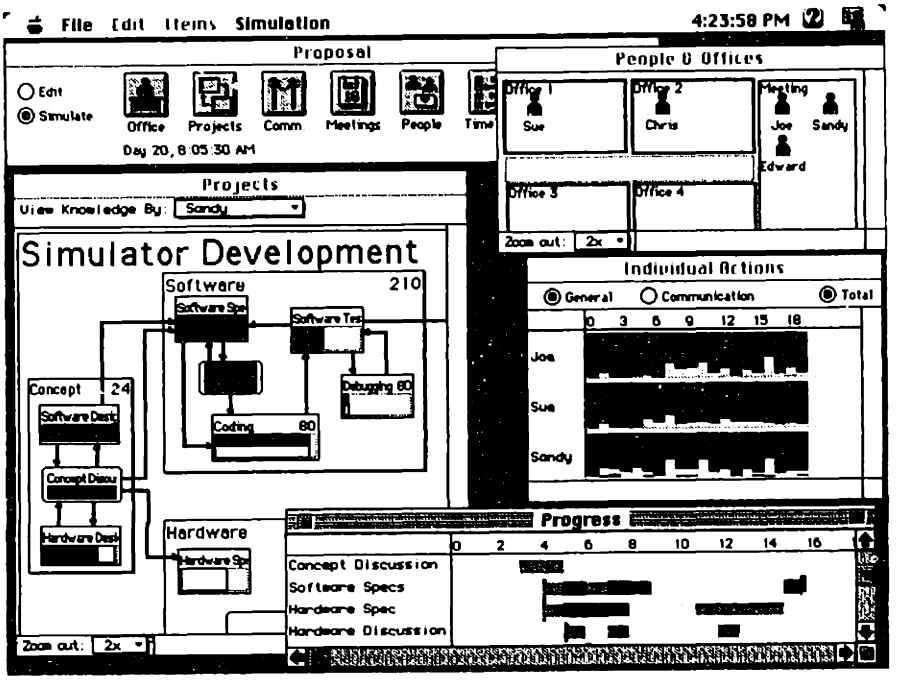


Figure 1.1: Screen capture of the *DiFS* simulation halfway through a design project.

study to date with the simulation was conducted at a small design firm. The activities of a team of engineers working on a six-month project were translated into a project model and compared with simulation results. Finally, experiments conducted with the model demonstrate how the simulation may be used to estimate critical paths through a design project, optimize resource allocation, and explore the effects of personality, team location, task topology, and communication methods.

The remainder of this thesis is divided into the following chapters:

Chapter 2: A description of the information-flow model of design projects and a discussion of advantages and disadvantages of this approach.

Chapter 3: A description of the implementation of the information-flow model as a computer simulation of an organization. Particular emphasis is given to how agents exchange information and work in groups.

Chapter 4: A discussion of the elements of the agent behavior model. Agents use their personal knowledge of task status, other agent behavior, recent events, and the local environment to select and carry out courses of action.

Chapter 5: Several examples of what happens in a simulation run, the types of behaviors observed, and the types of results that an experimenter can extract.

Chapter 6: Preliminary validation of the simulation and experimental techniques used to extract results.

Chapter 7: A discussion of the largest corporate validation study of the simulation.

Chapter 8: A demonstration of how the model and simulation may be used by managers seeking to understand or improve project performance.

Chapter 9: Thesis conclusions and a discussion of possible future work.

Chapter 2

Information-Flow Model

This chapter describes the model of the design process used in the thesis. The model consists of a network of tasks to be accomplished connected via dependencies that represent a need for information transfer. It will be referred to either as the information-flow model, or as the *DiFS* model.

The information-flow model was created after months of discussions with engineers and managers and reading of the research literature. My intent was to create a simple, well determined model of the design process that specifically accounts for the information transfer required between engineers. The scope of the model is appropriate for small groups of engineers working on design projects: on the order of 5 to 10 engineers working for 3 to 6 months. Larger projects are possible, but the level of detail required for modeling makes them tedious to construct.

This chapter is divided into three sections: a description of the model, a discussion of the implications of this modeling approach, and a summary of the model and its principle assumptions.

2.1 Representation

The information-flow model (*DiFS* model) represents the interdependent nature of the design process. This section briefly outlines the core assumptions and elements of the model, and then elaborates on each of the elements of the model in turn.

2.1.1 Core Assumptions

Before discussing the information-flow model representation, it is necessary to mention the core assumptions that the information-flow model is built upon so that the reader understands the advantages and limitations of this modeling approach. The purpose of this section is to clarify the basic model assumptions; section 2.3 discusses further assumptions and implications of the modeling approach.

The following are the basic assumptions of the information-flow model:

1. Both work and communication requirements are modeled as quantities of time that an individual or group must spend.

2. Work and communication times are specified by the experimenter setting up the model. Assignments for individual people are also fixed by the experimenter.

3. No attempt is made to capture the appropriateness of the modeled design process or the quality of the product generated.

The primary implication of these assumptions is that the particular *product* of the engineering design process is not represented. The information-flow model simply does not make any

predictions about the quality or appropriateness of the product being produced; the information-flow model concentrates on the time requirements of the modeled process. There is no representation for flaws in the design process or mistakes made while working on the project.

Because the quality of the product is not measured, design iteration in the information-flow model is limited to pre-specified iteration modeled by the experimenter, rather than iteration resulting from correcting mistakes. The difference between these two methods is that the information-flow model specifies the total work and communication times when the model is set up, whereas an iteration model specifies a minimum time and then calculates the total required time based on when activities occur and errors made while doing the activities. For example, consider a simple design task that normally takes 100 hours of work. The information-flow model will require 100 hours of work to finish this task. An simplistic iteration model might specify that 10% of the work a particular person does is invalid and hence must be redone. A person assigned to the nominally 100 hour task will actually take $100 + 10 + 1 + 0.1 + \ldots = 111.1$ hours to accomplish the task.

Focusing the information-flow model on explicit time representations of work and communication requirements simplifies the model and makes it easier to calculate what happens during the life of a design project. It is still possible to experiment with effects of design process and quality; however, the experimenter must decide for him/herself how these variations will affect the topology and time requirements of the model.

2.1.2 Model Overview

The information-flow model can be thought of as an augmented precedence network. A design project is assumed to be made up of a collection of well-defined tasks. Tasks are related to each other via dependencies that specify temporal relationships and required information transfer. People do work on the tasks they are assigned to.

Figure 2.1 shows a sample information-flow model that is based on an actual engineering project. This model represents a team of engineers working on a simple computer interface experiment. The purpose of the experiment is to test human reactions to data presented on a computer screen. It consists of four tasks: three individual and one group discussion. The tasks are: the DESIGN of the experiment, the programming of the INTERFACE SOFTWARE and data collection mechanism, an ANALYSIS PROGRAM that will interpret collected data, and a group discussion or REVIEW of the project. The first three tasks are to be done by individuals. The last task (REVIEW) is a group discussion, and hence is represented by a rectangle with rounded corners. The number in the upper-right corner of each task is how many hours of work are required to complete the task. Arrows connecting the tasks represent dependencies between the tasks.

There are two principle differences between an information-flow model and more common precedence networks such as PERT diagrams. First, in a PERT diagram each task is done by a team of one or more people. In the information-flow model tasks are divided into tasks that are done by a single person and tasks that are done by a group discussion. Second, a dependency in a PERT diagram represents a simple precedence relationship; that is, the upstream task must finish before the downstream task begins. In the information-flow model the dependencies represent a need for information transfer from the upstream task to the downstream task. The information to be transferred is generated as a result of progress on the upstream task, hence the dependency specifies both a precedence relationship and a required amount of time to be spent in communicating the information both on the part of the person supplying the information and on the part of the person receiving the information (unless they are the same person). Moreover, the *DiFS* model permits a flexible expression of when the information is required as a function of how complete the downstream task is.

Note that the dependencies in figure 2.1 form a loop among the four tasks. This loop may be thought of as a simple form of iteration. Each dependency has parameters that specify when the information generated by the upstream task is needed by the downstream task. In the case of

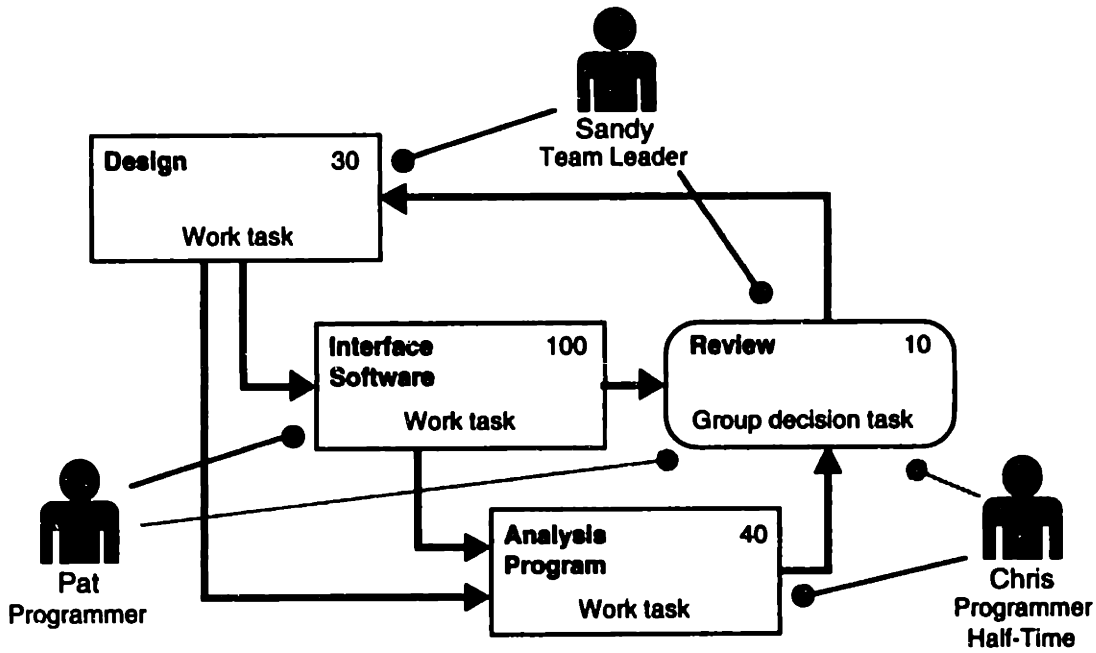


Figure 2.1: A sample project model representing the creation of a human-factors experiment. The number in the upper-right hand corner of each task is the total number of hours required to complete that task.

the human-factors project model, the DESIGN task cannot be completed until some preliminary information is available from the REVIEW task.

A qualitative version of what will happen in this project can be sketched: First, the team leader will work for a few days to establish the preliminary design. The information generated will be transferred to the other two individuals in the projects (this transference takes time). The individual responsible for INTERFACE SOFTWARE will commence working first, followed by the individual responsible for ANALYSIS PROGRAM. When sufficient work has been accomplished, all members of the team will meet to review progress to date (in the REVIEW task). This will result in additional work by the team leader on the DESIGN task. The design changes will be transferred to the other team members, and they will continue work on their assigned tasks. The review cycle will repeat itself a few times until all tasks are finally complete.

A visual inspection of figure 2.1 suggests that the overall project duration will be on the order of 140 hours, based on the apparent longest path through the iteration loop. Information transfer times, organizing people to meet to discuss the REVIEW task, and overlapping of task start times will affect that estimate. Parameters for tasks, people, and dependencies must be specified in order to calculate the duration of the overall project. These parameters are defined and discussed in sections 2.1.3 through 2.1.5.

A final comment: the project model of figure 2.1 is a model of what happened in the creation of one particular human-factors experiment, *not* a model of *how* human-factors experiments should be created. This project model is based on a real design project. In the design project, the individuals responsible for writing the software were inexperienced in graphical interfaces and human-factors experiments. Hence there was little discussion between the team members as to the contents of the design. This is reflected in the model; note that there is no direct flow of information from the person working on the INTERFACE SOFTWARE to the person creating the DESIGN. In an ideal human-factors experiment design, suggestions and advice from the programmers would be directly incorporated into the DESIGN.

2.1.3 Tasks

A task in the information-flow model represents a well defined piece of work to be accomplished. We assume that an accurate estimate of the task duration is available. We also assume that it is clear when a task has started and when it has terminated. These assumptions are comparable to PERT/CPM modeling assumptions.

The information-flow model defines two types of tasks: **work tasks** and **group decision tasks**. A work task is a piece of work to be accomplished by an individual. A group decision task is a piece of work to be accomplished by a number of engineers working in a team. Work tasks can represent a variety of things; for example, working on a drawing, doing engineering calculations, or contacting vendors. Group decision tasks represent a group of individuals who come together to resolve issues and make joint decisions.

To be precise, progress on a work task is made by having the individual assigned to the task spend time working *alone* on the task. Progress on a group decision task is made when the group of engineers assigned to the task meet in a common location and *simultaneously* specify that they are working on the task. Note that these definitions are different from that of a traditional PERT/CPM model, where a task is a piece of work to be done by an arbitrary number of individuals, but no mention is made of whether or not they must coordinate their activities.

Why are work tasks just for individuals? The objective of the *DiFS* model is to model information flow in design. Dependencies (discussed in section 2.1.5) between tasks control the frequency and duration of communications between engineers working on separate tasks. If we allow multiple engineers to be assigned to a single work task (and hence allow them to work separately on the same project), then we must specify how frequently the engineers coordinate their activities. Normally this coordination takes place by sharing information. Rather than define one coordination mechanism for inter-task information sharing and one for intra-task information sharing, I chose to restrict work tasks to a single person. All coordination methods between engineers occur because they must exchange information about tasks. The price of this restriction is that the size of each task in the information-flow model is generally smaller than tasks in the PERT/CPM model. For a given project size, the information-flow model will have a larger number of tasks than the PERT/CPM model and the average size of those tasks will be smaller than in the PERT/CPM model.

Unlike work tasks, decision tasks may have an arbitrary number of individuals assigned to them. The coordination of engineers is not a problem in a decision task because it is assumed that work can only be done on a decision task if all assigned people are working simultaneously in a joint discussion. That is, everyone is in the same location attending the same meeting.

Task Parameters

Work and decision tasks can be characterized by three parameters: **work time**, **communication time**, and **convergence curve**.

Work time is a measure of how long the task will take to complete. It is measured in nominal person-hours. That is, a work task with a work time of 20 person-hours would take a normal engineer 20 working hours to complete, assuming that engineer had all necessary information to complete the task and a suitable working environment. An extraordinary engineer might complete the same task in less time. In the case of a decision task, the work time represents the amount of time that the group *as a whole* needs to work on the task.

The communication time (or comm time) of a task is a measure of the information content of the task. It represents the quantity of information generated by the individual or individuals while working on the task. This information may be transferred to another person. Hence communication time is measured in nominal person-hours. A nominal person-hour is the quantity of information that can be transferred from one person to another in a face to face discussion that lasts an hour. Information content of a task is measured in nominal person-hours because it is easy to estimate and convenient when calculating project durations.

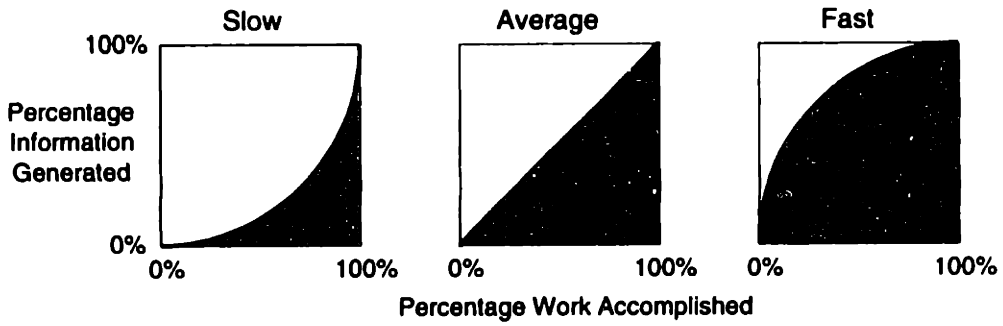


Figure 2.2: Sample information convergence curves.

The act of working on a task generates information, but that information is not necessarily generated at a uniform rate. For example, a finite element analysis task might spend a large portion of its time setting up the model and a small portion of its time generating the results of interest. On the other hand, a design task might generate a large quantity of information initially and then slowly fill in minor details.

The quantity of information generated by a partially completed task is a function of the **information convergence curve** of the task and the communication time of the task. The information convergence curve of a task is a function that translates work done on the task into the quantity of information generated by the task. The information convergence curve of a task can be represented by a non-dimensionalized monotonically increasing curve that maps from the hours spent working on the task (work time) to the hours of information generated in the task (comm time). A sample set of curves is shown in figure 2.2. Information convergence curves are normally expressed in non-dimensional form where they map from 0 to 100% of work time to 0 to 100% of communication time.

2.1.4 Individuals & Roles

In a corporation, work is done by individuals. In the information-flow model, work is done by individuals too. Each person on an engineering project is modeled as a separate entity in the *DiFS* model. Each person does one thing at a time; either working, communicating, or nothing at all. Each person is assigned responsibilities. These responsibilities are encoded via assigned **roles**.

Individuals

The advantage in modeling individuals rather than groups is that the information exchange model is simplified. Each individual keeps track of what he/she knows at any time. Task knowledge only changes when either (1) work is done on an assigned task, or (2) an individual acquires new information from another individual. Both of these activities take specified amounts of time. If teams were modeled, then somehow we would have to account for the amount of time it takes information to diffuse across the team and the amount of time that it takes team members to coordinate their activities.

The *DiFS* model also assumes that individuals only do *one thing at a time*. That is, at any given moment an individual is either (1) working on a project, (2) communicating with another individual, or (3) doing nothing. An alternate system would be to calculate what percentage of time an individual was spending on each activity. The advantage of the single activity model is that it logically matches what happens in the real world; if four individuals wish to hold a meeting, then all four are unavailable for work on other tasks or communication with other engineers. The disadvantages of this approach are that it requires more bookkeeping to track what individuals are doing and that it requires coordinating the activities of multiple

individuals. These problems could be avoided by simply counting the number of hours per day spent on each type of activity, but then the ordering of actions based on new knowledge generated or acquired would be problematic.

Roles

Roles connect people to tasks. There are exactly two types of roles currently supported in the *DiFS* model: **responsible** and **assistant**. Each task must have exactly one individual who is **responsible** for it. Work tasks are always done by a single person, hence the person who is assigned to the work task must be the responsible person. Decision tasks may have a group of individuals assigned to them; in this case, exactly one person of the group is assigned the **responsible** role and the remainder are assigned **assistant** roles. The responsible person for a decision task is considered to be the discussion leader. Generally this would be the person of highest rank or greatest authority who leads the discussion and has the final say in disagreements or choices to be made by the group. However, all team members are assumed to jointly participate in making group decisions.

In practice there are a large number of other possible roles that people can play on a task (for example, **reviewer**, **consultant**, **manager**, **liaison**). Implementing these roles requires clearly specifying how much participation and knowledge is required for the person assigned that role and how their participation affects the progress of a task. The current version of the software does not implement these roles (except for some managerial aspects). For a discussion of role extensions, refer to section 9.2.2.

People Parameters

The experimenter specifies parameters for each individual that control how efficiently he/she works and communicates and what percentage of the working day is available for working on projects. These parameters are: **working efficiency**, communication efficiency (or **comm efficiency**), and **available time**.

Working efficiency (work efficiency) is how effectively an individual works with respect to the nominal working rate. Work tasks are characterized by a work time that is the nominal number of hours of work required to complete the task. Depending on his/her work efficiency, an individual may finish the task in more or less time. Work efficiency is expressed as a percentage of nominal. For example, a work efficiency of 200% is twice as fast as the nominal rate of work. An individual with a work efficiency of 200% finishes a work task in half the time of an individual with a work efficiency of 100%. An individual with a work efficiency of 50% takes twice as along as the individual with a work efficiency of 100%. For decision tasks (where many individuals are involved), the controlling work efficiency is that of the responsible individual.

Communication efficiency (comm efficiency) is how effectively an individual communicates with respect to the nominal communicating rate. The information content of a work task (comm time) is stored as the nominal amount of time required for one person to communicate all of the information generated in the task to another person in a face to face discussion. The actual amount of discussion time required varies as a function of the comm efficiency of the person doing the talking. Comm efficiency is expressed as a percentage of nominal; a comm efficiency of 200% is twice as fast as nominal, whereas a comm efficiency of 50% is half as fast as nominal.

Decision tasks are done by groups of individuals holding a common discussion. Because they are held in a conversation format, both the work efficiency and the communication efficiency parameters of the responsible person are applied. For example, if the responsible person on a decision task has a work efficiency of 120% and a comm efficiency of 125%, then the overall efficiency of decision making is the product of the work and comm efficiencies, or 150%. In other words, the task will take two-thirds as long as its nominal time (a six hour task will take just four hours to complete).

People in the information-flow model do not necessarily work full time. They may be assigned to the project only part time. The third parameter of an individual, available time, measures

how much time per day an individual is available. It is expressed as a percentage of a normal nine-hour day. For example, a person with an available time of 50% works from 8 in the morning until half past noon. At 12:30 P.M. that person is unavailable for work or communication until the next day. Individuals with small available time come to work only once a week or less frequently (but always on Monday). The rule is that an individual always shows up for at least two hours; hence a person with an available time of 25% would work 2.25 hours every morning, whereas an individual with an available time of 5% works 2.25 hours every Monday morning. Note that during the hours the individual is available, he/she works and communicates as effectively as any other individual. People on partial schedules simply go home earlier and come in less frequently. This approach to part-time work eliminates the problem of defining how long communications take with people who are only available part of the time; the communications take as long as they ever did, but they may only occur during specific times of the day. The specific hours that individuals work are arranged so that periodically it is guaranteed that all individuals will be in the office at the same time (and hence can hold a meeting if so desired).

The disadvantages of using the part time approach are (1) normally individuals who work part time have some flexibility in scheduling when they work and (2) it does not represent individuals who go on vacation or are out of town occasionally on business. In the interests of keeping the model simple such flexibility was not incorporated, but it may be included in the future.

2.1.5 Dependencies

Work on a single task in a design project almost always depends on getting results from other tasks. In engineering design this dependency generally can be expressed as a need for information. Often the information required is not all of the information generated by the other task, nor does the first task have to be completed before the second may begin (i.e., the two tasks may overlap). Two tasks that both depend upon a third may also require different types of information from the third task. To reflect the varying types, quantities, and temporal requirements for information from one task to another, I introduce a method of modeling the specific information required by a dependency between tasks. Dependencies between tasks are represented by arrows, as seen in figure 2.1. The arrow points from the upstream or driving task to the downstream or dependent task. The person working on the downstream task will need to acquire some of the information generated in the upstream task from the person responsible for the upstream task. Dependency parameters set by the experimenter specify when this information is needed and how much of it is needed.

Dependency Parameters

The total quantity of information generated by a task is measured by its comm time. All or part of the information generated by the task is needed by tasks that depend upon it. Because downstream tasks may require different pieces of the information generated, the "slice" of the information required is modeled by two parameters: **scope** and **thoroughness**. *When* the downstream task need the information at differing times; this need for information is expressed by the **completion function**.

Scope represents the breadth of knowledge required. A dependency with narrow scope means that the downstream task only needs information generated from a relatively small portion of the upstream task. For example, assume the upstream task is the design of a chair and the downstream task is the ordering of the fabric to cover the chair. In this case, the dependency would have narrow scope because ordering the fabric only requires a narrow range of information from the design task. Conversely, if the downstream task was preparing a manufacturing facility for chairs, then a great breadth of knowledge about the chair design would be needed and hence the dependency would have a wide scope.

Thoroughness represents the depth of knowledge required. A dependency that required only an overview of the information generated by the upstream task is one that requires little

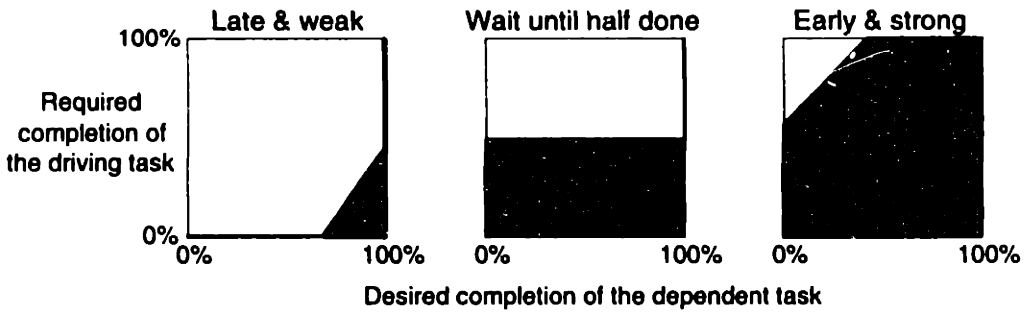


Figure 2.3: Sample completion functions. The white regions represent valid states, the grayed regions represent invalid states. The dark line is the border of the valid region.

thoroughness. For example, if the upstream task is the chair design and the downstream task is a brief managerial review of the task progress, then the dependency would specify an overview. On the other hand, the preparation of a manufacturing facility for chairs would probably require great thoroughness of knowledge about all details of the chair design.

The person who is responsible for a task and is creating the information always knows it at 100% scope and thoroughness. The **completion** of the task is the percentage of work in the task that has been accomplished. When a person transfers information about the task to another person, he/she specifies the subset of the scope, thoroughness, and completion knowledge to be sent.

The completion function of a dependency represents *when* information is needed. Two tasks may overlap by a greater or lesser extent depending upon how far the downstream task can proceed before acquiring information from the upstream task. The completion function maps the desired completion of the downstream task to the required completion of the upstream task. Sample completion functions are shown in figure 2.3.

Consider the completion functions shown in figure 2.3. Assume that the upstream or driving task is Task A and the downstream or dependent task is Task B. The first sample completion function says that Task B can work up to approximately 70% complete before acquiring any information from Task A. Before Task B can complete, the first 40% of the information generated in Task A must be acquired. Note that Task B can be finished well before Task A has finished. The second sample completion function, "Wait until half done" says that Task B cannot start until Task A is half finished. But Task B never requires any more information from task A; it can be finished without any additional work taking place in Task A. The third sample completion function, "Early & strong", says that Task B cannot start until Task A is 60% complete and that Task B cannot proceed past about 40% complete until Task A has finished. This sample completion function is close to being a serial dependency between the two tasks. A true serial dependency would be represented by a completely filled in completion function box.

A nice way to think about the three parameters of a dependency (scope, thoroughness, and completion function) is to think of the information required by the dependency being in the form of a rectangular solid (see figure 2.4). The scope and thoroughness axes are fixed by the scope and thoroughness parameters of the dependency. The completion axis is a function of the desired completion of the downstream task. This desired completion level is mapped through the dependency's completion function to give the required completion level of the information solid.

Group decision tasks present a special challenge to the information-flow model. A single individual working on a single task clearly must have acquired all of the proper knowledge in order to work on that task. But the knowledge requirements of a group of individuals are not as clear. There are three plausible alternatives: (1) all individuals assigned to the group must each satisfy the dependency requirements, (2) the aggregate knowledge of the individuals must satisfy the dependency requirements, or (3) the responsible person must satisfy the dependency

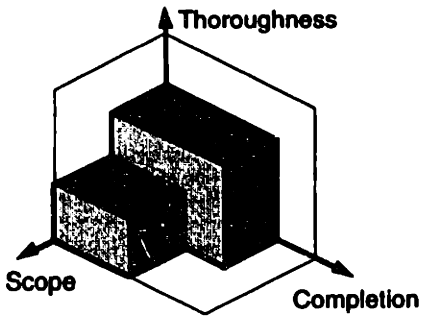


Figure 2.4: Information solid. This is a representation of a union of two pieces of information. The first is of scope 100%, thoroughness 40%, and completion 50%. The second is of scope 50%, thoroughness 66%, and completion 75%.

requirements. The *DiFS* model can be operated using any of these three options, but the third option is the one that is normally selected (i.e., only the responsible person must satisfy the dependencies). This decision was made based on two observations. First, most decision tasks represent a group of individuals getting together to resolve issues arising from coordinating their respective tasks. Hence the aggregate knowledge of the group normally already satisfies the dependency requirements. Second, by only specifying the knowledge requirements of the group leader, the bookkeeping required is simplified.

Dependency Calculations

The scope, thoroughness, and completion function specify the portion of the information generated in the upstream task that is needed to attain a particular completion level of the downstream task. The nominal amount of time it will take to communicate the required "chunk" of information is assumed to be a percentage of the comm time of the upstream task.

The quantity of information available from a given task at any point in time can be calculated based on the amount of work that has been done on the task. For a given task A, let W_A be the work hours, C_A be the comm hours, and $R_A(x)$ be the information convergence function, where $R_A(x) : [0,1] \rightarrow [0,1]$. Let w_A be the total number of hours worked on task A at a given point in time ($0 \leq w_A \leq W_A$). The quantity of information available at any given point is then $I_A = C_A R_A(w_A/W_A)$, measured in nominal hours of communication.

The scope, thoroughness, and completion function of a dependency specify the portion of the information generated in a task that is needed by the dependent task. The time required to transfer this information may be calculated. Assume there are two tasks, A and B with a dependency from A to B. Let S_{AB} be the scope of the dependency ($0 \leq S_{AB} \leq 1$), T_{AB} be the thoroughness of the dependency ($0 \leq T_{AB} \leq 1$), and $P_{AB}(x)$ be the completion function, where $P_{AB}(x) : [0,1] \rightarrow [0,1]$. We can calculate the required amount of information needed to allow task B to finish a certain quantity of work w_B. This is given by:

$$I = C_A T_{AB} S_{AB} R_A\left(\frac{w_A^*}{W_A}\right) \tag{2.1}$$

with

$$w_A^* = W_A P_{AB}\left(\frac{w_B}{W_B}\right) \tag{2.2}$$

where w_A^* is the required amount of work that must have been accomplished in task A and I is the total amount of information that must be known.

When an individual is responsible for two or more tasks that depend on a single task, then that individual has the advantage of being able to combine informational requirements. This is done by representing the information requirements as a union of rectangular solids in the

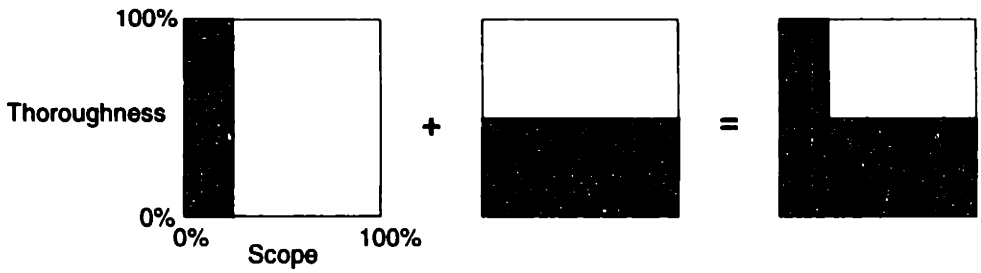


Figure 2.5: Multiple needs for information from one task. The first is of scope 25%, thoroughness 100%. The second is of scope 100%, thoroughness 50%. Overall, 62.5% of the generated information is needed.

information space. Overlapping information only needs to be asked for once. For example, assume that an individual needs to satisfy two dependencies that have the same upstream task (see figure 2.5). If one dependency needs narrow scope (25%) and high thoroughness (100%), and the other dependency needs wide scope (100%) but lower thoroughness (50%), then the combined quantity of information needed will be the union of these two rectangles, or 62.5% of the nominal communication time (appropriately modified by the desired degree of completion).

It is common for individuals to acquire information about a task via multiple transfers of information. Each transfer of information consists of a unique subset of the information solid. If you think of the derivative of the convergence curve of a task as giving the "density" of the information solid at any point of scope, thoroughness, and completion, then the nominal amount of time it takes to transfer an arbitrary piece of information is just equal to the integral of its "density" over its volume.

Individuals may not fill in the information solid arbitrarily. Each additional chunk of information must rest on a piece more substantial than itself. For a given point in the information space to be valid, each point at the same scope and thoroughness must be filled in for all lower completion levels, and each point at the same completion level, but lower scope and thoroughness, must be filled in. This is equivalent to saying that the total known information solid must be able to be decomposed into the union of simple information solids, where a simple information solid is a rectangular solid with one corner at the origin of scope, thoroughness, completion = (0,0,0).

An individual who receives chunks of information out of sequence has to discard the unsupported information chunks. For example, if Susie knows nothing about a task, then she gains no knowledge by receiving a chunk of information designed to update a person who knows completion level 90% of the task to completion level 100%. The information is discarded because she does not have the proper foundation on which to base the information chunk. Section 4.2.1 discusses how these knowledge requirements are enforced in the computer simulation.

Note that the *DiFS* model of how knowledge is represented implies that information transfer linearly adds; that is, the time it takes to receive the entire chunk of information in one shot is the same as the sum of the times it would take if you received 10 updates over the life of the task. This is a simplifying assumption; for a discussion of alternatives, see section 9.2.

Dependency Example

Understanding how a dependency is incorporated into a design model is fundamental to understanding the information-flow model. Consider the simple dependency relationship expressed in figure 2.6. TASK B requires information from TASK A. The dependency specifies a scope of 75% and a thoroughness of 50%, representing a fairly wide scope of information needed, but not a great depth. The information convergence curve of TASK A is needed to calculate the time required to transfer information at various levels of completion; this graph indicates that TASK A generates the bulk of its data within the first 10 hours of work.

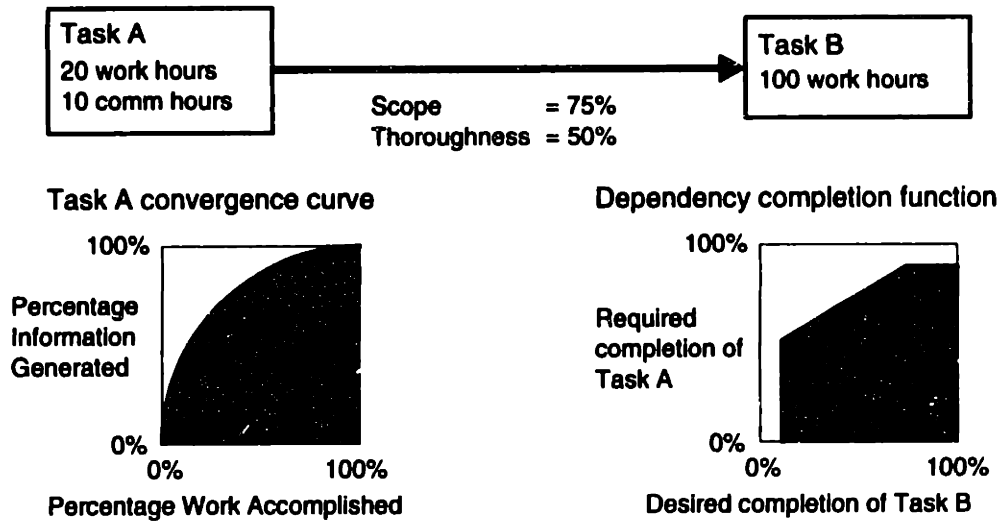


Figure 2.6: Sample dependency relationship showing upstream information convergence curve and dependency completion function.

Before any information is transferred from TASK A to TASK B, the dependency completion function indicates that TASK B can do approximately 12 hours of work. To proceed beyond that point, TASK A will have to be completed to at least the 50% level (or 10 hours of work) and TASK B will have to acquire the generated information.

Figure 2.7 shows the information transfer requirements assuming that information is transferred exactly three times at TASK A completion levels of 25%, 75%, and 94%. Figure 2.8 shows the information convergence curve of TASK A and the dependency completion function at each of these three transfer levels.

At the first information transfer, TASK A is 25% complete (5 hours of work completed), but has generated almost two-thirds (64%) of the total information to be created. The communication time of TASK A is 10 hours, so 6.4 hours of communication is available. However, the dependency specifies 75% scope and 50% thoroughness, so that quantity is scaled by $0.75 \times 0.5 = 0.375$ to 2.4 hours of communication. Assuming that the upstream person has a normal communication efficiency of 100%, it takes 2.4 hours to transfer the information from the upstream person to the downstream. Note that even though information has been transferred, the nature of the dependency still limits TASK B to a completion level of at most 12%.

For the second information transfer, TASK A is 75% complete (15 hours of work completed) and has generated 94% of the total information to be created. Hence 9.4 hours of information is available to be transferred. All of this information does not need to be transferred because (1) the person downstream has prior knowledge from the last information transfer, and (2) the dependency specifies limited scope and thoroughness. Only the necessary hours of information are exchanged; this can be calculated to be just 1.1 hours of information. After this exchange, the person working on TASK B can complete it up to 54% of its final level, or 54 hours of work.

For the final information transfer, TASK A is 94% complete (19 hours of work completed) and has generated 99% or 9.9 hours of the 10 hours of information. However, TASK B does not need such complete information. A TASK A completion level of 88% is sufficient, which corresponds to 98% information generated. The extra information that has been generated is not needed and hence is not transferred. After this final information transfer, the person responsible for TASK B has satisfied all dependency requirements.

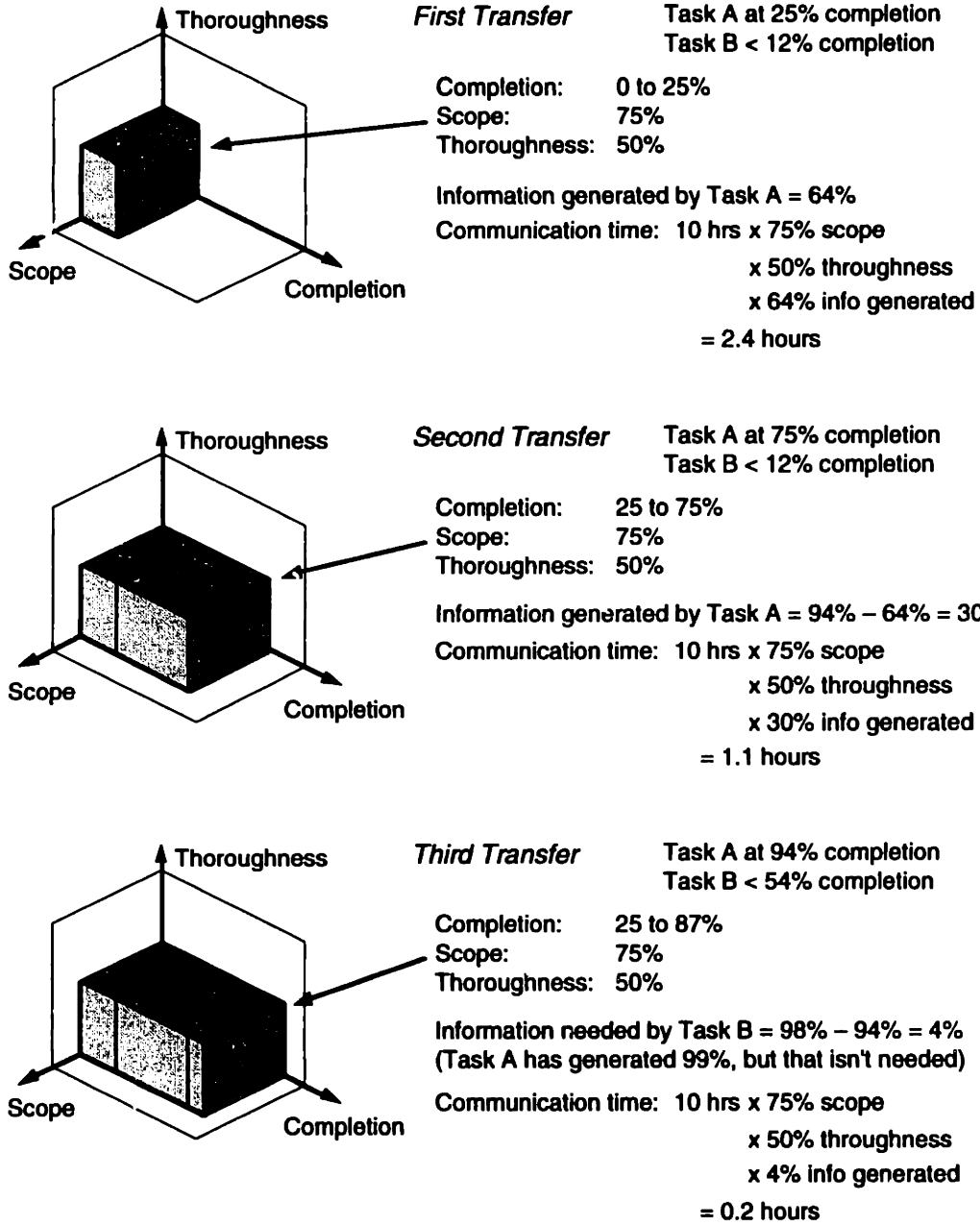


Figure 2.7: Information transfers of sample dependency relationship of figure 2.6. Three separate information transfers are conducted with TASK A at completion level 25%, 75%, and 94%. The nominal time required to communicate the information is a function of the completion level being transferred, the convergence curve of TASK A, and the scope and thoroughness of the dependency.

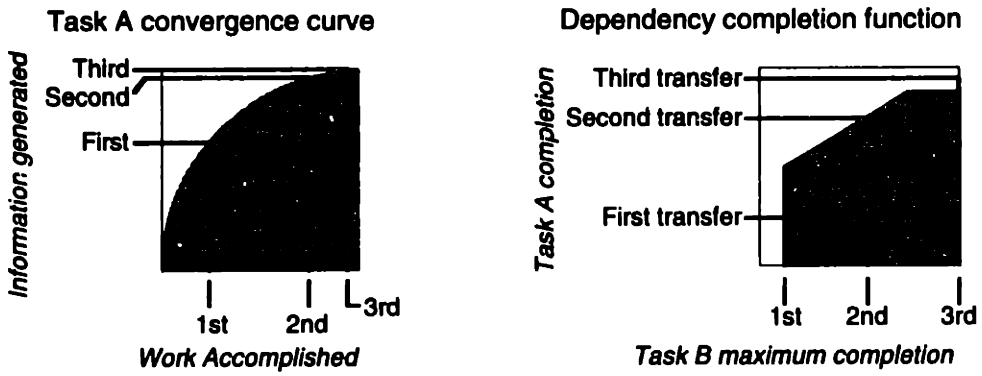


Figure 2.8: Information convergence curve of TASK A and maximum completion levels of TASK B at each information transfer (as given by figure 2.7).

2.1.6 Sample project

The preceding sections have specified the parameters that are associated with tasks, with people, and with dependencies between these tasks. With this information it is possible to calculate how long it would take a team of engineers to accomplish a design project. Of course, this calculation is strongly dependent upon the sequence of actions that the engineers take; the more effectively they coordinate their activities, the quicker the project will be completed.

The human-factors experiment from section 2.1.2 will be used as an example (see figure 2.1 on page 29) of project performance. A complete set of task, person, and dependency parameters has been assigned to the this project (as shown in figure 2.9).

Figure 2.10 is a timeline of one possible series of actions that would complete the project. The timeline was created by hand by following the rules of the information-flow model: First, individuals only do one thing at a time; communicate, work, or nothing at all. Second, two engineers communicating with each other do so simultaneously. Third, work on the group decision task is conducted simultaneously by all three engineers. Fourth, Chris is a part time worker who only shows up in the morning.

The total amount of time required to finish the human-factors project is a little over 20 days, or approximately 184 working hours. This is rather longer than the original estimate of 140 working hours. Better coordination of information exchange between the participants can lower this time, but it is tedious to arrange by hand.

2.2 Modeling Issues

Three complex modeling issues need to be discussed before finishing with the information-flow model. The first is how to decide when work on a design project has finished. The second is how to deal with models of rework. The third is why traditional optimization algorithms are unsuitable for the model, and hence why I describe a simulation approach in later chapters.

2.2.1 Project Termination

Unlike a traditional PERT model, the information-flow model does not have a well defined task or set of tasks that clearly mark the termination of the project. For example, the graphical arrangement of tasks in figure 2.1 suggests that the REVIEW task will be the last to finish, but in fact the dependencies allow any of REVIEW, ANALYSIS PROGRAM or INTERFACE SOFTWARE to be the last task finished.

The logical solution to this problem is to specify that the project has finished when each of the tasks in the project have finished. An individual task has finished when the person

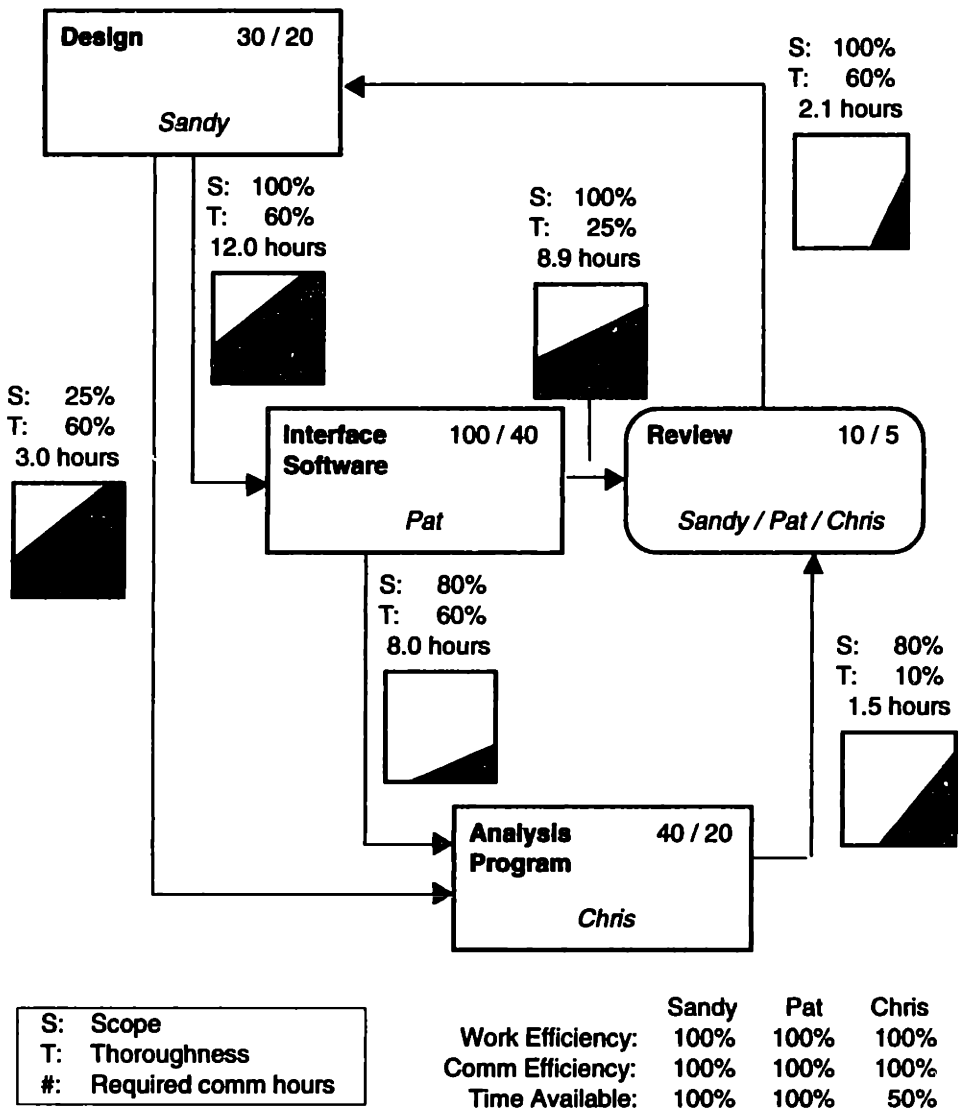


Figure 2.9: Sample parameters for the human-factors project. The work and communication time for each task is given in its upper-right hand corner. All information convergence curves are linear. The boxes with the gray polygons are completion functions for the dependencies (see figure 2.3).

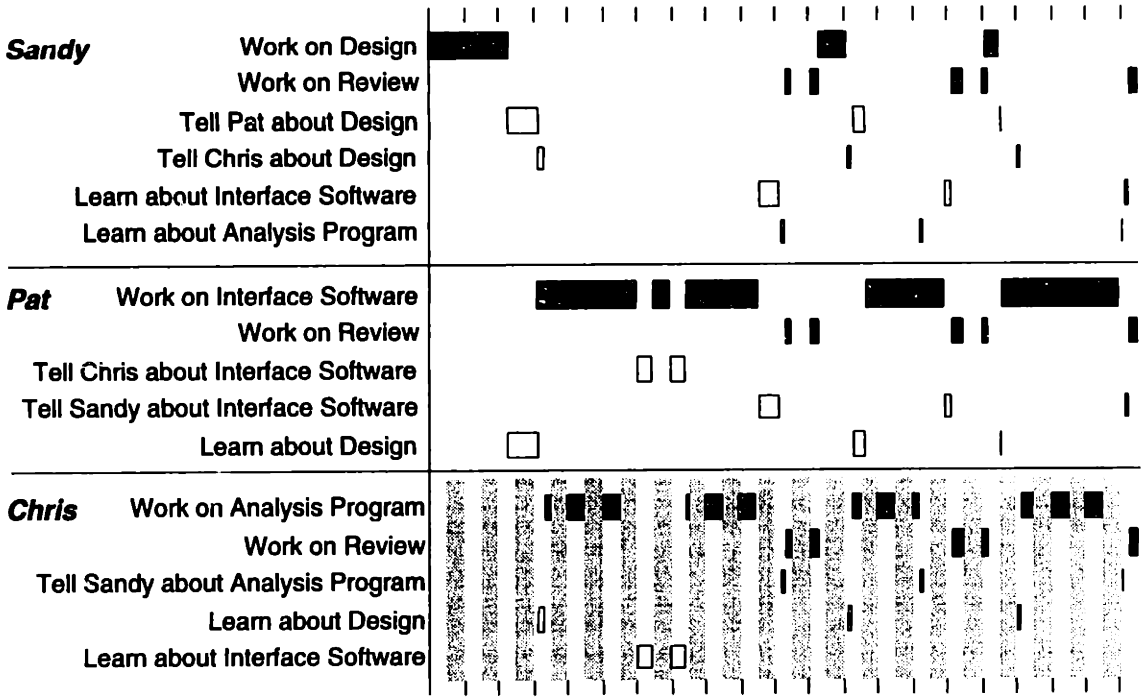


Figure 2.10: Sample project timeline for the human-factors experiment. Hash marks represent nine hours days. Boxes represent time spent working (black) or communicating (white). Note that Chris is only available in the morning.

responsible for that task has completed work on it. Note that in the *DiFS* model it is quite natural for all of the people to run out of things to do without any one person being aware that all of the tasks in the project have finished. This arises because each person's knowledge is private. Hence project termination is normally defined to occur when all individuals know that the tasks they are responsible for are finished.

In the *DiFS* simulation, the introduction of **management tasks** allows a different definition of project termination to be used (section 3.2.1).

2.2.2 Modeling Rework

Rework in the design process is work that has to be redone in a task because the first time it was completed it was either (1) based on inaccurate information, or (2) done incorrectly. Note that the first category covers tasks that need to change because an upstream task was reworked and the changes propagated down. In many design processes rework is required because preliminary decisions need to be made so that a design may be evaluated and then corrected. This type of rework is called iteration.

The human-factors experiment of figure 2.10 demonstrates a form of iteration. The project loop was circled three times. During this time, information generated by the two software tasks was fed into the group discussion. Results from the group discussion were used to modify the experiment design and from there back into the software tasks.

This particular type of iteration is a form of a model of rework. It has the advantage that the total quantity of work in each task is known and the relationships between the tasks are known, hence there is no randomness involved. On the other hand, the number of iterations is generally not well-defined. For example, figure 2.11 is an example of two interdependent tasks. Any series of steps of alternately working on these two tasks must remain within the white valid region of the diagram. But there is no restriction on how many steps are taken or whether steps are taken at all. A valid approach would be to work simultaneously on both TASKS A and B.

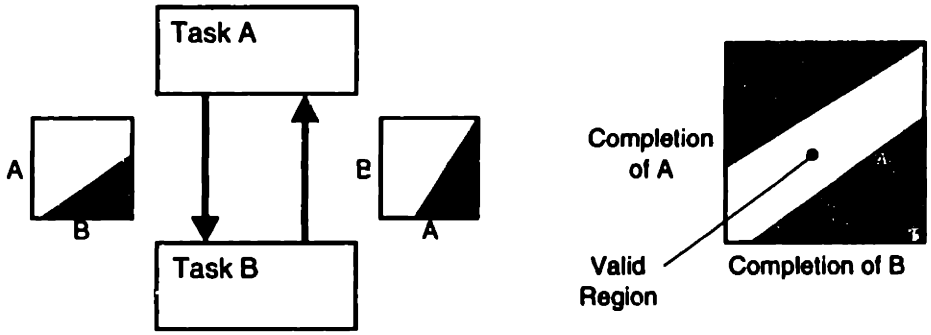


Figure 2.11: Exampie of two interdependent tasks and how the intersection of the dependencies restricts the order in which the tasks can be accomplished. The diagram on the right shows the parameterized completion space with the two dependency completion functions superimposed.

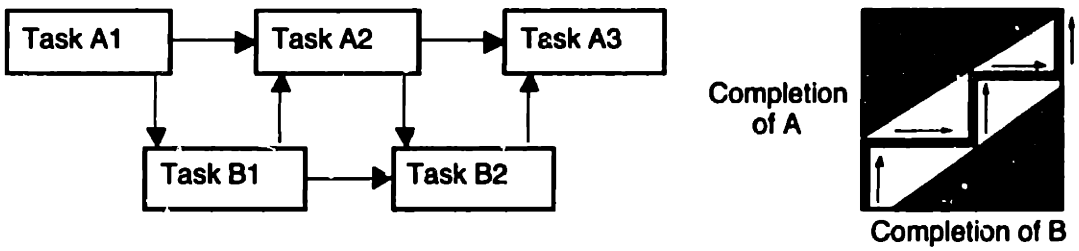


Figure 2.12: Example of two interdependent tasks (figure 2.11) being unrolled into five separate tasks with no loops between tasks.

Within the information-flow model there is no immediate method of representing real iteration in the sense that a product is developed and refined and refined again until it reaches satisfies some criteria of completion or quality. Such a process requires either (1) the number of loops through the iteration cycle be well defined or (2) a defined way of deciding when a task satisfies a quality metric. In the case of the former approach, one can always unroll the iteration loop. For example, figure 2.12 shows the sample from figure 2.11 unrolled into a series of subtasks. This approach requires the modeler to decide upon the number of iterations. The case of measuring the quality of a design and iterating until it satisfies a criteria is not possible to represent in *DiFS*. Refer to section 9.2 for a discussion of possible extensions that address this problem.

2.2.3 Optimization

The human-factors experiment of figure 2.10 is not optimal in the sense that we could find a different way of allocating time to make the overall duration of the project shorter. For example, two people could listen at the time one is talking, or information transfer could be coordinated more closely to guarantee that all three individuals were working at their maximum capacity.

One method of exercising the *DiFS* model would be to optimize the order in which individuals work and communicate with the objective of minimizing the overall duration of a project. For a given project, this optimization would determine the minimum possible duration of the project given the allocation of people and the parameters assigned to tasks, people, and dependencies. The sensitivity of the overall project duration to model parameters could be calculated by perturbing the parameters and re-optimizing to calculate new project durations.

The actual optimization problem is quite difficult. This can be seen by comparing it to a known class of difficult optimization problems. Consider a project with a set of sub-projects, each consisting of a series of tasks connected serially with minimal information transfers re-

quired by the connecting dependencies. If we pretend that people are "stations" and that the sub-projects are "jobs", then this is the classic *Job-shop scheduling* problem (each station in the job-shop is equivalent to a person in the *DiFS* model; each task in the sub-project chain represents the need to do something at a particular station). The job-shop scheduling problem is notoriously difficult to optimize (for example, see [60, 81]). More direct solution methods for general projects are even more difficult to optimize [22, 31]; hence heuristic methods are normally employed.

The second difficulty with optimizing the duration of a project modeled with *DiFS* is that this requires the individuals in the project model to behave perfectly. That is, they must perform their tasks in just the right sequence, transferring information as needed and waiting when appropriate. This type of global coordination between humans generally does not occur in an industrial setting. Rather, individuals work on what seems to be appropriate to them at the moment based on a *bounded rationality* view of the world and the perceived consequences of their actions [53].

The two problems with optimizing the projects in the information-flow model (computational difficulty and violation of bounded rationality) eventually led this research in the direction of simulation. The simulation model is discussed in chapters 3 and 4.

2.3 Discussion of Assumptions

A number of assumptions have been made in formulating the *DiFS* model. This section discusses what the assumptions are and why they were made. It is divided into assumptions about tasks and assumptions about people.

2.3.1 Task Assumptions

Task assumptions deal with tasks and how dependencies connect tasks in the information-flow model. Section 2.3.2 discusses assumptions made about how people satisfy dependency requirements and perform work.

The first assumption is one that is easy to forget, but is fundamental to the entire modeling effort: A project can be subdivided into a set of well-defined, independent tasks. By "well-defined, independent" tasks, I mean that clear stopping and starting points for each task can be identified and that when dependency criteria have been met, a task can be started independently of starting or stopping other tasks in the model. This assumption lies at the core of PERT/CPM methods of modeling projects. For discussion on the appropriateness of this point, refer to Eyring [28] or Wiest [83].

The following assumptions have been made about tasks:

- Tasks in a project can be modeled as pieces of work that are done by either an individual (work tasks) or a group working in concert (group decision tasks). Work tasks are done by individuals working alone. Group decision tasks are done by a group of engineers working together simultaneously in a meeting setting.

- Task durations can be estimated. Task durations are measured in work time, or the nominal amount of time it would take to complete the task assuming all dependency requirements have been satisfied.

- Task information content can be estimated. The information content of a task is measured by communication time, or the nominal amount of time it would take one person to completely explain the task to another qualified person in a face-to-face discussion.

- Dependencies between tasks can be represented as a need for part of the information generated by the upstream task. The required slice of information can be parameterized by scope, thoroughness, and completion function.

Issue: Why are work tasks defined to be done by just a single individual?

Most of the tasks within the information-flow model are work tasks. Work tasks are done by a single individual. PERT and CPM methods normally allow groups of people to be assigned to a single task; hence the information-flow model is a finer subdivision of the design project.

The decision to make work tasks the responsibility of a single individual represents a tradeoff between modeling convenience and the representation of information requirements. Large tasks done by groups simplify the modeling process, but leave open the question of how the individuals assigned to the task will coordinate their work and exchange information. Tasks done by individuals do not have this problem, but because they typically represent a smaller piece of work, more of them are needed to model the design project. The great advantage of the individual task size is that the coordination requirements between individuals can be expressed as information dependencies between tasks.

Issue: To model a design project, many individual tasks and dependencies need to be specified. With all of these assumptions, how can we be confident of the model predictions?

The answer to this issue is that although the accuracy of the model of any single task or dependency may be limited, in the aggregate the model may be well behaved. It may not be possible to model a large design project and then precisely predict when each particular portion of the design project will be done and in what order. However, it may be possible to get a good estimate of the overall duration of the entire design project and the sensitivity of the project to variations in task sizes or personnel assignments. Later chapters of this thesis are devoted to the problem of determining the accuracy and reliability of the aggregate behavior of a large model.

Issue: What about the quality/rework issue? Shouldn't task size be a function of how well the upstream tasks succeeded and the quality of the information available?

There are several answers to this issue: First, the *DiFS* model provides a simple method of modeling rework, as discussed in section 2.2.2. Second, interesting questions on rework or quality can be addressed by how the *DiFS* model is exercised. For example, modeling the effect of unexpected rework can be as simple as adding a new dependency and seeing what happens when the model is exercised. Third, we do have a number of possible extensions to the model (section 9.2) that explicitly incorporate rework and quality issues.

2.3.2 People Assumptions

The following assumptions have been made about people and how they deal with information in the model:

- Individuals are assigned fixed roles on tasks in the project. They can be responsible for a work task, responsible for a group decision task, or participate on a group decision task. Task assignments are fixed during the life of the model; individuals are not dynamically reassigned from task to task.

- Individuals do one activity at a time. They can either communicate information with another person, work on a task, or be idle. Part-time workers work on tasks outside the scope of the design project and hence are unavailable part of the time.

- In order to make progress on an assigned task, an individual must possess appropriate knowledge about all tasks that drive the assigned task. Parameters in the upstream tasks and associated dependencies control the duration of the information transfer. For decision tasks, only the individual responsible for the decision task must satisfy the dependency requirements.

- Parameters specified for each individual control how quickly they work on tasks (work efficiency), how quickly they communicate information to other people (comm efficiency), and what percentage of time they are assigned to this project (available time). Individuals

who work less than full time are available in the morning. Progress on decision tasks is affected by the responsible person's work and comm efficiency, but not by the size of the group.

- Information communicated to individuals is always accurate and accumulates linearly. That is, there is no advantage or disadvantage to receiving information in many small pieces separated over time, or receiving it in one big chunk after a task has been completed. The total amount of communication time will remain the same.
- A piece of information communicated to an individual is only useful if the appropriate amount of prior information is known by the individual. An individual who receives chunks of information out of sequence has to discard the unsupported information chunks.
- There is no rework or working ahead. If an individual is stopped on a task by a need for information from a driving task, then no more work can be done until that information is available. It is not possible to work past the point specified by the driving dependencies.

Issue: Information accumulation is unrealistic. The current model assumes linear information accumulation without regard to when the information was generated or how recently the individual was updated. That is, it takes the same total amount of communication whether a person is told 100% of a task's knowledge after the task has been completed, or if that information is passed along 10% at a time as the task is being completed.

It is implicitly assumed in the information-flow model that information, once generated, is always accurate and useful. It is also assumed that *when* the information is acquired does not make a difference. This is not consistent with our experience. In reality, an individual who wishes to acquire information about a task is better served to wait until the task has completed, lest he/she acquire a large quantity of information that will change and need to be reacquired.

These assumptions were made for two reasons: first, they greatly simplify the model. Second, only a portion of the cost of frequent communications arises from repeating or correcting old information. The remainder of the cost is the overhead of arranging communication. This overhead takes the form of setting up a meeting or discussion, starting the conversation, requesting the appropriate level of detail, and satisfying social conventions. The overhead associated with communication is explicitly represented in the simulation of the information-flow model, described in chapter 3.

Issue: Working ahead based on partial knowledge should be allowed. An individual should be able to make some form of progress on a task even though he/she does not have the complete information required by the dependencies.

This issue can be considered part of the rework issue, where the quality of the progress that an individual could make would be based on some measure of the quality of the partial information available. It has not been addressed in the *DiFS* model, although a logical extension to the model provides these facilities (see section 9.2).

Issue: The modeling representation is not rich enough. It should be enlarged to capture more types of roles, cooperative behaviors, teamwork, asynchronous communication (writing reports and memos), and other such effects. These additional factors would affect how effectively individuals work together, how long it takes for them to transfer knowledge, and what types of coordination are necessary between individuals.

The only real answer to the issue of insufficient modeling detail is to defer the reader to later chapters of this thesis where a number of these factors have been incorporated or experimented with. Chapter 3 discusses how different types of communication have been incorporated. Chapter 8 discusses how some of the cooperative behavior and teamwork issues can be explored within the context of the model. And finally section 9.2 discusses extensions to the model that more directly take into account the rich and complicated nature of individual roles, behaviors, and coordination.

Issue: The fitness of an individual's assignment to a particular role has not been addressed. It is possible within the simulation to arbitrarily assign any person to any role on any project, regardless of their proposed abilities.

This issue is one of omission: that is, *DiFS* does not incorporate a way to specify skills possessed by an individual and skills needed by a task. In fact, we originally considered including such factors. They were not included because they require extra input by the experimenter, yet they are only useful if you are attempting to use the computer to optimally allocate personnel to tasks. Although the *DiFS* simulation can be used for this purpose, it is not the main thrust of this thesis. Hence the assignment of skills and abilities is left to the future work section.

2.4 Summary

The information-flow model is a model of a group of people working on a design project. It attempts to capture the necessary flow of information between the individuals and the coordination of their activities as they attempt to complete the tasks associated with the design project.

The last two sections on task and people assumptions have raised a number of modeling issues dealing with assumptions, omissions, and potential difficulties with the current model. Many of the issues raised were dealt with by referring the reader to the "Future Work" section of the document. This has been done not because I consider these issues unimportant, but because I have tried to create a model that was as simple as possible and yet still could capture the concept of necessary information exchanges between engineers.

Ultimately the measure of a design model is how well it can represent the process of engineers working on a design project and how it can assist managers in doing their job better. Before these effects can be discussed it is necessary to cast the model in a form that can be easily manipulated by an experimenter and can make predictions about project performance. The next chapter discusses the *DiFS* simulation. The *DiFS* simulation is a computer program that implements the information-flow model by using computer agents to represent people working on the design tasks in a simulated office environment. Later chapters will discuss validation experiments and how the simulation may be used to make predictions about project performance.

Chapter 3

Implementation

This chapter of the thesis describes the implementation of the *DiFS* simulation. The *DiFS* simulation is an implementation of the information-flow model as a time-based simulation with computer agents playing the part of the people in a simulated office environment.

This chapter is divided sections that discuss the simulation objectives, extensions to the information-flow model, the office environment, how agents communicate, how meetings are scheduled, and a summary of the actions that agents may take. The behavior model of the agents within the simulation is described in chapter 4. You may also wish to refer to appendix A, which is a copy of the user's manual for the *DiFS* simulation program or appendix B, which is a description of the algorithms and software used to implement the simulation.

3.1 Simulation Objectives

This section discusses the objectives of the simulation and alternative approaches that were considered and rejected.

3.1.1 Goals

The intent of the information-flow model is to represent a group of engineers working together on a design project, discussing problems, and sharing information. In creating the simulation, my goal was to use this model of the engineering process in the following ways:

- To estimate the duration of a design project and when resources will be used.
- To identify critical tasks and personnel assignments that can potentially impede progress on the project.
- To examine alternative personnel assignments and team sizes to look at resource and cost tradeoffs.
- To experiment with communication issues such as the co-location of team members or how team members communicate (using technologies like electronic mail or videoconferencing).
- To identify ways that a manager may influence project duration by encouraging work on critical tasks or organizing team meetings.

Additionally, I made the requirement that the method of using the model should incorporate the idea of bounded rationality of human behavior (see [53]); that is, the people assigned to work on the design project do not behave "perfectly." Instead, they should behave as well as could be expected, assuming they have finite knowledge and finite reasoning capability. Note that this requirement does not preclude globally optimizing the approximate order in which tasks are accomplished. It does preclude optimizing the precise times the information transfer occur.

3.1.2 Alternative Approaches

There are two standard methods of utilizing or "exercising" a model: mathematical and simulation (see [62]). Both of these methods were considered for the *DiFS* model. This section discusses the pros and cons of these methods.

Mathematical approaches attempt to algorithmically solve equations representing a project model or otherwise directly calculate useful metrics. For example, directly calculating the slack times associated with various tasks in a PERT model [83], or calculating the eigenvalues associated with a DSM (Design Structure Matrix [70]). One standard mathematical approach is to optimize a design project by minimizing a cost function associated with the project model. Depending on the complexity of the model, this may be as simple as using the Simplex method, or as complicated as a mixed integer nonlinear programming method. For extremely complex models, optimization may require a reasonable starting point. These starting points are calculated using heuristics.

The information-flow model is complicated enough so that traditional mathematical techniques are intractable. Moreover, implementing the bounded rationality behavior model is quite difficult in an optimization approach. Section 2.2.3 discusses some of the difficulties inherent in the optimization approach applied to the *DiFS* model. Hence a simulation approach was chosen for *DiFS*.

Simulation approaches come in a variety of flavors, but the basic idea is the same: the model of interest is cast in a time-dependent fashion. Time is either assumed to flow continuously (e.g., system dynamic models [29]), or at discrete intervals when events occur.

I initially explored using either an event-based technique or a system dynamics technique for the information-flow model. System dynamics quickly proved to be inappropriate for the scale and type of simulation proposed; it is better suited for models where individuals spend percentages of their time continuously on various activities. However, individuals in the real world tend to do one thing at a time and stick with that action until they are finished or get interrupted. The *DiFS* model tracks individuals and their tasks, hence a discrete time, event-based model was the most natural and straightforward approach.

3.1.3 Selected Approach

A multi-agent simulation using discrete time events was selected for the information-flow model. To address the issues of communications and bounded human rationality, each human in the *DiFS* simulation is represented by a computer agent. Those computer agents interact in a simulated office environment. In order to minimize questions of validity and to assist in the construction of the simulated office environment, the simulation deliberately mimics real-world office environments, communication methods, and management structures. These structures are extensions to the *DiFS* model that control how the computer agents will interact and accomplish work.

The following is a list of the extensions to the *DiFS* model that make up the core of the *DiFS* simulation:

Augmented Task Model: A method of defining management tasks is introduced that groups related tasks into logical units and provides simple managerial behavior. The augmented task model enables the scheduling of periodic team meetings and provides a mechanism for identifying the termination of the entire project.

Office Environment: An office environment is provided for the computer agents. Agents are assigned offices to work in. Meeting rooms are provided for group discussions and team meetings.

Communications System: A flexible communications network allows agents to communicate with each other either asynchronously or synchronously. Types and speeds of communication are defined by the experimenter.

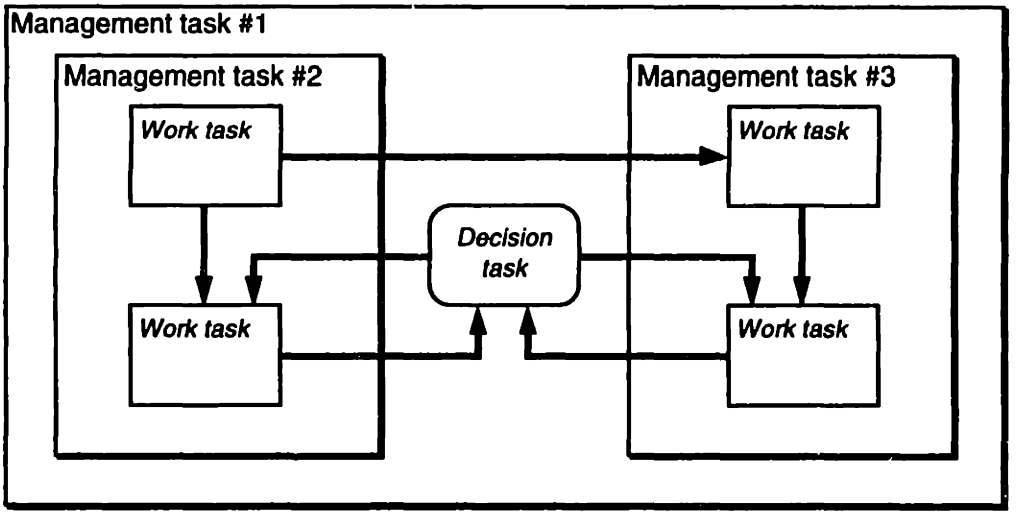


Figure 3.1: Example of management tasks.

Meetings: A global meeting mechanism allows agents to schedule meetings to work on a group decision task or to hold a team meeting.

To handle these model extensions, an augmented set of agents actions is provided beyond the simple set of work, communicate, or do nothing. These actions allow agents to send asynchronous messages, travel about the office, and locally coordinate their activities with other agents.

The remainder of this chapter discusses the extensions to the information-flow model and the actions agents may take. The logic of how a computer agent selects an action based on recent events, private knowledge, and the local environment is called the *agent personal behavior model.* The personal behavior model is discussed in chapter 4.

3.2 Augmented Task Model

The augmented task model extends the *DiFS* model to provide simple management structures. This section is divided into three parts: a description of how the task model has been augmented with management tasks, a description of additional parameters associated with tasks, and a description of how the task model influences agent behavior.

3.2.1 Management Tasks

The **management task** (or **meta task**) has been added to the information-flow model for several reasons: to introduce a simple form of management, to make it easier to determine when the simulation run has finished, and to provide a mechanism for scheduling periodic team meetings.

The management task does not represent work to be done; it represents a grouping of related tasks. Each management task contains an arbitrary number of other tasks. Management tasks are allowed to contain other management tasks. In fact, the entire design project itself is always contained within a single top-level management task. The design project is arranged in a tree structure. Each branch in the tree is a separate management task and the leaves of the tree are either work tasks or group decision tasks. Figure 3.1 shows an example project with three management tasks.

Management tasks do not have any work associated with them. Hence management tasks

may not be connected by dependencies either to each other or to work or group decision tasks.[1]

One individual is assigned to be **responsible** for a management task. Other individuals may be assigned to be **participants** in a management task. The individual who is responsible for the management task is considered to be acting as the manager of all individuals working on tasks contained within the management task. Currently the acting manager does the following things: he/she calls team meetings, keeps track of the progress of all tasks contained within the management task, and provides information on task progress to his/her superior (the agent responsible for the management task that contains this management task).

3.2.2 Task Parameters

The augmented task model adds three new task parameters. These parameters assist agents in the prioritization of activities and scheduling of regular group meetings. They are task **priority**, **meeting frequency**, and **documentation thoroughness**. All types of tasks (work, group decision, and management) have a task priority. Management tasks have a meeting frequency, and work and group decision tasks have a written documentation thoroughness. These parameters are assigned by the experimenter.

The priority of the task represents the relative importance of a task in relation to other tasks. Task priorities are given by a simple ordinal scale from "very low" to "very high". Task priorities are used by the computer agents to help decide what to do. The intent is that higher priority tasks should be given more time and finished sooner than lower priority tasks. However, this is only a suggestion to the computer agents; task priority does not enforce any particular action by the computer agents.

The meeting frequency of a management task specifies how frequently the individuals assigned to roles on the management task should schedule and hold team meetings. Normal options are: never, daily, twice weekly, weekly, twice monthly, and monthly. The agent responsible for the management task schedules the meetings. All agents assigned a specific role on the management task show up for team meetings and preferentially discuss tasks and dependencies related to the particular management task.

The documentation thoroughness of a work or group decision task represents the thoroughness of the information available in a written form that has been generated while doing that task. This written information is referred to as the **formal** documentation. All work and decision tasks generate information. That information is stored in the head of the people who have participated in the generation of information. However, some types of tasks may also generate information in the form of written documentation. That written documentation is generally not as thorough as the knowledge that is stored in people's heads, but it does have the advantage that it can be easily handed to another person. The *DiFS* simulation allows the person setting up the simulation to specify the thoroughness of the written documentation that is generated. This documentation is assumed to always be at the current completeness level of the task and 100% scope.

Table 3.1 lists all of the parameters assigned to tasks by both the information-flow model and the augmented task model. These parameters are specified by the experimenter.

3.2.3 Agent Behavior on Management Tasks

The addition of management tasks to the information-flow model adds some complexity to how agents need to behave. First, agents assigned to a management task may need to schedule and attend team meetings. Second, agents working on tasks that are contained within a management task need to keep the manager updated on the status of the tasks they are working on.

The current implementation of computer agents handles team meetings as follows:

[1] In prior simulations we experimented with dependencies between management tasks as a method of representing "lump" dependencies between the tasks contained within each management task. This capability has been removed.

Table 3.1: Parameters assigned to each task by the experimenter.

Parameter	Value	Task types
Work time	Hours	Work and group decision
Communication time	Hours	Work and group decision
Convergence curve	Function	Work and group decision
Documentation thoroughness	0–100%	Work and group decision
Task priority	By list	All
Meeting frequency	By list	Management

1. The individual who is responsible for the management task has the responsibility of scheduling the team meeting.

2. Team meetings will only be scheduled if the management task is actively being worked on. That is, meetings will not be scheduled if either (1) none of the tasks within the management task have started, or (2) all of the tasks within the management task have finished. Note that as the responsible agent is the one who schedules the meetings, it is that agent's viewpoint that matters.

3. All individuals who are **participants** on the management task will be invited to the team meeting. Individuals who do not have an assigned role on the management task, but do have a role on tasks contained within the management task are *not* invited.

4. A management task that has no participants or that has a meeting frequency of "never" will not hold team meetings.

5. The duration of team meetings is set by the experimenter. Team meetings normally have a duration of one hour.

6. At the team meeting any topic of conversation is valid. However, the agents are aware of the purpose of the meeting, so discussions regarding tasks contained within the management task are encouraged.

Agents that are responsible for a management task are considered to be the manager of the subtasks contained within the management task. The current simulation implementation assigns them the following responsibilities:

1. The manager attempts to keep track of the status of all of the subtasks contained within the management task. This includes subtasks of contained management tasks.

2. Knowing the status of a task is considered to be equivalent to knowing the current completion level of the task. Think of this as possessing task information at the current completion level with 100% scope and 0% thoroughness. Status reports take just a few minutes to transmit.

3. The manager asks for status reports from the agents directly responsible for the tasks contained within the management task. For example, assume that Joe is the manager and Sue manages a management task contained within Joe's task. Joe will ask Sue for information about work tasks contained within Sue's management task. Joe will not directly contact the agents responsible for those work tasks.[2]

4. Managers prefer to receive status reports at regular intervals. How frequently managers ask for status reports is controlled by the experimenter.

[2]Unless Joe or Sue is one of those agents.

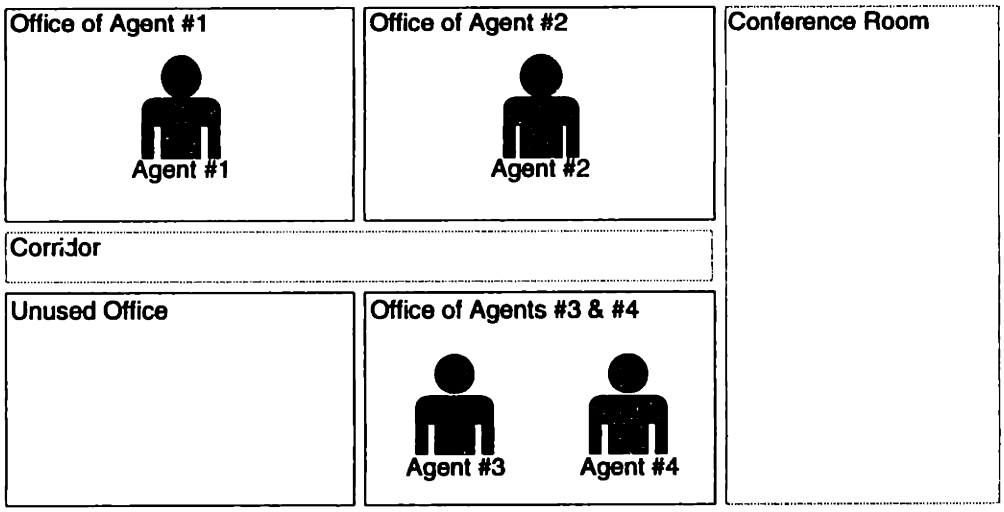


Figure 3.2: Sample Office Environment. Agents are shown in their assigned offices. During the actual simulation run, agents move freely between the different offices and the conference room.

The use of management tasks and status reporting provides a convenient method of determining when the simulation should end; namely, when the person responsible for the top-level task is aware that all tasks within the simulation have completed. As this is the one person who we can guarantee is keeping track of the status of each task in the simulation, this is a logical way to terminate the simulation.

A final thought about management tasks: they are mostly a convenient fiction. They provide a way of hierarchically ordering the tasks in the simulation, they provide a method of scheduling regular team meetings, and they demonstrate the idea that managers keep track of the tasks under their control. But in reality managers spend a great deal of time dealing with administrative issues and handling exceptions arising out of the tasks they are responsible for. In particular, Galbraith [30] would claim that exception handling is a manager's most important function. The *DiFS* simulation does not deal with exception handling, although it is an extension we have considered incorporating (see section 9.2). People interested in simulating managerial exception handling might wish to consult the Virtual Design Team literature [17].

3.3 Office Environment

Computer agents work within the confines of a simulated office building. Agents come to work every morning. They work diligently in their offices and attend meetings in designated conference rooms. They go home after they've logged a full day of work. This section describes the parameters that specify the office building and how agents behave in the office environment.

3.3.1 Office Parameters

The office environment within the *DiFS* simulation is represented graphically as a collection of rooms. Figure 3.2 shows a typical office layout. For convenience we assume that the office is represented by a simple floor plan.

There are two types of rooms in the office environment that computer agents can inhabit: offices and meeting rooms. Each agent in the simulation is assigned to an office. This office is where the agent's desk, telephone, mailbox, and so forth are located. More than one agent can be assigned to an office; it's assumed that they each have a desk located within that office. Agents work on assigned work tasks only when they are located within their own office.

Meeting rooms are available for group discussions or team meetings. The scheduling and usage of meeting rooms is discussed in section 3.5.

A computer agent is always in one of three positional states: at home, in a room, or in transit. A computer agent at home can't be seen or contacted by other agents. A computer agent in a room can be seen or contacted by other agents within that room. An agent in transit is traveling either to or from home, or traveling between rooms. Agents in transit cannot be seen or contacted by other agents.

The time it takes for an agent to travel from one room to another is proportional to the geometric distance between the two rooms. The agent essentially disappears from the simulation for the few minutes he/she is in transit between the rooms, although there is a slight pause associated with the time it takes the agent to exit the room.[3] All time values associated with traveling are under the control of the experimenter.

3.3.2 Agent Behavior in the Office Environment

Placing computer agents in an office environment adds some complexity in how the agents need to behave. First, the agents must decide when to come to work in the morning and when to leave in the evening. Second, the agents must decide when to travel from room to room in the office building.

Full-time agents in the *DiFS* simulation work a standard nine-hour day. Promptly each morning at 8:00 A.M. agents are awakened and travel to their respective offices. Full-time agents normally leave the office and return home at 5:00 P.M. in the evening. Part-time agents arrive and depart the office based on their **time available** parameter. Holidays and weekends are ignored in the simulation, so a standard work week is 5 days long and represents 45 hours of work per full-time agent.

Agents must decide where they wish to be in the office building and travel to the desired location. By default, agents prefer to be located in their own office. They do all work on individual work tasks in their own office. It is assumed that all non-portable equipment possessed by the agent (e.g., telephones, fax machines, answering machines) is located within his/her office.

Agents normally leave their office only to attend meetings, go home for the day, or attempt to contact another agent. Office walls are considered to be solid; an agent is only aware of other agents currently located in the same room. Hence an agent that wishes to exchange information with another agent must either be located in the same room as that other agent or connect to that agent via some form of communication that spans rooms (for example, a telephone).

3.4 Communications

Dependency requirements between tasks force agents to transfer information to one another. The communication time of a task, prior knowledge possessed by the agent, and associated dependency parameters control how long this communication will take, assuming that it takes place in a face-to-face discussion. However, in a real office environment a great deal of communication takes place via other channels. People place phone calls, write memos, and send faxes. These alternative communication methods have advantages and disadvantages; a phone is quick to place, but is limited to two people. A memo takes time to write, but the reader can peruse it at his/her leisure. Faxes are an efficient way to send prepared, formal documentation, but are time consuming for informal communication.

For the *DiFS* simulation we were interested in providing a few of the standard types of communication that occur in an office. Rather than explicitly specifying the allowable types of communication, I have created a general mechanism by which individuals can communicate with each other. The experimenter specifies the particular types of communication at can occur.

[3] In a prior version of *DiFS* I tracked the progress of agents through the corridors of the building. This allowed them to run into other people in the corridors. Unfortunately, it also added a fair amount of complexity to agent behavior to handle the navigation correctly.

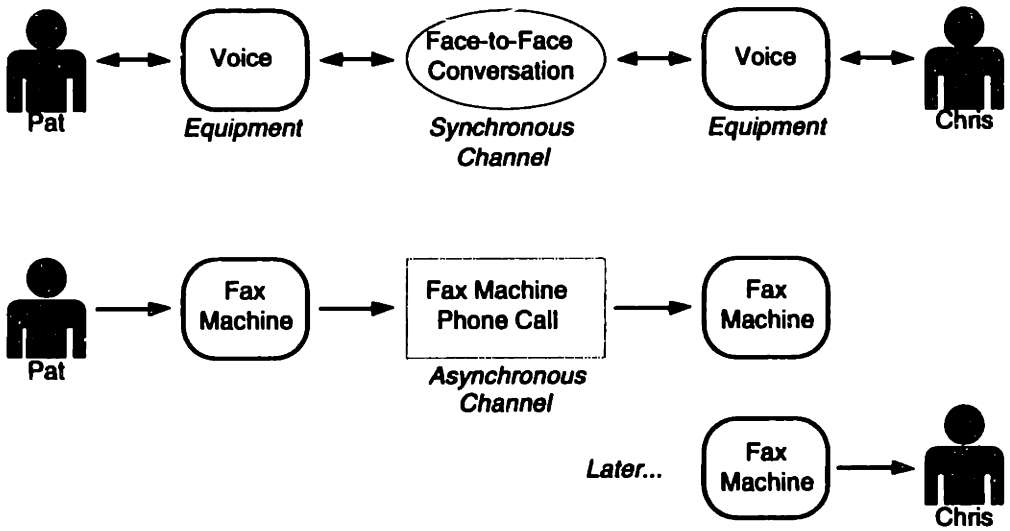


Figure 3.3: Sample communications, both synchronous and asynchronous.

This communication model appears relatively complex at first glance. However, this complexity provides a great advantage: all types of communication from face-to-face conversations through fax machines and pagers can be implemented using the same standard interface. The agent uses this standard interface to select the communication method that appears most advantageous for the current situation.

3.4.1 Communications Overview

Agents within the simulation communicate information by using **equipment** and **channels**. Sample types of equipment include: telephones, fax machines, pagers, mailboxes, and the human voice. Channels are divided into two categories: synchronous and asynchronous. Synchronous channels connect two agents so that they can talk back and forth in a conversation. Examples of synchronous channels included telephone calls, videoconferencing, and face-to-face discussions. Asynchronous channels allow an agent to transmit a message to a piece of equipment owned by another agent. The other agent can retrieve the message at any time. Examples of asynchronous channels include fax machine to fax machine, telephone to answering machine, and telephone to beeper. Figure 3.3 shows the agent-equipment-channel chain for both asynchronous and synchronous communications.

A standard synchronous communication sequence might run as follows: Chris decides to talk with Pat. If Pat is not currently located in the same office as Chris, Chris must either find Pat or attempt to contact her using a telephone or some other between-office communication channel. Assume that Chris and Pat are in the same room. Chris must now verify that the equipment necessary for the synchronous communication channel "talk" is available, namely Chris's voice. Chris opens a synchronous "talk" channel and invites Pat to join in. Pat may or may not answer Chris's request. If Pat does join the channel, then both Chris and Pat have an opportunity to talk. Each formulates a message that she would like to send and specifies how urgent the message is. The talker with the more urgent message gets to send it. The message takes a finite amount of time to transmit. The message can be interrupted during transmission by a Pat or Chris, or by a third party who joins the channel. If the message is not interrupted, both parties are informed that it has been sent, and the agents once again negotiate to see who talks.

A standard asynchronous communication sequence might run as follows: Assume Pat has decided to send Chris a fax. First, Pat goes to her office, because that is where the fax machine is. Second, Pat verifies that the fax machine is not already in use. Third, Pat writes out the

content of the message and starts the fax transmission. The fax machine takes care of sending the message, so Pat do other activities while the fax is being transmitted. Later that day, or perhaps a day or two later, Chris notices that there is a fax waiting for her. She retrieves the fax and reads Pat's message.

The agent behavior required to converse or transmit a message is relatively complicated. The agent must select the appropriate channel to use, access the right equipment, and start up the communication channel. The agent behaviors must be sophisticated enough to recover gracefully if anything goes wrong. The advantage of this relative complexity is that the simulation is realistic; an agent out of his/her office cannot receive phone calls. A fax may sit in the "in" basket indefinitely. An agent can start and stop conversations at will.

The remainder of this section discusses the equipment agents use, synchronous and asynchronous channels, and the behavior implications of this communication model.

3.4.2 Messages & Equipment

Agents within the simulation send messages back and forth over the communication channels. The actual content of the messages is discussed in section 4.2.1. The only relevant consideration as far as the communication channels are concerned is the *size* of the message. The size of a message is assumed to be the amount of time it would take one person to speak the content of the message to another person in a face-to-face conversation (note the correspondence with information requirements in dependencies). The size of a message is usually expressed in nominal seconds. I should add that the actual size of a given message will vary based on the communication effectiveness of the agent sending the message (see section 2.1.4).

For two agents to communicate, one or both must select a communication channel. That communication channel connects a piece of equipment owned by the initiating agent with a piece of equipment owned by the receiving agent. *All* communications between agents must go through each agent's equipment (see figure 3.3). This includes such common communications as two agents just talking (equipment = voice) or handing a document to another person (equipment = hands).[4]

Equipment is defined by three parameters: how many channels may use it at the same time (**maximum simultaneous usage**), whether or not it is **portable**, and whether or not it may store messages.

The maximum simultaneous usage of a piece of equipment is the maximum number of communication channels that may utilize the equipment at a single time. This number is normally either 1 or ∞. For example, a standard telephone can only handle one conversation at a time. Similarly, a fax machine may only receive one fax at a time. On the other hand, a mailbox can receive an arbitrary number of messages simultaneously. A standard telephone with call waiting is an example of a piece of equipment that can handle a maximum of two channels simultaneously.

Portable equipment is carried by the computer agent it belongs to. Non-portable equipment always remains in the agent's assigned office. Examples of portable equipment include cellular telephones, beepers, an agent's voice, and an agent's hands (good for dropping messages on the desk of another agent). Examples of non-portable equipment include standard telephones, fax machines, mailboxes, and computers (used for e-mail).

Some types of equipment can store messages. This type of equipment may be on the receiving end of an asynchronous communication. Stored messages may be replayed at a later time by the owner of the equipment. Examples of equipment that stores messages include: answering machines, fax machines, computers (used for e-mail), and mailboxes.

Message-retrieval parameters are associated with equipment that can store messages. These parameters specify the amount of time it takes an agent to retrieve a stored message and how

[4] I emphasize this point because a common mistake made when using the simulation is to forget to equip agents with the ability to talk to each other. I've done it myself about half a dozen times. If you forget, you can spend hours wondering why two agents sharing an office will call each other on the telephone.

long that piece of equipment is tied up while the message is being retrieved. Note that these two times refer to different concepts; in the former, this is the amount of time it takes the human to scan and understand the message. In the latter, this is the amount of time that the machine is utilized in the retrieval process. For example, when listening to a message stored on an answering machine, the answering machine is tied up during the entire time the message is retrieved. But retrieving a document from a fax machine takes practically no time; the actual reading of the document may take a substantial amount of time. While an agent is reading the document, the fax machine is free to receive another document.

The two message-retrieval parameters are **agent retrieval time** and **machine access time**. These parameters are in the form of a constant amount of time plus an amount of time proportional to the size of the message (measured in seconds). For example, the amount of time it takes to retrieve and listen to a message from an answering machine might be assumed to be 10 seconds plus the length of the message. The machine access time might be slightly longer to account for the rewinding of the tape after the message has finished playing. For a fax machine, the agent retrieval time might be five seconds (to pick up the document) plus a time proportional to the size of the message; if you assume that people read faster than they listen, it might be on the order of 50% of the size of the message.

3.4.3 Synchronous Communication Channels

Synchronous communication channels represent two-way conversations between two or more agents. The two most common synchronous communication channels are face-to-face discussions and telephone calls. The advantages of a synchronous channel are (1) the sender knows the recipient(s) received the message, (2) the recipient(s) may immediately reply to the message, and (3) a series of messages may be exchanged, not just a single message. The disadvantages of a synchronous channel are that (1) participants must all arrange to be on the channel at the same time, (2) the channel is interrupted by individuals joining or leaving at arbitrary times, and (3) negotiations must occur between the agents to decide who gets to speak.

A synchronous channel can be in one of two possible states: **startup** and **active**. To open a new synchronous channel, an agent grabs the appropriate piece of equipment and places the new channel in the startup state. This agent attempts to attract the attention of other agents to join on the channel. The startup phase generally does not last more than a minute, and may be quite short. If at least one other agent joins the channel by the termination of the startup phase, the channel is switched to the active state. While active, agents may arbitrarily join or leave the channel. However, the channel will close the instant that the number of agents using the channel drops below two.

Figure 3.4 shows the standard progression of a synchronous communication (in this case, a telephone call). Note that the channel is in the startup state until Chris and Pat are both on the channel and ready to communicate. Once active, the channel can be either between messages or actively transmitting a message from one agent to the other agents on the channel. When the channel is between messages, each agent specifies what he/she would like to say and how urgently to say it. The agent with the highest urgency wins and gets to transmit his/her message. The other agent(s) may interrupt the talker at any point in time by specify a new message and urgency, thus preempting the current talker.

There are three types of parameters associated with a synchronous communication channel: **general information**, **start information**, and **message information**.

The **general information** parameters of a synchronous channel are (1) the equipment type used for the channel, (2) whether or not participants must be in a common location, and (3) the maximum number of individuals that can share the channel. For example, a telephone call might use "phone" equipment, would not require participants to be in the same office, and would allow a maximum of two individuals to be present on the channel. Assuming that these are not cellular telephones, the individuals involved in the telephone call would need to be located in their respective offices. On the other hand, a "casual discussion" channel might

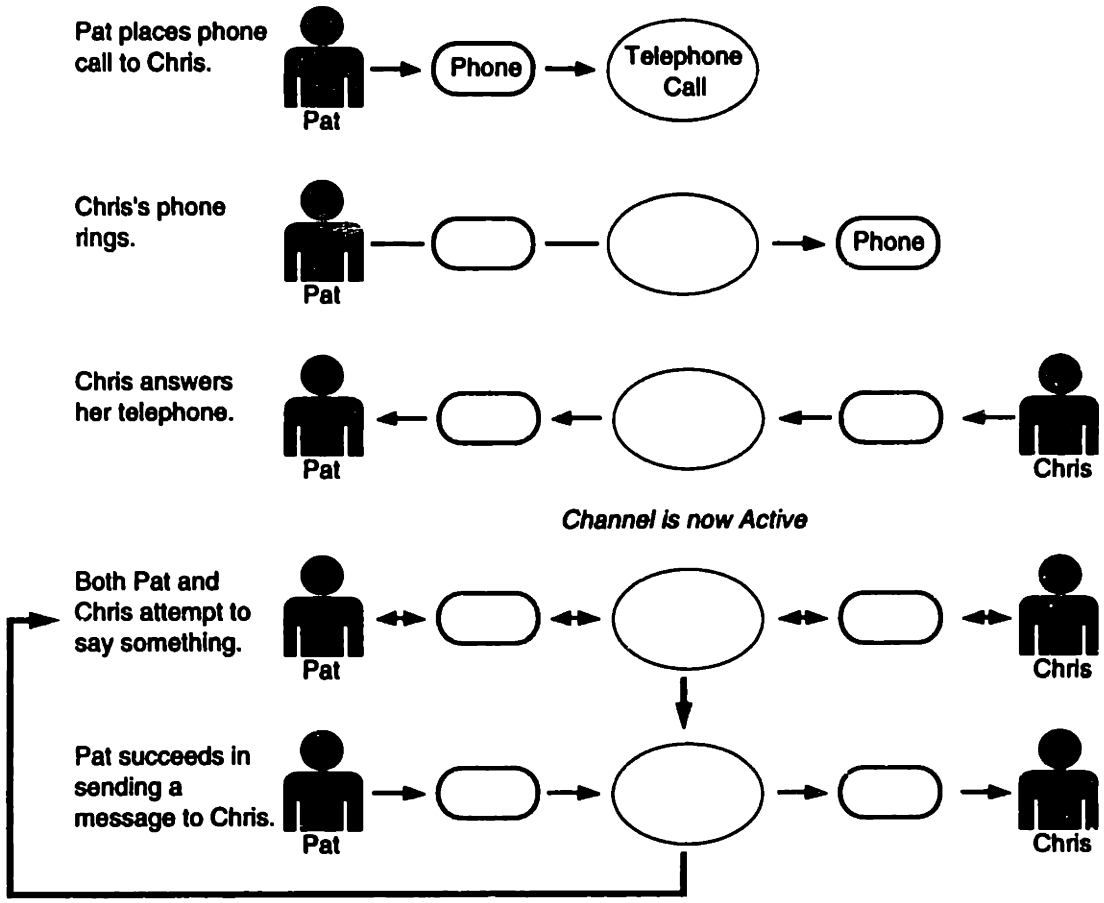


Figure 3.4: Stages of synchronous communication. The communication can be terminated at any time by either of the two participants. Note that Pat will not always succeed in being the talker.

use the "voice" equipment, require participants to be in the same office, and allow up to eight individuals to participate. The voice equipment is portable (people don't leave their mouths in their office) and the channel can be opened at any common location, so this might be a good choice of a channel for a scheduled meeting. For groups of size larger than eight, we might wish to create a special "meeting" channel to represent the more formal discussions that take place.

The **start information parameters** of a synchronous channel are used to determine how long it takes to open the channel and how individuals are informed that their presence is requested on the channel. Opening up a synchronous channel requires three steps: first, the agent must ensure that the required equipment is available for use. Second, the agent prepares the channel. For a telephone call, this consists of dialing the number of the intended recipient. For a conversation, this is equivalent to clearing your throat. The duration of this preparation phase is specified by channel parameters. Third, the agent *rings* the intended recipients a fixed number of times. At each ring, the intended recipients are informed that someone is trying to make contact with them (assuming the intended recipient is in the same office as the receiving equipment). For example, a telephone normally rings once every six seconds, and the caller will generally let it ring at least four times before giving up. Hence an agent when placing a phone call initially spends a few seconds dialing the number and then spends as long as 24 seconds waiting for the receiver to pick up. If the receiver picks up within that time period the phone conversation channel connects. If the receiving phone is busy or if the receiver does not pick up within 24 seconds, the phone conversation channel is aborted.

The **message information** parameters of a synchronous communication control how efficiently messages are transmitted over the channel. A given synchronous channel may be more

Table 3.2: Sample synchronous communication channel parameters.

	Conversation	Telephone call	Teleconference
Equipment type	Voice	Telephone	Telephone
Agents located in the same office?	Yes	No	No
Maximum number of agents	10	2	6
Conversation startup time	4 seconds	10 seconds	120 seconds
Speaking time multiplier	1.0 ×	1.1 ×	1.2 ×

or less efficient than a face-to-face conversation; the channel parameters specify how much more or less efficient. Most communication channels are less efficient than face-to-face conversations. For example, talking online via typing at a computer keyboard is considerably slower than talking face-to-face.

Table 3.2 shows sample parameters for three synchronous communication channels: a casual conversation, a telephone call, and a teleconference. The numbers picked for the parameters are representative, but not experimentally determined. A distinction is made between telephone calls and teleconferences; the telephone call is quicker to set up, but can only handle two agents at a time. Note also that the message transmission speed drops between conversations, telephone calls, and teleconferences. Two minutes worth of information to be transferred takes 132 seconds on a telephone call and 144 seconds in a teleconference. The extra time required for the teleconference over the telephone call corresponds to speaking slower and making sure that each person on the line is listening.

3.4.4 Asynchronous Communication Channels

An asynchronous communication channel is a method for one agent to send a single message to another agent without requiring the presence of the second agent. The most common types of asynchronous communication channels are faxes, voice mail, electronic mail, and inter-office mail. The advantages of asynchronous communication over synchronous are (1) the sender does not need to worry if the recipient is currently available or even in the office, (2) the recipient may attend to a message at his/her leisure, and (3) certain types of formal messages may be sent at little or no cost to the sender. The disadvantages are (1) the sender has no way of telling when the receiver will actually get the message, (2) certain types of messages may take considerably longer to send asynchronously because the sender must actually write out the content of the message, and (3) the sender cannot get an immediate reply to an inquiry. Figure 3.5 shows the stages of sending an asynchronous message.

There are two principle types of asynchronous communication channels: **formal** and **informal**. Each may handle messages only of the appropriate type; that is, a formal asynchronous channel only takes formal messages and an informal channel only takes informal messages.

What is the distinction between formal and informal messages? Most messages are informal. Formal messages represent the transfer of the written documentation generated as a part of the process of working on a task (see section 3.2.2). All other types of messages are informal.

Why are there specific asynchronous communication channels for formal and informal messages? Because parameters assigned to the asynchronous channel specify how long it takes an individual to compose and send a message of an arbitrary size. A formal message generally requires a great deal less time to compose than an informal message because the message content already exists in a written form. The parameters associated with formal asynchronous channel can be set to reflect this. Moreover, certain types of asynchronous communication channels are simply inappropriate for formal documentation. For example, it is generally not possible to transmit formal documentation via an answering machine message.[5]

[5]It would be possible to modify the *DiFS* simulation so that an asynchronous channel could specify whether or not formal documentation could be sent and what parameters would be associated with message composition in this case. I have not "improved" the simulation in this way because the present method works adequately

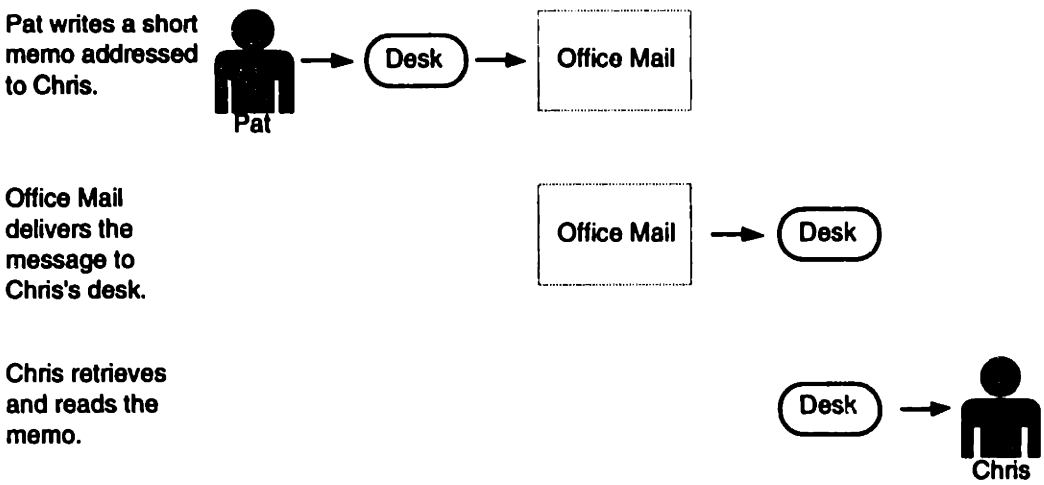


Figure 3.5: Stages of asynchronous communication. Note that the first two stages may overlap in some types of asynchronous communication (e.g., faxes or voice-mail).

There are two types of parameters associated with formal and informal asynchronous communication channels: **general information** parameters and **message information** parameters.

The **general information** parameters of an asynchronous channel are (1) the type of equipment that is used to send the message, (2) the type of equipment that is used to receive and store the message, (3) whether or not the sending person and receiving equipment must be located in the same office, and (4) the maximum size of the message. For example, a voice-mail message might be sent using the "phone" equipment, received using the "answering machine" equipment, not require the sending person to be in the same room as the receiving equipment, and carry informal messages of at most one minute in size. On the other hand, dropping a formal document on another agent's desk might be sent using the "hand" equipment, received with the "desk" equipment, require the sending person and receiving equipment to be located in the same room, and not have any limit on maximum size. In this case, the sending agent will need to be in the receiving agent's office as desks are non-portable equipment.

The **message information** parameters specify how long it takes to formulate and send the message, and precisely how long the agent, sending equipment, and receiving equipment are tied up during the transmission. These parameters are expressed in terms of a constant plus a multiple of the message size. A variety of combinations of these parameters suffices to represent the different types of asynchronous communications. For example, a voice mail message ties up the agent, the sending telephone, and the receiving answering machine for a length of time that is roughly equal to the size of the message, plus a little time for actually placing the call and listening to the recorded announcement. On the other hand, writing a letter to someone may tie up the sender for a long time before the letter is dropped into the mailbox. The letter transmission time may be on the order of days, but when it finally does show up on the recipient's desk, it doesn't tie up the desk itself for more than a second or two.

Table 3.3 shows some sample parameters associated with four asynchronous communication channels. This table attempts to give the flavor of the types of parameters associated with asynchronous communication channels; detailed information on parameters may be found in appendix A. Please note that the precise values of the individual parameters are subject to debate and should not be regarded as final.

The four sample channels of table 3.3 are divided into formal/informal channels. The Formal Fax and Drop-off channels are formal and may only transmit formal documentation. Formal documentation is information that has been generated while working on a design task that is automatically available in a written form (see section 3.2.2). Information about a task that

and I can see no great advantages to consolidating the asynchronous channels.

Table 3.3: Sample asynchronous communication channel parameters.

	Informal Fax	Formal Fax	Voice Mail	Drop-off
Sending equipment type	Fax Machine	Fax Machine	Telephone	Hands
Receiving equipment type	Fax Machine	Fax Machine	Voice Mailbox	Desk
Equipment in same office?	No	No	No	Yes
Formal document channel?	No	Yes	No	Yes
Sending agent time	1 minute +2×	1 minute	30 sec + 1.1×	10 seconds[†]
Receiving agent time	1.2×	1.2×	1.1×	1.2×
Maximum message size	Unlimited	Unlimited	5 minutes	Unlimited
	For a 5 minute message			
Sending agent time	11 minutes	1 minute	6 minutes	10 seconds
Receiving agent time	6 minutes	6 minutes	5.5 minutes	6 minutes
	For a 30 minute message			
Sending agent time	61 minutes	1 minute	Not possible	10 seconds
Receiving agent time	36 minutes	36 minutes	NA	36 minutes

[†]Travel time is not included.

generates formal documentation may be transferred over a formal asynchronous channel, up to the limits of the documentation thoroughness. Information about *any* task may be transferred over an informal asynchronous information channel, but typically such transfers take longer than the formal channel.

The four sample channels of table 3.3 also differ by where the sending agent must be located. For the first three channels, the equipment used to send the information is non-portable and hence located in the sending agent's office. The last channel, Drop-off, corresponds to an agent traveling to another agent's office and dropping a formal document on the receiving agent's desk. For this channel, the sending equipment is portable but the receiving equipment is not. Hence the sending agent must be in the receiving agent's office.

The sample transmission times and receiving times are a simplification of the actual parameters used to specify asynchronous communication channels. Note that these times are specified in terms of a constant time (corresponding to equipment preparation time) and a multiplier of the message size (corresponding to either writing the message or reading the message). The Informal Fax channel has a particularly large multiplier assigned to the sending agent; this multiplier captures the idea that the sending agent must write out the content of the fax before sending it. Formal documents have already been composed and hence the sending times are independent of the size of the message (that is, the sending time of agent; the sending time of the equipment may be longer). Once sent, both formal and informal documents are relatively quick to read.

3.4.5 Communications Behavior

As you might expect from the shear length of the preceding sections, the agents in the simulation spend a great deal of time dealing with communication issues. Each agent is issued a complement of equipment at the start of the simulation. Some of that equipment is carried by the agent and some of it remains in the agent's office. Each agent is fully aware of all of the possible methods by which information may communicated, both synchronous and asynchronous.

What does an agent do when it is time to communicate some information to another agent? The agent must select a channel, get to the correct location and secure the appropriate sending equipment. If it is a synchronous channel, the agent then attracts the other agent(s) attention, waits for a response from the other agent(s) and proceeds to converse. If it is an asynchronous channel, the agent composes the message and sends it. At all time the agent behavior must be

ready to deal with the communication not succeeding.

The fundamental behavior problem that is raised is how the agent selects a communication channel. There are many factors to consider: is the message to be sent formal or informal? Does the agent need to discuss a variety of things? How fast are the different channels? How long does it take to set up the different channels? Does the agent need to go to a particular location to initiate use of the channel? Is the current location of the recipient known? All of these factors and more go into the decision of what type of channel to use. It is important to remember here that *agents are not humans*. When choosing a communication channel, they really are just looking at the list of possible channels and picking the one that seems to have the most favorable advantages given the current situation.[6]

The other area where the agent behavior have to be particularly careful is in the middle of a synchronous communication. One of the challenges in creating reasonable personal behavior is getting the simulation agents to be polite. Humans take it for granted that people don't walk away while someone else is talking or abruptly hang up the telephone. Computer agents have to be programmed to behave in a polite fashion. Section 4.2.2 discusses the issues that arose in creating polite behaviors and how this has been resolved.

For more information on the communication model, the reader should consult appendix A for a discussion of the parameters used in the communication model and appendix B for a discussion of the software implementation.

3.5 Meetings

Agents within the simulation schedule and hold meetings for two purposes: to work on group decision tasks and to hold periodic team meetings for a management task. In the working world a great variety of methods exist by which people arrange and hold meetings. Rather than try to represent all possible scheduling mechanisms, I introduced a standard mechanism of scheduling and holding meetings.

The office building contains a single global meeting calendar that can be freely accessed by any of the agents within the office environment. On this calendar are recorded all scheduled meetings including who will be attending, where the meeting will be held, the scheduled start time of the meeting, the intended duration of the meeting, and the subject of the meeting. The subject of a meeting is either a group decision task or a periodic team meeting for a particular management task.

An agent who wishes to schedule a meeting simply informs the global calendar of the proposed task to be discussed at the meeting, the proposed duration of the meeting, the desired date, and the communication channel to be used. The global calendar selects the actual time of the meeting and the room that the meeting is to be held in, and creates a list of the appropriate agents to be invited to the meeting.

Two interesting circumstances can arise when scheduling a meeting. First, it is possible that no conference rooms are available at the desired time (or the experimenter may not have specified any conference rooms). In this case, meetings are scheduled to be held in the office of the person calling the meeting. Second, an agent may specify a synchronous communication channel uses non-portable equipment and hence does not require participants to gather at a common location. In this case, no room is reserved. Instead, agents are responsible for being in their own office at the time of the proposed meeting. The most common example of this situation is a teleconference; i.e., a multi-agent synchronous communication channel that is conducted via equipment type "telephone".

Agents pay two prices for the convenience in scheduling a meeting. First, the scheduling act itself usually takes a relatively long time, on the order of 15 minutes or more (although this

[6] A nice way to remember this is to think of all of the communications channels as being named "Channel 1", "Channel 2", and so forth. While you're at it, think of the equipment as being named "Equipment 1", "Equipment 2", and so on. This helps remind you that a channel called "telephone call" doesn't mean that it will be used like a telephone.

value may be modified by the experimenter). Second, meetings always have to be scheduled at least one day in advance. Forcing meetings to be scheduled at least a day prevents "instant" meetings from being called, but it does not preclude the possibility that several agents may gather in an office for a common discussion. It just means that they can't gather as a result of using the global meeting scheduler.

The scheduling of meetings is the primary difficulty in holding meetings in a simulation. The required agent behavior to actually attend a meeting is relatively simple. First, agents must plan on traveling to the appropriate meeting location so as to arrive there around the time the meeting is starting. Second, the agent who called the meeting is responsible for opening up the synchronous communication channel that the meeting will be held over. Third, agents in a meeting have a strong incentive to discuss affairs pertinent to the chosen meeting topic and generally remain in the conversation at least until the scheduled end of the meeting. Of course, these are only *suggestions* to the agents; agents are always free to skip meetings, depart meetings early, or simply talk about other subjects during the meeting.

3.6 Simulation Implications

The preceding sections have discussed the various elements of the *DiFS* simulation and how they fit together. Each of the preceding sections have introduced agent behaviors. These behaviors added new **actions** that agents may take; actions such as answering telephone calls or scheduling a meeting.

This section is divided into two parts; the first summarizes the possible agent actions defined by the different parts of the simulation and the second discusses limitations of the current model. Chapter 4 contains a discussion of how the computer agents have been implemented.

3.6.1 Agent Actions

At any given instant during the simulation run, an agent within the *DiFS* simulation is called upon to decide what to do. Based on the discussions in preceding sections about the office environment and individual responsibilities, we can construct a list of the different types of actions that an agent can engage in. Successfully accomplishing an action is dependent on the agent satisfying any constraints the action may have.

Work: An agent may always work on a work task that he/she is responsible for. The office environment requires that the agent be located within his/her assigned office. The information-flow model requires that the agent have satisfied all dependency requirements for working on this task.

Start a Conversation: An agent may attempt to start a conversation or meeting with one or more other agents (section 3.4.3). The agent specifies the desired people to contact and the type of synchronous channel to use.

Answer Conversation Request: An agent may answering a pending synchronous conversation request from another agent.

Join Conversation: An agent may join in any active synchronous conversation being held by two or more other agents, assuming that the correct equipment is available.

Talk: When in a synchronous conversation, an agent may at any time specify that he/she is starting to transmit a new message to the other agents in the conversation. This new message interrupts anything currently being spoken in the conversation. Normally agents attempt to talk in a conversation during a conversational pause.

Listen: An agent in a synchronous conversation who has nothing pressing to say, but wishes to allow other agents an opportunity to speak may always politely listen. Note that agents

don't have to listen in order for a successful conversation to take place. If two or more agents talk at the same time, the loudest one wins and the others are converted into "listen" commands.

Transmit: An agent may transmit a message via an asynchronous communication channel (section 3.4.4). The agent specifies the message, the channel to use, and the person to contact.

Receive: An agent who has messages received from asynchronous communications may retrieve a message and absorb its content (section 3.4.2).

Schedule: An agent may schedule a group decision meeting or a periodic team meeting (section 3.5).

Travel: An agent may travel to any office or meeting room within the office environment (section 3.3.1).

Go Home: An agent may leave the office environment and go home for the day. Traditionally, full-time agents leave at 5:00 P.M. and return at 8:00 A.M. the following morning.

Idle: When all else fails, an agent may always do absolutely nothing. Because we believe that individuals in a corporation will have tasks to accomplish other than the tasks modeled in the simulation, agents prefer to go to their assigned office to be idle.

In addition, an agent is aware of the following environmental information:

- Location within the office building. Can be at home, in a room, or in transit.
- Time of day.
- The other agents who are in the same room and what they are doing.
- The state of the agent's personal equipment if that equipment is currently located in the same room as the agent. Portable equipment is always carried by the agent; non-portable equipment may only be examined if the agent is in his/her assigned office. Agents may count the number of stored messages.
- When actively participating in a synchronous communication channel, the agent is aware of the other individuals on that channel, who is currently talking, and what they are talking about.

3.6.2 Limitations

The *DiFS* simulation implementation provides an augmented task model, office environment, communications model, and meeting scheduler. The current implementation creates a functional, but necessarily incomplete world for the computer agents. It is important to recognize the limitations of this world because these limitations affect the types of results that can be extracted from the simulation. Many of these limitations fall into the category of future work (section 9.2) and may be addressed in future versions of the simulation.

The augmented task model enhances the information-flow model by incorporating management tasks. These management tasks are used to schedule and hold periodic team meetings and to keep managers up to date on task status. They do *not* include standard managerial behavior such as handling exceptions arising from lower-level tasks or allowing the manager to reassign project personnel based on the status of the project.

An important task assumption that has been made is that there is currently no way of assigning start and stop dates to tasks. An agent's priority for working on a particular task can only be derived from the priority placed on it by the experimenter, as well as demands for information from other agents. The disadvantage of this particular assumption is that

many real projects have fixed deadlines and these deadlines drive the particular activities that engineers choose, as well as how late they will work in order to get activities done. On the other hand, specifying these deadlines in advance can be difficult and calculating simulation results even more difficult; if agents are allowed to work late to accomplish a particular task on time, the number of days taken to finish the design project is no longer a valid metric of project performance.[7]

The office environment provides locations where agents work and meet, as well as a limited model of traveling between various locations. It does not provide for multiple office buildings or running into agents while traveling the corridors. There is no lunch room or common location where agents go to socialize. Agents are assumed to always work in their offices; this precludes multiple offices or working on the shop floor.

The communications model is flexible and allows the person setting up the simulation to specify how agents may communicate and how effective each of these communication methods are. On the other hand, it does not explicitly control how different types of project documentation are to sent; for example, some tasks are best represented by blueprints, others by memos, and still others by spreadsheets. Each of these types of documentation potentially could be transmitted to other agents at different rates.

The scheduling of meetings is much simpler than what occurs in many corporations.

3.7 Summary

This chapter has described a method of extending the information-flow model to include an augmented task model that provides basic managerial structure, an office environment where the computer agents work, a communications model for how agents may exchange information, and a method of scheduling meetings.

The augmented task model adds management tasks to the *DiFS* model. Management tasks serve two functions: they provide a mechanism for team meetings and they impose a hierarchical structure on the design project that allows for limited managerial behavior. The hierarchy of management tasks simplifies the determination of when a design project has completed.

The office environment provides offices for the agents and conference rooms for holding meetings. Agents may travel around the office environment to contact other agents. The walls of the office environment limit agents to seeing only other agents that share the same room.

The communications model provides a flexible method of specifying how agents may exchange messages with other agents. The communications model is composed of three parts: equipment, synchronous channels, and asynchronous channels. An agent accesses communication channels through his/her personal equipment. Synchronous communication channels connect two or more agents in a common discussion. Asynchronous communication channels allow an agent to send a message to a piece of equipment owned by a different agent. The agent receiving an asynchronous message may retrieve it at his/her leisure.

The global meeting scheduler is an office-wide bulletin board that is used to schedule group decision meetings or team meetings. It provides a common interface for the agents to set up meetings at the cost of requiring meetings to be scheduled at least one day in advance.

These extensions to the *DiFS* model place the computer agents in a realistic office setting so that they may interact with each other and work on their assigned tasks. How the agents decide what to do at any given point in time is the subject of chapter 4.

[7] Adding task deadlines is high on my list of simulation enhancements.

Chapter 4

Personal Behavior

This chapter of the thesis describes the implementation of the personal behavior model of the computer agents within the *DiFS* simulation. The personal behavior model of a computer agent is the "brain" of the agent. The personal behavior model translates stored knowledge, awareness of the local environment, and short term memory into decisions of what the agent should do.

Unless otherwise stated, throughout this chapter the generic term *agent* will be used to mean the personal behavior model used to control the actions of each computer agent. For example, "an agent possesses current knowledge of the state of each task he/she is responsible for" means that the personal behavior model of the individual agent has immediate access to, or stored information on, the current state of each task that the agent is responsible for.

This chapter is divided into a number of sections that explain the agent personal behavior model: a description of what the behavior model must accomplish, message types and conversation behavior, the scripts used to translate basic goals into actions, the mental model each agent builds of the simulation world, how agents decide what to do, and a summary of the assumptions and implications of the behavior model.

4.1 Requirements

Researchers have considered the question of the required capabilities of a social agent working within an organization (sometimes referred to as an organizational agent). According to Carley [12], a general social agent must exhibit the following six basic capabilities: (1) perceive the environment and take action, (2) have a memory of the perceived environment, (3) be able to follow instructions, (4) analyze a task and decompose it into subtasks to do, (5) communicate with other agents, and (6) analyze the social environment within which it operates.

Agents within the *DiFS* simulation meet the criteria for a general social agent. They perceive agents and equipment located in the office they currently occupy, and remember past locations of other agents. They do not explicitly instruct each other, but do follow the requirements for accomplishing the design project by selectively working on assigned tasks and communicating with other agents. Their analysis of the social environment is limited to following conversational etiquette and deciding how to respond to inquiries. How agent behavior criteria is formulated and how it is implemented is the subject of sections 4.1.1 and 4.1.2.

4.1.1 Modeling Objectives

The personal behavior model of an individual agent is called upon to specify exactly what the agent will do at any given moment in time in the simulation.

The information-flow model specifies that agents do exactly one thing at a time, are not telepathic, and should exhibit bounded rationality. The *DiFS* simulation specifies the types of

activities that the agent can perform (section 3.6.1). Given these requirements, we can formulate a list of the basic behaviors the agent should exhibit:

- An agent attempts to make good progress on all assigned tasks.
- An agent goes home every day at the appropriate time and show up for work the next morning.
- An agent requests information about tasks from other agents and responds to requests for information.
- An agent attends scheduled meetings and discusses topics appropriate to that meeting.
- An agent who is responsible for a group decision task or management task schedules meetings when appropriate.
- An agent retrieves and reads messages send to him/her asynchronously.
- An agent is capable of holding a conversation with one or more other agents.
- An agent can always find something to do, even if that means wasting time.

The requirements specified for agent behavior by the information-flow model and the *DiFS* simulation do not cover all of the behaviors that are present in a real corporation. The following is a list of desired agent behaviors that have been added to the basic behavior list.

- Agents should be respond to priorities placed on tasks in a meaningful way. If the person setting up the simulation specifies that a particular task is more important than other tasks, then agents should do their best to get that task done as quickly as possible.
- Agents should be able to influence each other. For example, frequent requests for information (nagging) should have an effect on the agent's priority of replying.
- Agents should behave in a manner consistent with common office decorum. An agent should answer the telephone if it rings and hold polite, but efficient, conversations.
- Agents should be influenced by the convenience of activities. For example, two agents sharing a common room are more likely to start a conversation than if they are in separate rooms.

The current implementation of the personal behavior model of the agents incorporates all of the above behaviors.

4.1.2 Approach

The personal behavior model can be divided into the part that stores what an agent knows and the part that translates knowledge into action. This section provides a short summary of the reasoning portion of an agent's personal behavior model. The general structure of the reasoning portion of a computer agent's personal behavior is laid out in figure 4.1. The reasoning process proceeds at three different time scales: planning, objectives, and actions.

The **planning** level of the reasoning process creates a simple plan that specifies what tasks the agent should concentrate on for the current day. An agent creates a new plan every morning. The plan specifies about how much time the agent intends to spend working on each task that the agent is responsible for. This plan is constructed based on the available time in the day that is not reserved for scheduled meetings, the current states of the tasks the agent is responsible for, and the priorities assigned to each of the tasks.

The **high-level reasoning** stage translates an agent's knowledge, memory, and environmental awareness into a current **objective**. Objectives are simple goals; they generally last

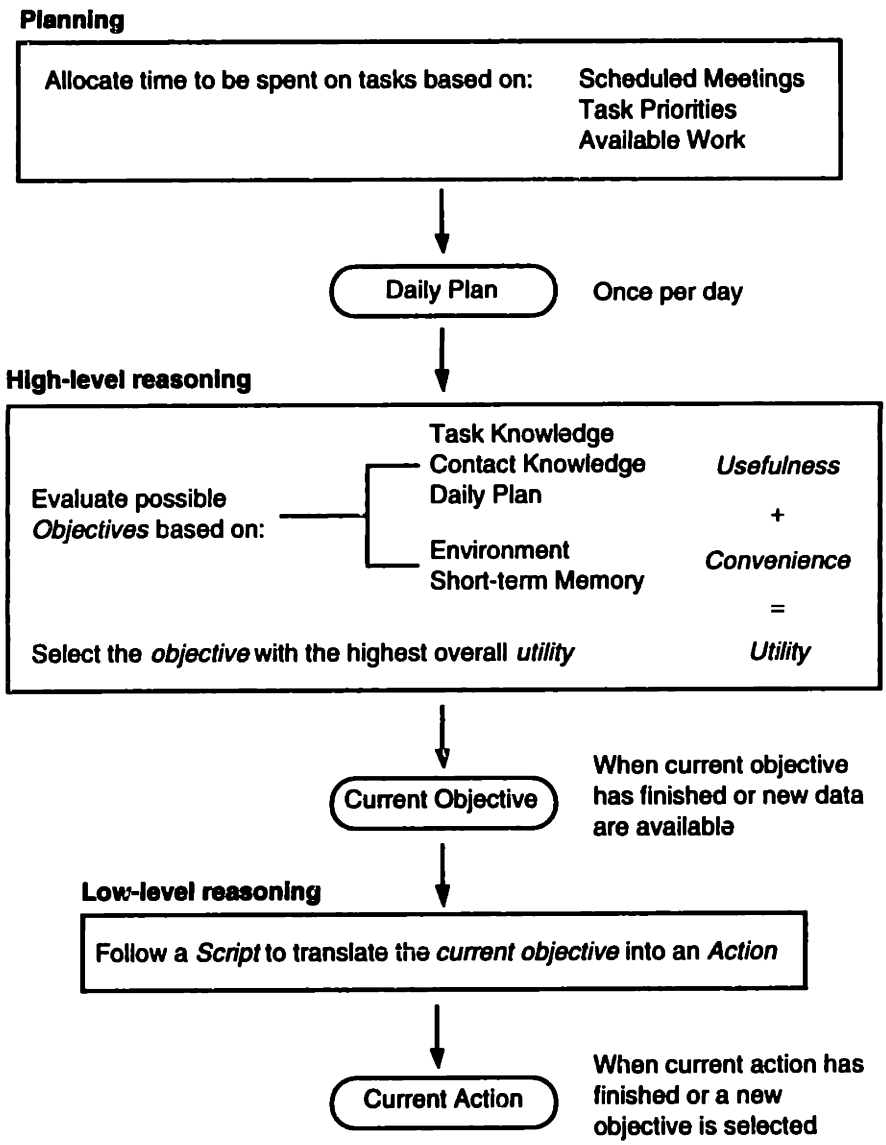


Figure 4.1: Block diagram of how agents reason

between a few seconds and a few tens of minutes. Sample objectives are: working on a task for fifteen minutes, attending a meeting, going home, contacting another agent, or answering the telephone.

When deciding upon a new objective, the agent evaluates all possible objectives and selects the one with the highest perceived **utility**. The utility of an objective is the agent's assessment of how **useful** the objective is and how **convenient** carrying out the objective will be. The usefulness of an objective is calculated based on the agent's daily plan, knowledge of tasks, and knowledge of contacts with other agents. The convenience of an objective is calculated based upon environmental factors and short-term memory. Sample environmental factors are: who is in the room with the agent, whether or not any telephones are ringing, and where the agent is currently located. Sample short-term memory factors include what the agent has just been doing and whether or not a question was recently asked of the agent.

The **low-level reasoning** stage translates the agent's current objective into the specific actions that are needed to carry it out. This translation is accomplished by the use of **scripts**. A script is a simple plan that specifies what an agent should do in order to carry out an objective.

The overall structure of the reasoning process is the same from agent to agent. However, stored knowledge, short-term memory, planning, and environmental awareness are different for each agent.

The overall combination of planning, high-level reasoning, and low-level reasoning creates agents that can decide what to do and carry it out in an efficient manner. Of course, there is a lot more to the agent behavior than a simple block diagram. Agents must communicate with each other effectively: this requires specification of the types of messages that are sent, how the agents coordinate their communication, and how received messages are acted upon. Agents must time their activities correctly so that they request information when it is likely to be available, provide information as other agents need it, and accomplish tasks in a reasonable order.

Issues of effective communication, timing, and action all come down to (1) storing the right knowledge and (2) calculating appropriate utilities of the various objectives that the agent can do. The organization of this chapter is designed to lead the reader through the parts of the personal behavior model that support or carry out the evaluation of utility before discussing the utility calculations themselves.

4.2 Communicating

Chapter 3 contains a description of the communication channels that agents use to exchange messages. This section discusses the types of messages that agents exchange and some simple rules of conversational etiquette. The different types of messages are used to convey information, decisions, and requests. Conversational etiquette is the balancing of agent behavior in conversation so that each agent is given an opportunity to speak.

4.2.1 Messages

Messages are how agents in the *DiFS* simulation communicate with each other. They are the *only* way that agents may communicate with each other. A message represents in an abstract sense a transfer of information. It can take written or oral form, depending on the underlying channel it is transported by. The simulation implements only a minimal number of message types: information transfer, requests, decision making, and gossip.

All messages are measured by the following standard parameters: **size, delivery times, source**, and **intended recipients**. The size of a message is how long it would take to be transmitted in a face-to-face discussion. The actual transmission time will vary based on the type of channel used to send the message. The delivery times of a message are timestamps of when the message was sent, when the message was delivered, and when the message was received. The source of a message is the agent who sent it. The intended recipients is a list of all the agents the message is being sent to. This list includes only the agents who are being sent the message simultaneously (as in a synchronous communication channel), not agents who will be sent the same message at different times.

Four types of messages are used to communicate between agents: information, requests for information, decisions, and gossip. These four types of messages serve to transmit information and information requests, post status updates, do group decision making, and lubricate conversational exchanges. Each type will be described in greater detail.

An **information message** is a message that contains information generated by a work or group decision task. An agent may compose and send an information message about any task in the design project up to the limits of his/her personal knowledge. An information message contains the following parameters: the task being discussed, the known completion level of the task, the information being transmitted, when the last time the person was updated on the status of the task, and (if the task has not yet started) when the task is expected to start.

The information being transmitted within an information message is a complex piece of data representing the particular portion of the information solid (see section 2.1.5) being sent. Think

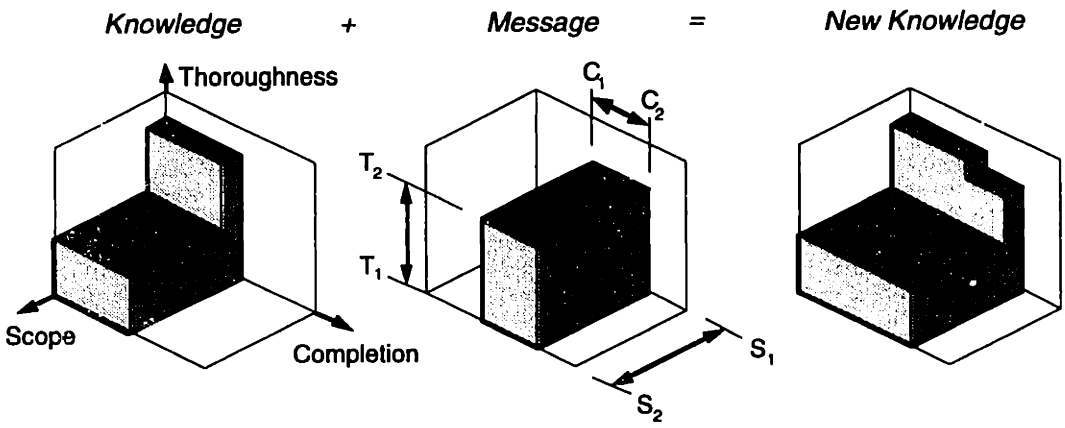


Figure 4.2: An example of an information message being added to an agent's task knowledge.

of it as a non-uniform, three-dimensional solid in task scope, thoroughness, and completion (see figure 4.2). The simplest form this data can take would be a rectangular solid consisting of:

$$\{(C_1, C_2) \times (T_1, T_2) \times (S_1, S_2)\} \tag{4.1}$$

where C_i are the lower and upper bounds of completion, T_i are the lower and upper bounds of thoroughness, and S_i are the lower and upper bounds of scope. The actual information transmitted is the union of one or more of these simple information solids.

A **request for information** is a message that informs the recipient that the requesting agent would like an information message in return. Requests are normally sent directly to the agent responsible for a task. A request for information contains the following parameters (in addition to the standard message parameters): the task, the desired information to be received, the priority of the request, and whether the requesting agent would like periodic updates of task status.

The desired information parameter of a request specifies the information solid that the requesting agent would like to receive in reply. This does not mean that the agent will only be satisfied with receiving a complete information solid in return. It only states that *ultimately* the agent will not be satisfied until this quantity of information has been received.

A request for information may also be used to indicate that the requesting agent is not interested in the details of the task, but would like a report of the current completion level of the task. Usually this type of message also specifies that the requesting agent would like status updates. Requesting status updates tells the recipient of the message that the message sender would like to be periodically informed of the task's status. Status updates are useful for agents who are responsible for management tasks.

Decision messages are a special type of message that represent work being done on a group decision task. Decision messages may be sent only in synchronous conversations where all group members are present. Decision messages are sent by the agent who is responsible for the group decision task. When a decision message has been successfully received by all of the participants in the group decision task, each agent adds the size of the decision message to his/her personal knowledge of the work done on the group decision task.

One might well inquire how a decision message can possibly reflect the active discussion that goes on in a group meeting. In fact, we assume that the group of agents is actively discussing and debating the decision task and its issues. The decision message is a convenient fiction that encapsulates the group discussion into a single message sent out by the responsible party. This encapsulation explains why all participants of the group decision task must be present in the conversation.

There are just two parameters associated with a decision message (beyond the standard parameters): the task being discussed and the new completion level. The new completion

level is where the progress on the task will advance to assuming that the decision message is successfully sent to all participants of the group decision task. If any participant interrupts, drops out, or otherwise disrupts the message, the message is considered lost and no work is recorded. Hence it is advantageous for decision messages to be kept relatively short.

The final type of message is **gossip**. A gossip message is random chatter sent by an agent. It carries no information content: fishing stories and golf scores have no simulation relevance. Agents generally gossip with each other when they don't have anything of relevance to say, but they desire to remain in conversation. As a rule, gossip is not sent over asynchronous communication channels.

4.2.2 Conversation Etiquette

Section 3.4.3 discussed the basic format of a conversation between several agents. A conversation is typically in one of two states: either someone is actively talking and the remainder are listening, or there is no current talker and a new talker must be selected. This new talker is selected by letting all agents specify what they would like to say (if anything) and how emphatically they would like to say it (by specifying the *urgency* of the communication). The agent with the most urgent message becomes the new talker.

The challenge with a multi-agent conversation is letting all of the agents have a chance to say things without any one agent monopolizing the conversation. Hence the term *conversation etiquette*. The flexibility of the *DiFS* conversation model introduces the potential for abuse. In early versions of the simulation software, I have watched conversations where one agent would ask the same question over and over again, or an agent would ask a question and then walk away without waiting for a reply, or one agent would gossip away without letting other agents say anything. In short, proper conversation etiquette is essential to a well behaved simulation.

The three factors that determine the success of a multi-agent conversation are (1) how emphatically an agent speaks, (2) how an agent selects an appropriate message, and (3) how an agent determines when a conversation is over.

The urgency of a message being spoken by an agent controls how likely that this agent will control the conversation and become the talker. The basic urgency of a message is proportional to the perceived utility of the message to the agent sending it. Utility calculations will be discussed in section 4.5. Added to this basic urgency there are two simple social rules: First, gossip always is assigned a lower urgency. Second, if the agent is the last agent to have talked in the conversation, then the urgency of his/her message is lowered. This reflects the idea that there should be give and take in the conversation.

The appropriateness of messages that an agent selects is calculated by comparing the utilities of the various possible messages (again, see section 4.5). Normal conversation convention suggests the following basic guidelines that are followed: First, messages should be relevant to the topic of conversation. In a group decision meeting, requests and information should be regarding the task being discussed or tasks that drive the task being discussed. In a scheduled team meeting about a management task, requests and information should be about tasks within that management task. Second, an agent who has just received a request for information should attempt to answer the request. Third, if an agent has just sent a request for information, then it is appropriate to give the receiving agent an opportunity to answer the request before asking for the information a second time.

Determining when a conversation is over can be tricky. If each agent left the conversation when he/she ran out of things to say, then other agents might not have the opportunity to ask questions or transfer information. The solution to this difficulty is to have agents attempt to send gossip messages in a conversation when they run out of things to say. The urgency of gossiping is always lower than that of asking questions or answering requests. The total amount of gossip spoken in a conversation is kept track of; the more gossip, the less useful the conversation becomes and eventually all of the agents leave the conversation.

Balancing the behaviors of agents to achieve a functional conversation is not easy, but

it does result in efficient and interesting conversations. The generality of the conversation mechanism provides some interesting opportunities for studying what happens as agents become less effective at exchanging information. These effects will be discussed in later chapters (see particularly chapter 8).

4.3 Agent Objectives & Scripts

Objectives and scripts form the base of the agent personal behavior model. The high-level personal behavior model of the agent translates personal knowledge and environmental awareness into a current objective. Low-level agent behavior utilizes a script to translate an objective into specific agent actions. The advantage of dividing between high and low-level behavior is two-fold: it simplifies the computer implementation and it permits the easy extension of the simulation to incorporate new objectives and scripts.

A **script** is a psychological term for behaviors that people use to translate simple goals into actions [1, 7]. For example, most humans follow a script to perform a simple daily activity such as brushing one's teeth. After ten or more years of brushing your own teeth, it takes virtually no conscious effort on your part to wander into the bathroom, grasp your toothbrush, apply toothpaste, brush, and rinse.

Agents in the *DiFS* simulation have a set of scripts that translate objectives into a series of actions. For example, consider an agent who has decided to attend a meeting. Attending a meeting requires the agent to perform the following actions: travel to the location the meeting will be held, wait until the agent in charge of the meeting shows up and calls it to order, and join the common meeting conversation channel. This series of actions performed by the agent is collected into a single logical entity: the "attend meeting" script.

The following is a list of the objectives (and corresponding scripts) currently available to agents in the simulation:

Waste Time: The agent kills a specified amount of time. Agents prefer to waste time in their own office.

Go Home: The agent leaves the office building, goes home, and doesn't return until the following morning.

Read Mail: The agent selects a waiting messages, retrieves it, and reads it.

Answer Interrupt: The agent enters into a conversation. This script is only available if the agent is being "rung"; that is, someone is trying to engage this agent in a synchronous conversation.

Solo Work: The agent attempts to work on a task for a particular length of time, normally around 15 minutes.

Send Request: The agent sends a request for information about a task using an asynchronous communication channel.

Send Information: The agent sends information about a task to the appropriate agent using an asynchronous communication channel.

Contact Person: The agent attempts to contact another agent via a synchronous communication channel.

Attend Meeting: The agent attends a meeting. This script controls getting the agent to the meeting and joining in the discussion. It does not control the conversation at the meeting.

Schedule Meeting: The agent schedules a meeting for a management task or group decision task.

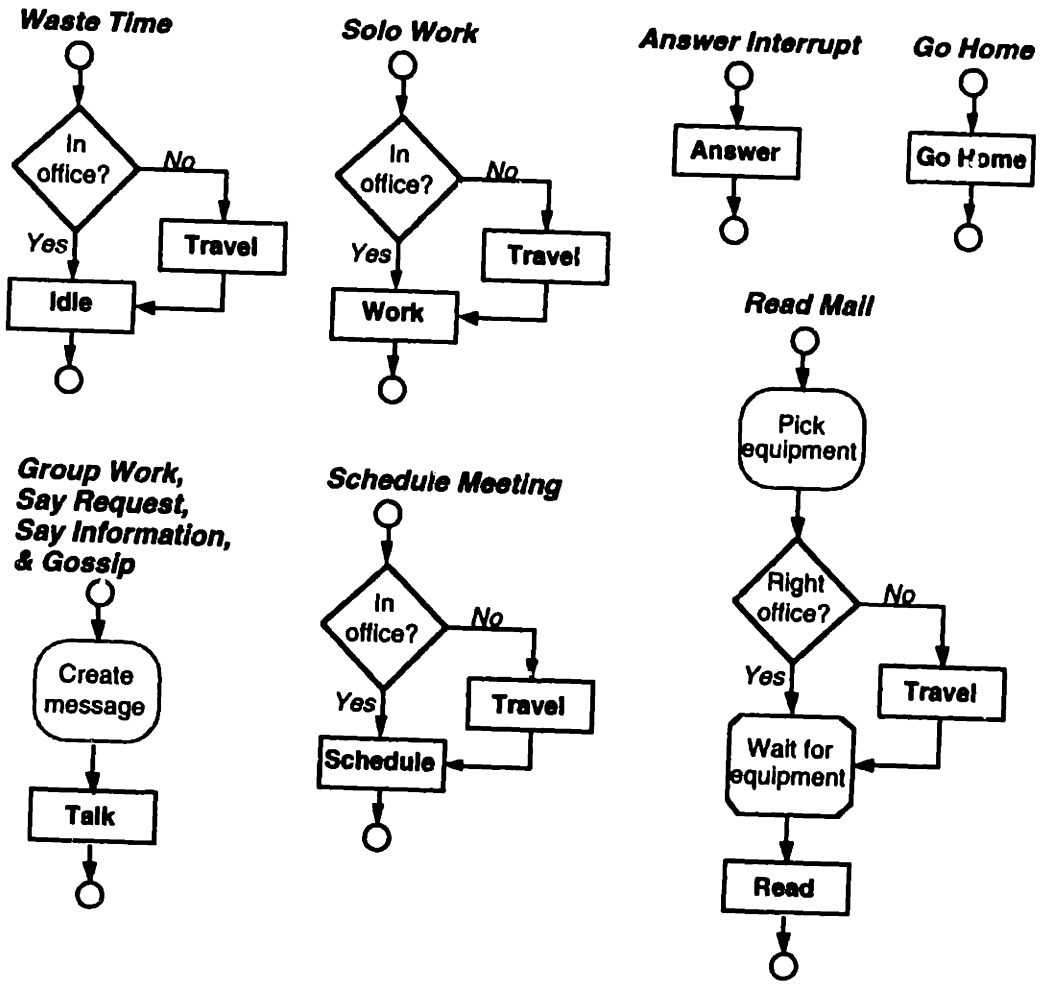


Figure 4.3: Behavior script flow diagrams, part 1/2.

The following objectives are appropriate only when an agent is engaged in a conversation:

Group Work: The agent attempts to send a decision message to the group.

Say Request: The agent attempts to send a request for information about a particular task to one agent in the conversation.

Say Information: The agent attempts to send information about a task to the group.

Gossip: The agent attempts to gossip.

A few things should be noted about certain scripts. First, when selecting a Send Request or Send Information objective, the agent does not specify the asynchronous channel to use. The script contains an algorithm to select the specific channel to use. This algorithm draws upon the knowledge of the agent, the current location of the agent, and other environmental factors. Similarly, when selecting the Contact Person script, the agent does not specify the synchronous channel to use. Second, when selecting a Send Information or Say Information objective, the agent does not specify how large the information message should be. As a part of the script, an appropriate message size is selected and dispatched.

Figures 4.3 and 4.4 are graphical representations of the script flow diagrams. The rectangular items represent actions to be taken by the agent (and hence take time). Items enclosed in octagons are *waits*; an agent upon reaching this point in the script will wait until the requirement in the box is satisfied. For example, in figure 4.3 the *Read Mail* script has a "wait for equipment"

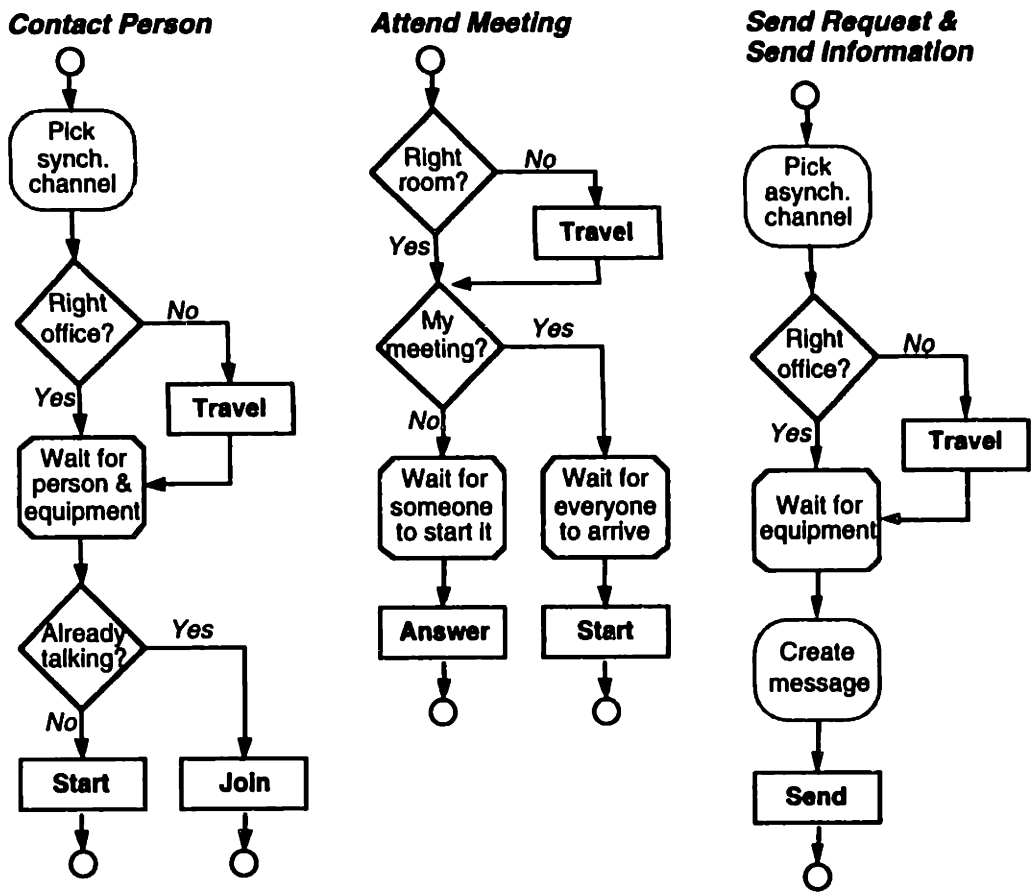


Figure 4.4: Behavior script flow diagrams, part 2/2.

box. At this point in the script, the agent will check to see if the necessary equipment is available for use. If it is not available, the agent will issue an *idle* action and continue to check the equipment until it becomes available. I should mention that any of these scripts can be interrupted at any point by an outside disturbance (such as another person walking into the office). Moreover, a script may also be interrupted by a *time out*. If the agent has to wait longer than a reasonable amount of time for a piece of equipment to become available or another agent to respond, then the script is canceled and the agent selects a new objective.

4.4 Mental Model

To successfully interact with the simulation world, an individual agent possesses a memory. The agent must remember facts about tasks: how much work has been done on them, what information is possessed about a particular task, who is responsible for what, and which tasks are waiting for information. The agent must remember facts about other agents: who has requested information, who the agent has sent requests to, and the last time a particular agent was seen.

A mental model is a psychology term for essentially a replication of the world inside a person's head. People create little models of the world in their brains because it allows them to predict what will happen. For example, the reader probably has a mental model that can predict what will happen if a full coffee mug was tossed into the air.

DiFS agents need to store a variety of pieces of information about tasks and other agents. I refer to this stored information as the agent's **mental model** of the design project. It qualifies as

a mental model and not just a collection of facts because (1) the structure used to store the data is deliberately similar to the project representation, and (2) the agent uses an understanding of the relationships specified by that structure to make predictions about the simulation. These predictions are simple ones, having to do with whether or not information will be available or if another agent will be in his/her office.

I should note that the *structure* of the mental model is the same from agent to agent, but the actual *contents* of the mental model differ from agent to agent. Each agent has his/her own knowledge of tasks and contacts. Knowledge possessed by one agent is never directly shared with another agent. The only way to exchange any portion of the knowledge an agent possesses is by sending an information or decision message.

The mental model used by agents has evolved as the simulation increased in complexity and realism. The original mental model used by agents was deliberately kept simple. As behavior problems were identified and corrected, the mental model added new types of knowledge and new ways of reasoning. The current mental model used by the agents is difficult to describe briefly. Hence it will be described in pieces: task knowledge, contact knowledge, and planning.

4.4.1 Task Knowledge

The core part of an agent's mental model is his/her **task knowledge**. The task knowledge of an agent stores what an agent knows about work tasks, group decision tasks, and management tasks. Task knowledge can be divided into tasks that an agent actively works on and tasks that an agent is interested in. Tasks that an agent has no involvement with are not represented. Figure 4.5 is a sample of what one agent knows about the human-factors project of figure 2.1 at one point in time. Note that agents within the simulation will know different things.

There are three reasons why an agent would be interested in a task: (1) The agent has a role on that task, either as the responsible party or as an assistant. (2) The task drives a task that is the agent's responsibility and hence the agent will need to acquire information from it. (3) The task is a part of a management task that the agent is responsible for, and hence the agent will need to get status reports about the task. An agent may be interested in a task for one or more of these reasons.

For each task that an agent cares about, there are a few standard parameters that specify what the agent knows and what the agent is interested in. For a management task, the agent keeps track of when the next scheduled meeting is and the overall state of the management task (has not started, actively progressing, or finished). An agent who is responsible for a management task will use this information to decide when to schedule the next team meeting.

Work and decision tasks have a large number of parameters associated with them, and these parameters may vary depending on if the task is one that the agent is responsible for, is an assistant on, or is simply interested in. The most important parameters measure the **information** known about a task. An agent may possess three types of knowledge of the information generated in a work or decision task. First, an agent may know the **status** of the task (i.e., how complete it is). Second, the agent may possess **regular knowledge** in the form of an information solid, measured by scope, thoroughness, and completion. This information is known by the agent and in some sense is stored in his/her head. Third, the agent may possess formal or **written knowledge** of the task. This is information that is stored in a document of some kind and hence is easy to transfer to other agents (see section 3.2.2). At all times an agent's formal knowledge will be a subset of his/her regular knowledge.

If the work or decision task is one that the agent has a role on, then the agent will automatically have accurate status, formal, and regular knowledge. However, if the agent does not have a role on the task, then all knowledge must be obtained from another agent, and hence the recency of the knowledge becomes important. For these tasks, the agent keeps track of *when* the knowledge dates from in the **last updated** field of the task. The "last updated" refers to the time that the original source of the information sent it out. For example, assume Agent 1 is responsible for a task and sends a status report on it to Agent 2. Agent 2 receives the status

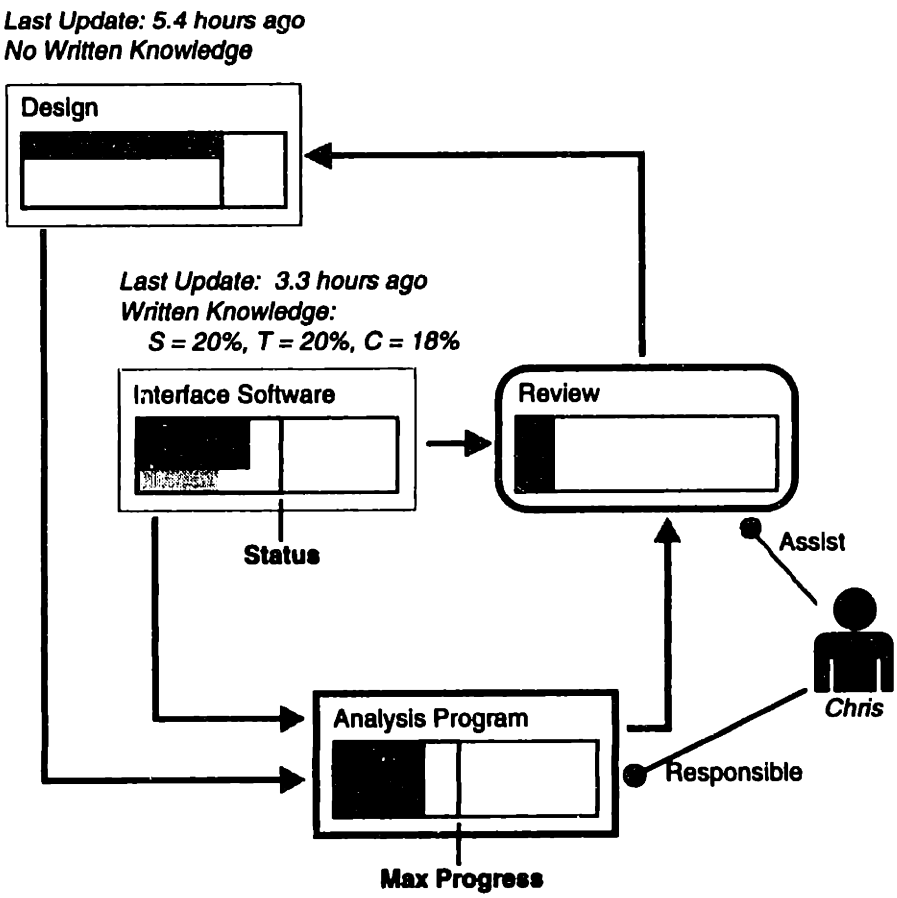


Figure 4.5: Sample task knowledge of a single agent. Width of box represents completion knowledge, height is scope, and darkness is thoroughness. This example is drawn from the human-factors experiment shown in figure 2.1.

report two days later and records in her memory that the information on the task was last updated as of two days earlier.

Given the last updated information stored about tasks that an agent is interested in, an agent can reason about when to ask for new information about a task. For example, if information was last updated just a few minutes earlier, an agent can reason that it is unlikely that new information will be available.

However, there is the question of when to first inquire about a task. Given a complex project with hundreds of tasks and complicated dependencies between the tasks, the tasks can be roughly ordered by when they are likely to be able to start. Agents maintain an estimate of when a task is likely to start so that they do not waste time requesting information before it could possibly have been generated. Preliminary start estimates are created by all agents at the beginning of the simulation. During the simulation run, as information is gathered by agents it is propagated through the agent's model of the project network and used to update task start time estimates.

Figure 4.6 demonstrates how task start times may propagate based on the arrival of new knowledge. The start time propagation algorithm is simple; it assumes that every other agent works all the time on every task they are responsible for. The propagation algorithm does not consider dependencies that the agent doesn't have in his/her mental model, does not consider meetings that other agents might need to attend, and does not consider time lost to working on multiple tasks or transmitting information. It is an optimistic propagation algorithm that continually updates start time estimates as new information is acquired by the agent.

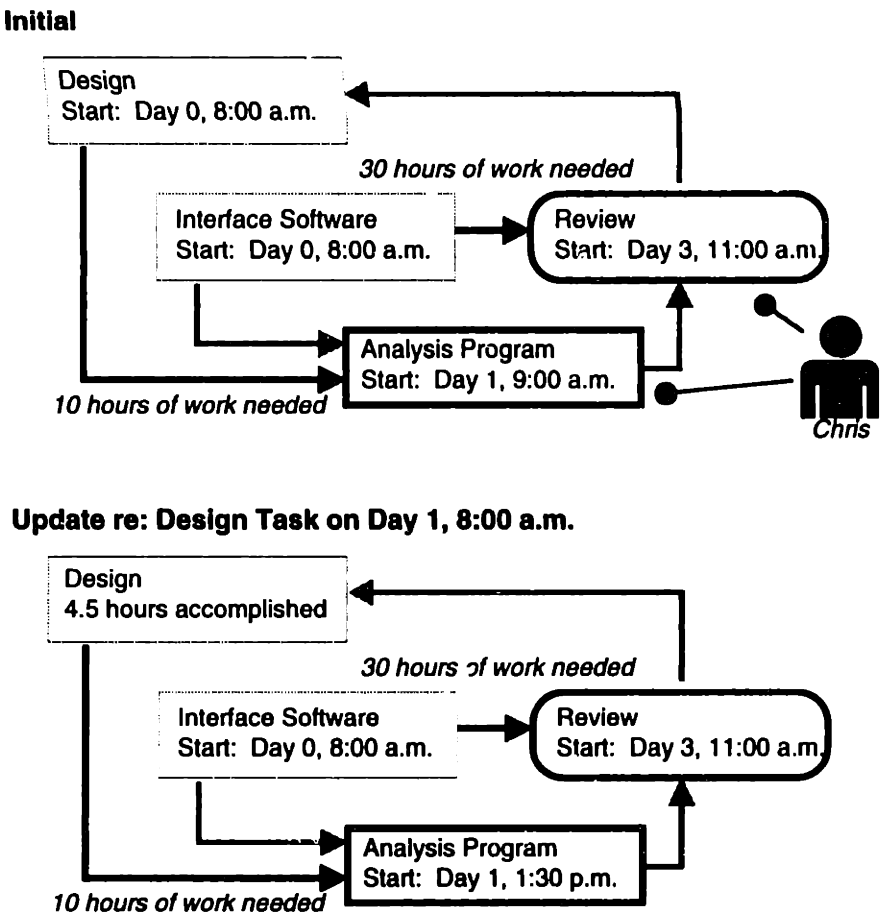


Figure 4.6: Sample start-time propagation. Dependencies drawn in black require work in the upstream task before downstream task can begin. All estimates ignore information transfer times. Note that Chris is unaware of the dependency connecting DESIGN to INTERFACE SOFTWARE.

4.4.2 Contact Knowledge

The mental model of task knowledge presented in the prior section is used as the base for the contact knowledge of an agent. Contacts are other agents in the simulation that an agent either acquires information from or sends information to. A contact who supplies information to an agent is referred to as a **source contact**. The connection between the task the information is about and the contact who supplies the information is called a **source**. A contact who receives information from the agent is referred to as a **sink contact**. The connection between the task that provides the information and the sink contact is referred to as a **sink**. Note that a contact may be regarded as both a source and a sink contact for different tasks.

Figure 4.7 is a sample of the types of contacts, sources, and sinks that may occur in a mental model. In this case the agent Chris has two contacts: Pat and Sandy. Pat is a source of information about the INTERFACE SOFTWARE task. Sandy is a source of information about the DESIGN task and a sink of information for the ANALYSIS PROGRAM task that is Chris's responsibility.

Parameters associated with contacts, sources, and sinks keep track of the agent's interactions with other agents. The most important type of interaction to keep track of is the types of messages that have been sent to and received from contacts, and when these messages occurred. Of secondary importance is tracking when agents have last been seen and where; this information turns out to be useful when an agent is trying to decide how convenient it will be to attempt

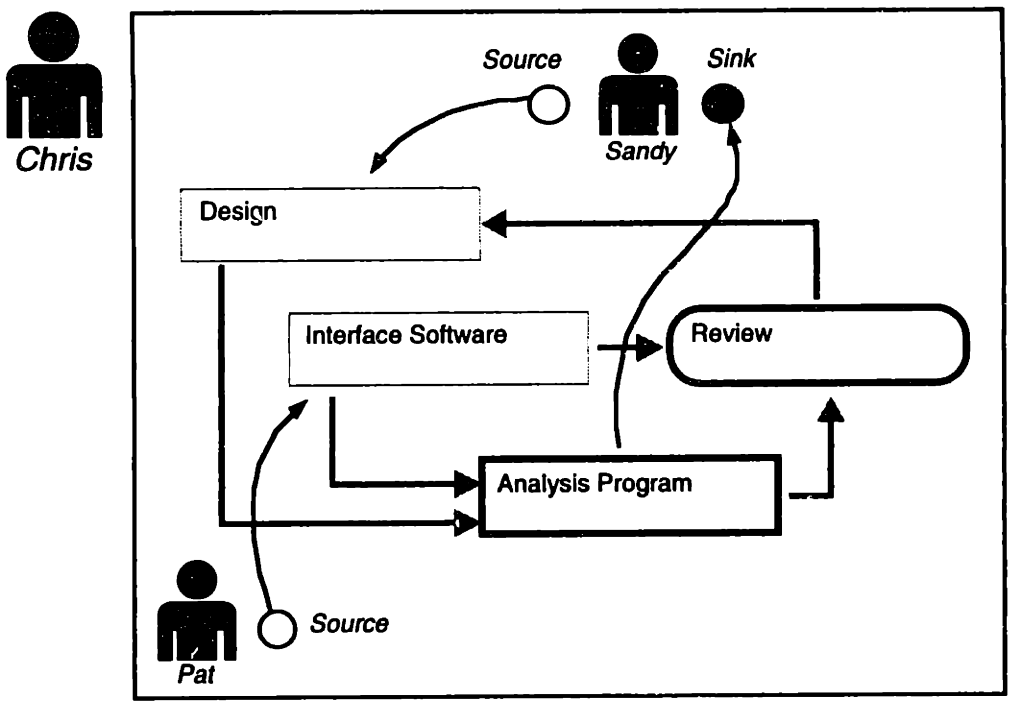


Figure 4.7: Sample contact knowledge. Here Chris is interested in learning about the DESIGN task and the INTERFACE SOFTWARE task. Chris must inform Sandy about progress in the ANALYSIS PROGRAM. Chris is only a participant on the REVIEW task and hence has no sources or sinks attached to this task.

to locate another agent.

The following information is stored for each sink: when the sink contact last sent a message requesting information, the last time information was sent to the sink contact, how many outstanding requests for information have been received without being answered, and the actual information that has been requested. If the sink contact has indicated that he/she would like status updates, the last status level sent is also recorded.

The agent uses the information stored in the sink to evaluate the utility of sending information to the sink contact. For example, if there is at least one request that has been received but not answered, and new information is available, then the priority of replying to the request is relatively high.

The following information is stored for each source: when the last request for information was sent to the source contact, how many requests have been sent without being answered, and when the last information message was received.

The agent uses the information stored in the source to evaluate the utility of sending a request for information to the source contact. For example, if the last time the agent received information from the source contact was at least a day or two ago and no requests have been made of the task recently, then the utility of requesting information is relatively high.

Additional information is stored about each contact that helps the agent to decide whether or not it will be easy to reach the contact via some kind of synchronous or asynchronous communication. The following information is recorded for contacts: the last time the contact was *seen* (i.e., in the same location as the agent), the last time the contact was seen in his/her office (i.e., where receiving equipment is generally located), and the last time a message was successfully or unsuccessfully sent by an asynchronous communication channel.

The contact information is helpful when the agent is trying to decide whether or not it will be easy to communicate with the contact. For example, assume Sue is trying to open a communication channel to Joe. If Sue decides a phone call is the best method, calls Joe, and

gets no answer, then Joe is probably not in his office and hence it is not advantageous for Sue to travel to his office to talk to him in person.

4.4.3 Planning

At the start of the work day an agent forms a tentative plan for roughly how many hours are to be spent working on each of the available tasks. This planned work time is stored in the agent's task knowledge. During the course of the work day, the agent keeps track of how much work has been accomplished in each planned task. This is used in utility calculations to help decide how important it is to work on a particular task.

The formula used for planning the work an agent is to do is relatively simple. Each task the agent is responsible for submits a bid for how many hours of work should be done on it, based on how many hours *can* be done given information limitations. The agent attempts to allocate the available hours in the work day by minimizing a cost function that combines the priorities of the tasks with the possible work hours.

To be specific, assume that there are N tasks an agent is responsible for, and H hours in the day available for work (after taking into consideration meetings). Assume that each task has h_i hours of work that can ideally be done on it, given dependency limitations. Assume also that each task has a weight w_i assigned to it that reflects how important working on this task is. Then the goal is to assign planned hours h_i^* to each task so as to minimize the cost function C

$$C = \sum_{i=1}^{N} w_i (h_i - h_i^*)^2 \tag{4.2}$$

subject to:

$$\sum_{i=1}^{N} h_i^* \leq H \tag{4.3}$$

$$0 \leq h_i^* \leq h_i \tag{4.4}$$

This minimization problem is solved by a standard Lagrange multiplier technique. If the Lagrange multiplier solution is infeasible (some $h_i^* < 0$), then the offending task is removed from the list and the process is repeated until a feasible solution is found.

4.5 Selecting an Objective

An agent selects an objective by evaluating all possible objectives and selecting the "best" one. This comparison of objectives is done by calculating a **utility** for each possible objective based on the current state of an agent's knowledge, the environment the agent is working within, and short-term memory of recent events.

The purpose of this section is to explain the logic of the utility calculations and discuss the different factors that affect the utility calculations. The purpose of this section is *not* to give a detailed list of all of the equations and constants used. The interested reader is referred to appendix B for a detailed formula list.

The *DiFS* computer simulation software has encoded within it the equations that are used to calculate utilities. The structure of these equations is fixed by the software and can only be changed by recompiling the entire program.[1] However, the actual *values* of the utility constants used in program calculations are under the control of the experimenter. Thus a certain amount of control over the behavior of simulation agents can be exercised by the experimenter (see chapter 6 and 8). The remainder of this section will discuss the form of the equations used to calculate utilities and will give sample values; remember that these values may be changed at any time by the experimenter.

[1] A useful extension to the program that may be implemented in the future is to allow the user of the program to write a separate behavior module as a plug-in architecture to the program.

4.5.1 Matching Utilities with Human Behavior

An agent must calculate a perceived utility for each possible objective. These perceived utilities are compared to each other and the objective with the highest utility is selected. The difficult part of the utility approach to human behavior is deciding how to rate the utility of individual actions. The ratings system must be sophisticated enough so that all of the requirements of section 4.1 are met.

Creating formulas that calculate perceived utility and result in good agent behavior was one of the more challenging aspects of the *DiFS* simulation. The chosen implementation method was to specify in the software the *form* of the utility functions and allow the person using the software to control many of the *constants* used in the formulas. This proved to be an excellent solution to the problem; the form of the equations has been changed only occasionally to correct bugs and add new types of knowledge to the simulation. The experimenter controlled constants have been optimized (based on sample task performance) and can be varied to reflect different agent personalities (as described in section 6.3.2).

The method used to compare perceived utilities is to measure the utility of each objective on an absolute utility scale. The formulas that are used to calculate perceived utilities are based on the objective's **base utility**, a set of utility **bonuses**, and interpolations between utility bonuses. The base utility of an objective is defined to be the "normal" utility of doing that objective. The base utility of an objective is modified by utility bonuses that are added or subtracted depending upon prior circumstances, local environment, and project knowledge. In some cases these bonuses are simple additions (for example, a bonus for working on a high priority task), or they may be interpolations (for example, the bonus for reading mail is a function of how many messages are waiting to be read).

4.5.2 Calculating Utilities

The possible utility of an given objective has been arbitrarily defined to lie on a linear scale that ranges from approximately 0 to 10. An utility of 0.0 or less is considered to be highly undesirable and the agent will never do that. An utility of 10.0 or greater is considered to be so highly desirable that the agent would automatically do that. Actual utilities will range between these two extremes.

The equations used to calculate the utilities of the various possible objectives are a combination of `IF-THEN` rules and weighted averages. For example, the basic utility of reading a stored voice-mail message is 0.0 if there aren't any messages, 5.0 if there is a single message waiting, and up to 7.0 if there are at least 10 messages waiting (linearly interpolated from 5.0 to 7.0).

Figure 4.8 shows the approximate ranges of utility values that can be calculated for the different types of objectives. Note that the utility calculations are divided into two categories: regular objectives and communication objectives. Regular objectives (e.g., *Go Home*, *Read Mail*) require no interaction with other agents. The utilities of regular objectives occupy narrow, well defined ranges, bounded by *Wasting Time* on the low end (2.5) and *Go Home* on the high end (10.0). The communication objectives (e.g., *Send Request*, *Group Work*) have a normal range that is typically higher than regular objectives, but a gray region that trails off to zero. Calculation of the utility of a communication objective starts by calculating the utility assuming no recent memory, resulting in values in the black bands. Short term memory effects are then applied to the utilities; these memory effects reduce the utility of performing a communication action based on recent events. For example, *Saying a Request* in a conversation normally has a utility between 7.5 and 9.0. This is reduced by factors that measure the duration of the conversation and the recency of the last request for information. These factors can reduce the utility of *Saying a Request* all the way to 0.0.

Figure 4.9 shows the general flow of the utility calculations through the mental model. They can be divided into three categories: influences, task utilities, and objective utilities. Influences are little incentives provided by other agents or tasks to get a particular job done a little faster. Task utilities are basic measures of the usefulness of working on a particular

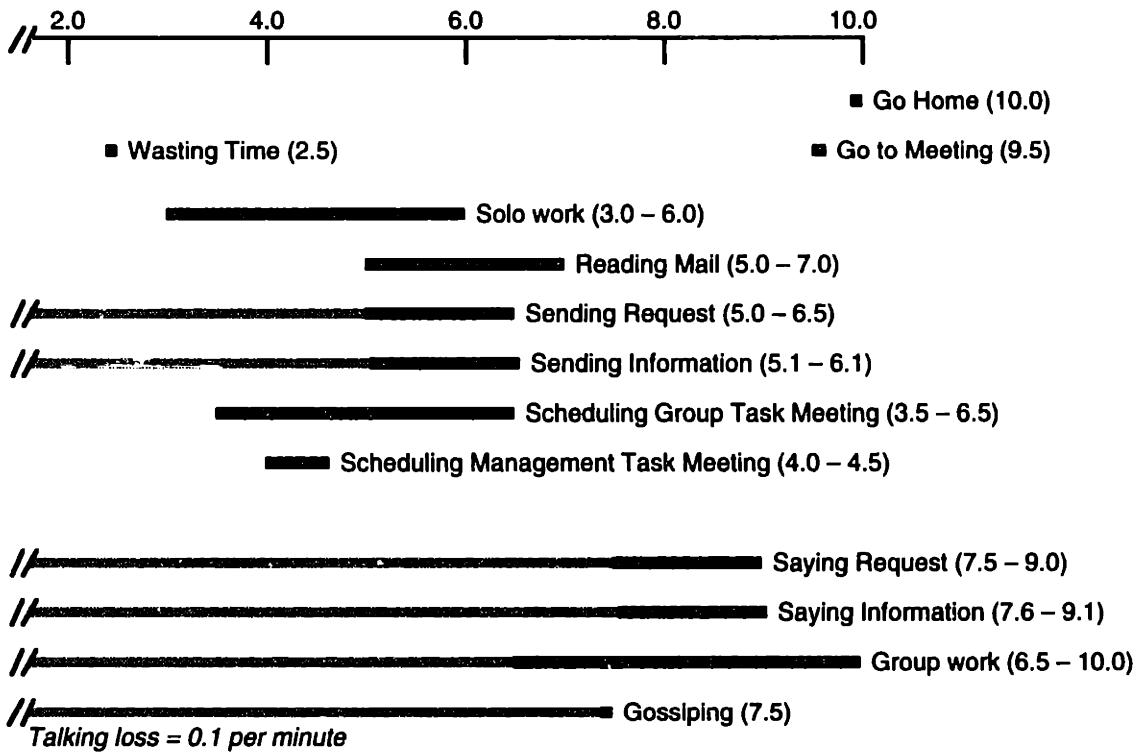


Figure 4.8: Typical ranges for utility calculations. These values do not take into account objective continuation bonuses.

task, requesting information about a particular task, or answering questions about a particular task to a particular person. Objective utilities translate task utilities into the convenience and importance of doing a particular objective.

Influences

Influences are effects from agents or other tasks that encourage an agent to work a little harder or answer a question. The two types of influences used in the *DiFS* simulation are (1) the influence of a task pushing for a driving task to get done quicker, or **dependency push** and (2) the influence of requests for information from other agents, or **nagging**.

Influences are normalized to return a range from 0 to 1 (they are later scaled by the particular utility being calculated). The strength of nagging is affected by the number of outstanding requests for information on a particular task as well as the amount of time that has passed since the request was made. The strength of a dependency push is proportional to the priority of the downstream task (i.e., an inconsequential task takes on new importance if it drives an important task) and the quantity of work that can be accomplished in the downstream task before more information is required.

General Utilities

General utilities represent the usefulness to an agent of (1) working on a particular task, (2) asking for information about a task, or (3) answering a request for information about a task. The utility of working on a particular task is referred to as the **task utility**. It is assumed to be a user-defined constant plus a bonus proportional to the priority of the task plus the effects of any influences that are acting on the task. The utility of answering a request for information (from a *sink*) is called the **sink utility**. It is assumed to be a user-defined constant plus a bonus for any nagging influences.

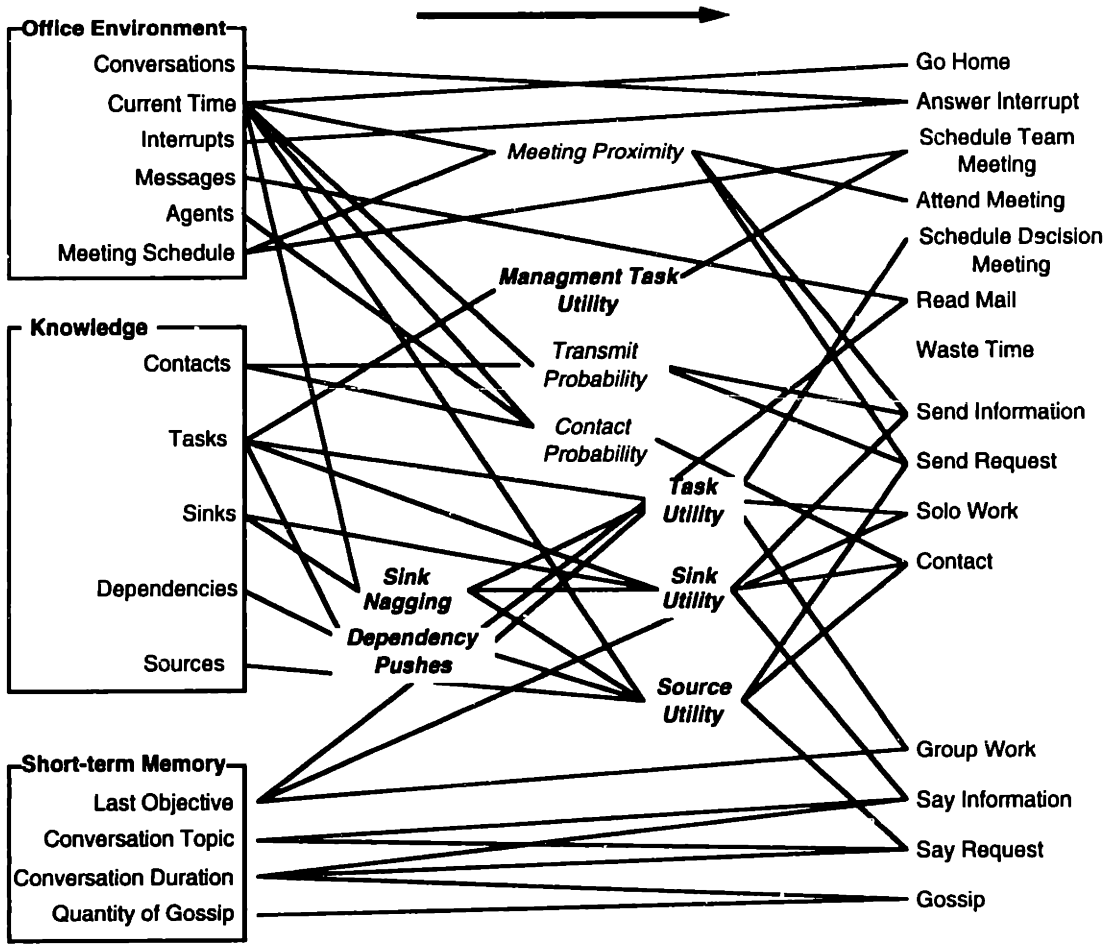


Figure 4.9: Structure of utility calculations.

The utility of making a request for information, or **source utility**, is slightly more complicated. If the agent is certain that the task has started and done some work, and either the agent doesn't have any outstanding requests or the last outstanding request was some time ago, then the source utility is a user-defined constant plus influence factors. However, if the agent does have an outstanding request that hasn't been answered and was sent in the recent past, the source utility is scaled by the length of time it has been since the last request. For example, assume the source utility of asking for information is (hypothetically) 6.4 and the time scaling factor is about 2 days. If it has been at least two days since the agent's last request, then the source utility of making a new request is 6.4. However, if it has only been one day since the agent has made a request, then the source utility of making a new request is scaled by 50% to 3.2. The choice of the time scaling constant encourages agents to not make frequent requests for information. Note that an agent who has nothing else to do will make requests more frequently than an agent who has other things to do of high utility. The idle agent will only wait until the source utility has climbed to the point where it is higher than the utility of doing nothing at all, whereas the busy agent waits until the source utility is higher than that of the other things the agent has to do.

Solo Objectives

The final stage of calculating utilities is to combine the task utilities with measures of the convenience of each objective and the effects of short term memory. Some objectives an agent pursues alone and some are pursued in a conversation. Objectives that are done by the agent

acting alone are called **solo objectives**. The form of the equations for each of the different types of solo objectives are discussed below.

Going Home is the objective of leaving work for the day. Its utility is based on the current time of day. The utility of this objective rises dramatically once the normal departure time of the agent is reached.

Attending a Scheduled Meeting has its utility based on the current time and the meeting's scheduled time. For a little while before the scheduled start of the meeting and during the bulk of the scheduled meeting, this objective receives a high utility.

Reading Mail is the act of retrieving messages from equipment and reading them. The utility of this objective is based on how many messages are pending and a little extra bonus if the last thing the agent did was read a message.

Wasting Time utility is a constant (normally 2.5).

Doing Solo Work is the act of logging hours on a solo work task. This utility is only non-zero if all dependency requirements have been satisfied. If they have been, then it's equal to the *task utility* of the task to be worked on, plus a bonus for having planned to work on this task today, and a bonus if this was the last objective the agent was performing. For example, assume Joe is responsible for a work task and has planned to work on it for 6 hours on a particular day. If the last thing Joe did was work on this task and he has logged 4 hours on it so far, then the utility of doing more work on the same task is equal to its task utility plus a bonus for planning and a bonus for most recent objective. Later on in the same day, if Joe has logged more than 6 hours and just got out of a meeting with another agent, then working on the same task has a utility only equal to the task utility.

Scheduling a Decision Meeting is the act of scheduling a meeting for a group decision task. This is only useful for the agent responsible for a group decision task and not currently in a conversation. The utility of scheduling a group decision meeting is based on the task utility plus a scheduling bonus *if* there isn't already a scheduled meeting. Otherwise, it's zero.

Scheduling a Team Meeting is the act of scheduling a team meeting for a particular management task. This is only useful for the agent responsible for the management task if it is known that tasks within the management task are currently active and if there are no other team meetings scheduled for this task. The utility of this objective is a constant plus a bonus for the priority of the management task.

Sending Information to another agent is the act of asynchronously providing information about a particular task to another agent that the current agent is *not* in conversation with. The utility of this act is equal to the sink utility, scaled by a transmission probability factor. If there is proximate meeting that the target agent will be attending, a meeting proximity cost is subtracted off before scaling by the transmission probability factor. The transmission probability factor is an internal estimate of the likelihood of the transmission going through. It ranges from 0% to 100%, and is estimated based on whether or not the last transmission attempt was successful and how long ago an unsuccessful transmission attempt was.

Sending a Request to another agent is the act of asynchronously requesting information about a particular task from another agent that the current agent is *not* in conversation with. It is exactly analogous to the *Sending Information* objective, with the exception that the source utility is used instead of the sink utility.

Answering an Interrupt utility is based on whether or not the agent is currently in a conversation, who is interrupting, and whether or not the agent has anything to say to them. If the agent is currently in a conversation, he/she will only answer an interrupt from another agent if the other agent has a higher rank than the agents in the current conversation (assuming the agent can tell who is interrupting).

Agents will only enter into a conversation if they know they will have something to say. If the utility of the current objective an agent is working on is high enough, then an agent will ignore conversation requests. As most humans are relatively polite and will at least acknowledge a conversation request, the factors that determine the utility of answering an interrupt have been selected so that agents will normally respond.

Contact is the act of attempting to start a synchronous communication (conversation) with another agent. The utility of contacting an agent is the product of the aggregate utility of things to discuss with that agent and the probability of successfully contacting him/her. The aggregate utility is the maximum of all of the sink and source utilities regarding that agent. A small bonus is added to the aggregate utility if it is larger than the maximum gossiping utility from conversations. This helps insure that conversations will not just be gossip exchanges. The contact probability is an internal estimate of the likelihood of being able to successfully reach the other agent. It is based on factors such as whether or not the other agent has gone home for the day, the location of the contacting agent, and how recently the two agents have been in the same location.

Conversational Objectives

When an agent is in a conversation there are four types of objectives that agent can select from that involve continuing the conversation: saying information, saying a request, doing group work, or gossiping. These are the **conversational objectives**. The first two are specifically directed towards another agent in the conversation; the remaining are directed towards the group at large.

All conversational objectives receive a bonus at the start of the conversation. Over the lifetime of the conversation, this bonus decreases and may go negative. For meetings, the bonus is held at the starting level until the scheduled termination time of the meeting whereupon it drops immediately to the level it would normally be at in a conversation. This decreasing advantage of the conversation reflects the idea that people feel uncomfortable spending a long time away from work. This doesn't mean that agents won't have long conversations; if agents wish to hold a long conversation it is broken up into a collection of short conversations.

Saying Information is the act of providing information about a particular task to another agent while in a conversation. The utility of this objective is equal to the sum of the sink utility, the conversational duration bonus, and a valid topic bonus. The particular sink is considered to be a valid topic if the current conversation is part of a scheduled meeting and the sink refers to a task that is the subject of the meeting or is otherwise strongly related to the subject of the meeting.

Saying Request is the act of asking another agent for information about a particular task while in a conversation (i.e., requesting information from a source). The utility of this objective calculated the same as the *Saying Information* objective, only using the source utility rather than the sink utility.

Doing Group Work is the act of sending a decision message regarding a group decision task. This objective is only possible if the agent is in a conversation with the other agents associated with the group decision task (and the agent is responsible for the group decision task). The actual utility is equal to the task utility, plus a bonus for group work, plus a continuation bonus if the last objective was group work, plus a bonus if the purpose of the conversation was to discuss this particular task. This combination of bonuses is usually

strong enough so that a scheduled meeting for the purpose of discussing a group decision task results in a large amount of group decision work being done.

Gossiping is the default objective for an agent in a conversation. The utility of gossiping is affected by the amount of gossip that has already gone on in the current conversation (the sum total of gossip by all agents involved in the conversation). The utility of gossiping is decreased as more gossip occurs in the conversation.

4.5.3 Example

To help clarify the objective selection process, this section presents a simple example of a single agent selecting an objective during a busy day at work. Consider the design project shown in figure 2.1 on page 29. Our data is drawn from an actual *DiFS* simulation run. To be specific, at 4:05:12 P.M. on the third day Sandy must decide what to do.[2] At this point in time, Sandy has already told Pat some information about the DESIGN task earlier in the morning. Chris has already left for the day. Sandy is currently in her office. The last thing she was doing was working on the DESIGN task (for the past 10 minutes).

Sandy first considers the standard options: going home, attending a meeting, answering a conversation request, reading mail, doing nothing. Sandy's default time for going home is 5:00 P.M.; it's before that time and hence the *Go Home* objective receives a utility of 0. There are currently no scheduled meetings for the remainder of the day; hence the utility of *Attend Meeting* is 0. There are no pending conversation requests, so the *Answer Interrupt* objective receives a utility of 0. Sandy has three pieces of equipment available to her (because she is currently in her office); they are her desk, telephone, and voice. Her desk and telephone are capable of receiving asynchronous messages, however neither of them have any stored messages. Hence the utility of *Reading Mail* is 0. In fact, the only standard option that has a non-zero utility is that of *Wasting Time* and that is set to the default value of 2.5.

Sandy now considers tasks that she is responsible for: the DESIGN task (a solo work task), the REVIEW task (a group decision task), and the EXPERIMENT BUILD management task. The REVIEW task cannot be started yet (and her current estimate of its earliest start time is day 6), so the options of *Scheduling a Meeting* or *Group Work* both receive utilities of 0. The EXPERIMENT BUILD management task has a parameter that specifies how frequently team meetings should be held. This parameter is currently set to "never" and hence the *Scheduling a Meeting* objective for the EXPERIMENT BUILD task also receives a utility of 0.

The DESIGN task is more promising. The current completion level of the task is 61%, but Sandy possesses sufficient driving dependency knowledge to finish that task to 73% (and hence can work for at least a little while before acquiring more knowledge). Sandy calculates the utility of working on the DESIGN task. Working on a work task has a base utility of 3.0. The DESIGN task has a normal priority; hence a bonus of 0.3 is added to the base utility. The INTERFACE SOFTWARE and ANALYSIS PROGRAM both depend upon the DESIGN task; a calculation of how far they can proceed based on current progress results in an additional bonus of 0.85 added as a "dependency push". Sandy had planned to do 8 hours of work on the DESIGN task and has only accomplished a little over 6 so far today; hence an additional 0.5 is added as a planning bonus. The overall utility of *Solo Work* on the DESIGN task is 4.65.

Sandy also has the option of contacting or sending messages to the other agents in the simulation (Chris and Pat). Both Chris and Pat require information from the DESIGN task and hence are **sinks**. Chris and Pat are also **sources** of information about their respective tasks. The six valid communication objectives are then (1) contact Chris, (2) contact Pat, (3) send a message to Chris containing information about DESIGN, (4) send a message to Pat containing information about DESIGN, (5) send a message to Chris asking for information about ANALYSIS PROGRAM, and (6) send a message to Pat asking for information about INTERFACE SOFTWARE.

[2] To generate this description, I used the debugger to step through and examine every stage of the "do what?" mental evaluation. Unfortunately, the simulation software doesn't allow regular users to do this.

Consider Chris first. Sandy possesses an internal estimate of the best possible start time of the ANALYSIS PROGRAM task; this suggests that Chris will not have been able to start it yet and hence *requesting information* from Chris receives a utility of zero. Chris needs information about the DESIGN task; however, there are no unanswered requests for information about this task from Chris and Sandy has last talked to Chris about the DESIGN task at approximately 9:45 A.M. this morning. Hence the utility of *sending information* to Chris about the DESIGN task is zero. Because Sandy has no reason to send information to Chris or questions to ask of Chris, the utility of *contacting* Chris is zero.

Sandy must also consider Pat. Pat, like Chris, is a **source** of information for Sandy about a task (the INTERFACE SOFTWARE) task and a **sink** of information for Sandy about the DESIGN task. The sink calculation is just the same as the sink calculation with Chris; it works out that Sandy has (1) told Pat information about the DESIGN task within the last 6 hours and (2) does not have any unfulfilled requests for information from Pat. Hence the utility of *sending information* to Pat about the DESIGN task is zero.

However, Pat is a source of information about the INTERFACE SOFTWARE task, and Sandy knows that Pat has been able to work on it since the morning. Hence Sandy infers that Pat probably has generated some information by working on that task. Sandy calculates the utility of requesting information (both asynchronously and synchronously). The base utility for requesting information is 5.0. To this is added a bonus for the importance of the tasks that the INTERFACE SOFTWARE task drives (namely the REVIEW task); this bonus adds 1.0 to the utility. The nominal utility of 6.0 is scaled based on how long ago Sandy last requested information about the task; in this case, the morning conversation translates into a scaling factor of about 52%, or an actual request utility of about 3.1.

Given that the request utility is greater than zero, Sandy considers either transmitting an asynchronous request or contacting Pat directly. In the past, Sandy has always succeeded in transmitting successfully to Pat, so the transmission probability is estimated as 100% resulting in a *send request* utility of 3.1. Pat is not currently in the same location as Sandy, and the sum total of things that Sandy has to talk with Pat about is relatively small, so the *contact* utility is calculated to be about 2.5 (this would be higher if Pat was in the same office).

Overall Sandy has four "useful" options: *Waste Time* (2.5), *Solo Work* on the DESIGN task (4.65), *Send Request* to Pat asking for information about the INTERFACE SOFTWARE task (3.1), or *Contact* Pat via some synchronous communication (2.5). Sandy decides to work on the DESIGN TASK. As it turned out, a few minutes later Pat called Sandy on the telephone to ask for more information about the DESIGN task and Sandy got the chance to request information about the INTERFACE SOFTWARE task.

4.5.4 Stochastic Variation

Up to this point, no aspect of the *DiFS* model or simulation has exhibited any type of stochastic variation or randomness. This is not accidental; there hasn't been any randomness. Not having stochastic variability in the model or simulation was a deliberate decision; variations should only be introduced if they add value.

However, there are a number of advantages to having a stochastic simulation. The most important advantage is that in a non-random simulation, the experimenter must always be concerned about small variations in the input file creating large variations in the simulation results. Stochastic variations can be used in a nonlinear simulation like *DiFS* to establish that the model is not sensitive to project perturbations. The second advantage of a stochastic simulation is that by incorporating small random effects, the experimenter may estimate not only the predicted outcome, but also the variability associated with that predicted outcome.

The *DiFS* simulation introduces stochastic variation at just two points; both of these points are under complete control of the experimenter. The first is in the arrival time of agents in the office in the morning. The second is in the agent's calculations of the utilities of the various objectives they may pursue.

Without variation in agent's arrival time, all computer agents will show up every morning at exactly the same time. Not only is this unrealistic, but more importantly it often results in a brief tie up of the telephone system in the office as many agents will attempt to call each other simultaneously. Hence a small variation in arrival time may be specified by the experimenter.

The **arrival time variation** is specified as a time in minutes, ΔT. The normal arrival time, T, is 8:00 A.M. Each morning every agent selects an arrival time randomly from the interval $T + \Delta T$. The arrival times are considered to be uniformly distributed random numbers. This simple variation virtually guarantees that no two agents arrive at exactly the same time. If the experimenter wishes to remove this variation, then set $\Delta T = 0$.

The second type of stochastic variation is introduced at the point where an agent calculates the utility of an objective. After the utility has been calculated, but before it has been compared to other utilities, a random "bonus" is added to the utility value. This bonus is a uniformly distributed random value between 0 and U_{SV}, where U_{SV} is a value set by the experimenter.

The effect of the utility variation is to slightly "fuzz" the utility calculations. This has three advantages: first, it mimics the difficulty humans have in selecting between two activities with roughly equivalent utilities. Second, the requirements and parameters set up by the experimenter are not affected; the only difference between multiple runs will be a slight variation in *when* individuals choose to do things (of course, over time small initial variations may result in drastic differences). Third, the utility randomization may be easily removed from the system by simply setting $U_{SV} = 0$.

4.6 Summary

This chapter has discussed the model of agent personal behavior used in the *DiFS* simulation. A conversation etiquette and set of messages have been specified for how agents communicate with each other in synchronous communications. A method of storing task and contact knowledge by an interconnected mental model of the design project has been presented. The stored mental model of an agent allows the agent to deduce some factors in the simulation such as when information is likely to become available. The agent behavior itself has been divided into two parts: a high-level part that selects the current objective, and a low-level part that uses scripts to carry out the selected objective.

This specification of the agent personal behavior model concludes the description of the *DiFS* simulation. For a discussion of the actual operation of the simulation software, refer to appendix A. For a discussion of the implementation of the simulation software, refer to appendix B.

Chapter 5

Typical Results

The purpose of this chapter is to introduce the reader to what it looks like to run the *DiFS* simulation and how one can extract results from it. Throughout this chapter (and later chapters in the thesis), I will use screen captures from the software as illustrations. However, this chapter is not intended to be an introduction to how to use the *DiFS* simulation, nor a reference manual for the features of the software. For that information, the reader should consult the *DiFS* User's Manual in appendix A.

This chapter is divided into three parts. The first section describes what happens in a typical simulation run using the human-factors project from chapter 2. The second section shows examples of how the human-factors project may be modified to examine the effects of project perturbation. The final section demonstrates a complex project model.

5.1 A Simple Example

The example project used throughout this section was first presented in figure 2.1 on page 29. It is a project model of three engineers working together to create a human-factors experiment. One engineer is in charge of designing the experiment and running the project review. The other two engineers are responsible for writing software. A screen capture from the *DiFS* program displaying the project topology is shown in figure 5.1. The project office environment is show in figure 5.2.

There are three computer agents: Sandy, Pat, and Chris. Sandy is responsible for the DESIGN and REVIEW tasks. Pat is responsible for the INTERFACE SOFTWARE task and a participant on the REVIEW task. Chris, a programmer who only works half days, is responsible for the ANALYSIS PROGRAM task and participates on the REVIEW task. As shown in the office layout, each agent has a private office to work in, and there is a single common conference room for group discussions.

Figure 5.3 shows the parameters specified for the INTERFACE SOFTWARE task. This task is configured to have a work time of 100 hours, a communication time of 40 hours, and a linear information convergence. Project priority is normal. As this task represents the writing of computer software, the documentation thoroughness has been set to a low level (computer programmers are notoriously bad about documenting what they have done).

The screen capture of the project model, figure 5.1, does not give a good sense of the nature of the dependencies between the tasks. Figure 5.4 shows the dependency between the DESIGN task and the ANALYSIS PROGRAM task. The dependency specifies a narrow scope and detailed thoroughness; in this case, the work to be done on the ANALYSIS PROGRAM consists of taking the data generated by the subjects and calculating a variety of metrics to be fed into a standard statistical package. This software will depend strongly upon the type of data recorded, but little upon the method of presentation to the subject. Hence the scope of this dependency is narrow and the thoroughness is detailed. The effect of the narrow scope is to restrict the amount of

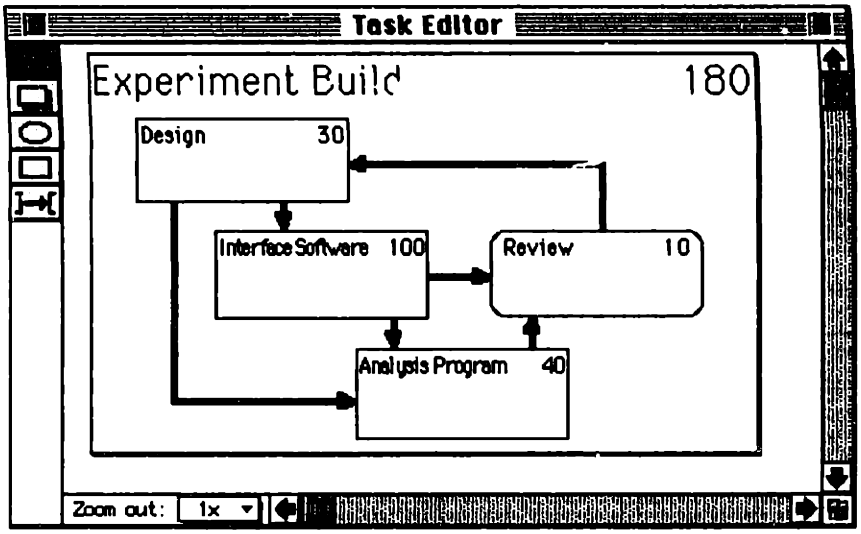


Figure 5.1: Software version of the human-factors design project.

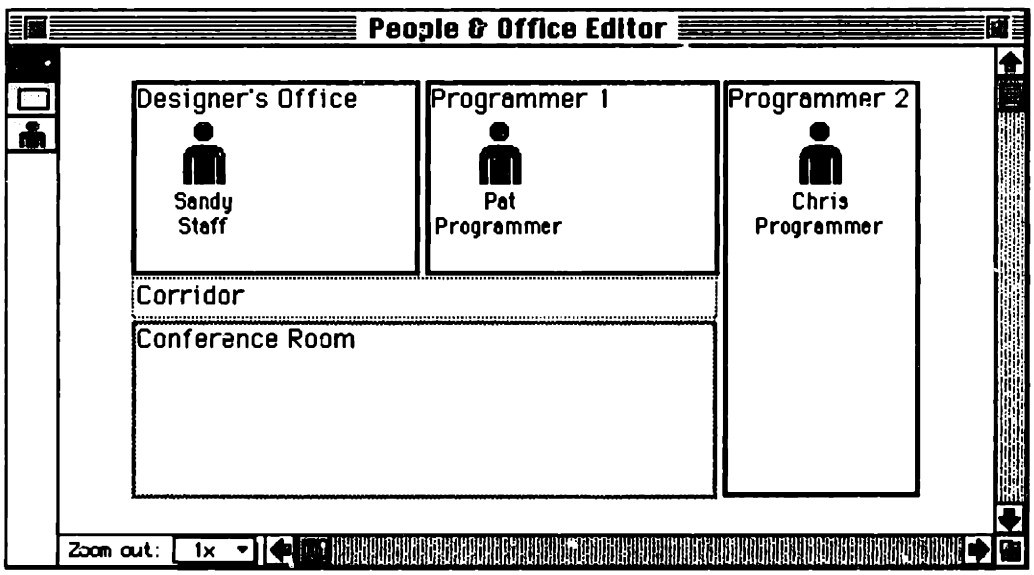


Figure 5.2: Office environment for the human-factors design project.

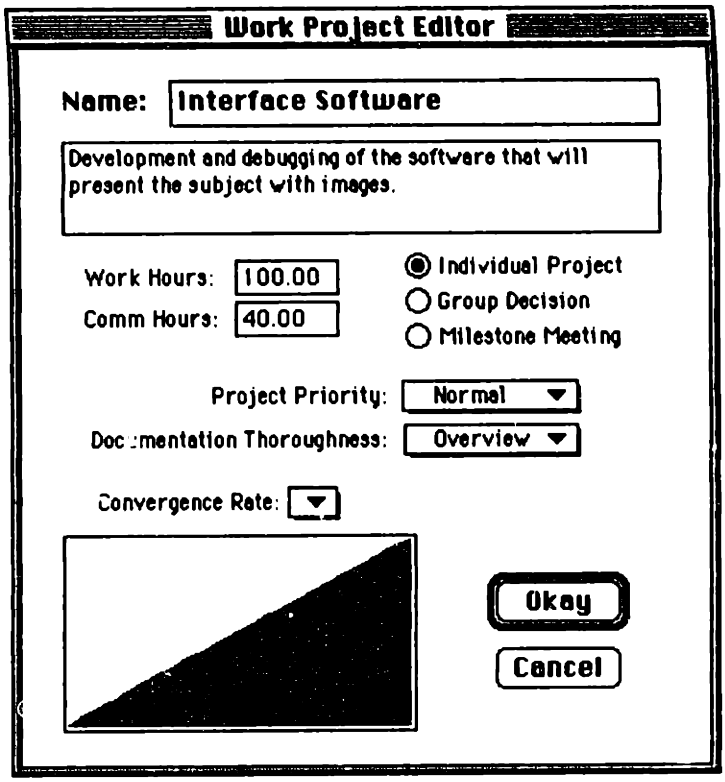


Figure 5.3: Task parameters for the INTERFACE SOFTWARE task.

information needed from the DESIGN task from a potential maximum of 10.0 hours to 3.0 hours.

Screen captures of the parameters for all four tasks and six dependencies will not be repeated here. Please refer to figure 2.9 on page 40 for a complete list of the task and dependency parameters.

The human-factors design project uses a standard set of simple office equipment: telephones, desks, and agent voices. Two types of synchronous communication are allowed: telephone calls and conversations. Conversations require the agents to be in the same room; telephone calls require them to be in their respective offices. Telephone calls have been configured to take 20% longer than conversations to send a message.

Two types of asynchronous communication are permitted: voice mail and formal mail of documents. Voice mail allows up to a two minute message to be left on the receiving agent's telephone. Regular mail allows an agent to send formal documentation from desk to desk; it is assumed to take 15 minutes for the documentation to be delivered. A more realistic model of the small office might include allowing agents to drop documents on each other's desks and send each other electronic mail.

5.2 Running the Simple Example

This section describes in detail what happens during a single sample run of the simulation.

Preliminary estimates of start times are calculated before the simulation run commences. Figure 5.5 shows the estimated earliest start times and calculated maximum possible completion levels (without acquiring information) for each of the tasks in the simulation. The figure shows that only one of the tasks, DESIGN, is capable of starting work. The black outline within the task indicates that approximately 70% of the DESIGN task can be accomplished before it will need information from the REVIEW task. The two software tasks (INTERFACE SOFTWARE and ANALYSIS PROGRAM) need to wait until at least the second day of the simulation before they

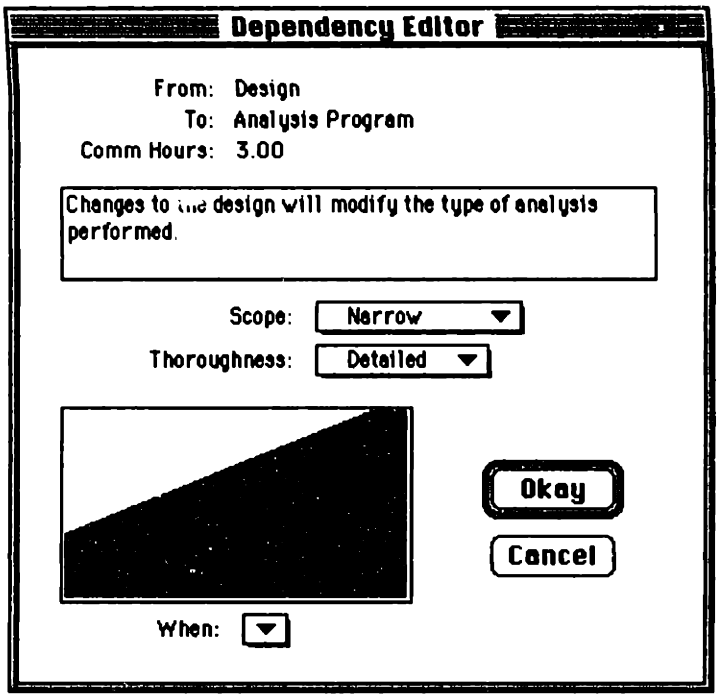


Figure 5.4: Dependency between the DESIGN task and the ANALYSIS PROGRAM task. The completion function is given by the gray polygon.

will have sufficient information to commence.[1] The calculated earliest start time for the REVIEW task is the sixth day.

5.2.1 First Day

8:00:00 A.M.: On the morning of the first day, all three agents travel to their respective offices. Sandy is intending to *Work* on the DESIGN task (utility 4.8). Pat and Chris are both *Wasting Time* because they have nothing better to do and they realize that there is no point in contacting Sandy for information because she won't have generated any yet.

8:05:30 A.M.: All agents have arrived in their offices. They continue to carry out their planned objectives; Sandy working and Pat/Chris killing time. Working and idle agents specify activities that last approximately 15 minutes. At the end of this time, the agent re-evaluates what he/she should be doing. For the sake of clarity, these re-evaluations will only be described if the agent selects a new objective. Short objective periods help an agent to evenly divide his/her time between different objectives.

12:35:30 P.M.: Chris decides to *Go Home* (utility 10.1). Chris is only scheduled for half-time employment on this project, and assuming a 9 hour day (with an arrival time of 8:05:30 A.M.), this is the correct time to go home. Because Chris was waiting for Sandy to generate information about the DESIGN task, Chris did not do anything this day.

12:50:30 P.M.: Pat decides to *Send Request* (utility 2.5) to Sandy. Note that this utility is just equal to the utility of *Wasting Time*; Pat has been waiting until the request utility is high enough to warrant asking for information. The selected channel is "Voice Mail", and the message simply asks for information about the DESIGN task. Sandy is still working on the DESIGN task.

[1] In the *DiFS* simulation, the first day is day 0. This is due to the C/C++ tradition of numbering arrays with the first element starting at index 0.

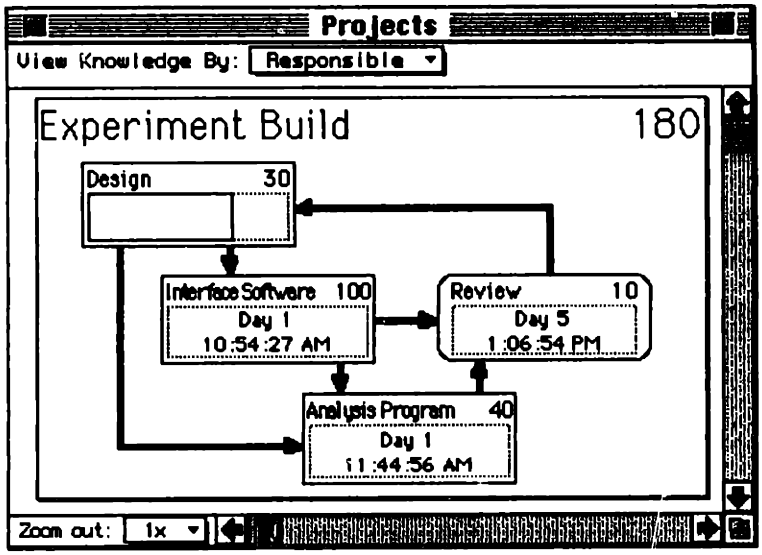


Figure 5.5: Start time estimates and maximum possible completion levels calculated before the simulation has commenced.

12:52:02 P.M.: Pat has sent out the message (it took just over 1.5 minutes to send). Pat returns to being idle and wasting time. Sandy is in the middle of working and hence does not immediately retrieve the message. Pat stores up the knowledge that her last attempted message to Sandy succeeded, and also notes that she now has one outstanding request for information sent to Sandy.

12:55:30 P.M.: Sandy finishes a small portion of the work in the DESIGN task, considers her options, and chooses to read the stored voice-mail message (utility 5.0).

12:56:38 P.M.: Sandy decides that Pat's message warrants an immediate response. Sandy selects the *Contact* response (utility 5.8) and picks the "Conversation" channel as the appropriate synchronous channel, having predicted that the two agents have over two hours worth of potential discussion and hence a telephone call would be inappropriate because it is considerably slower than a face-to-face discussion. Conversation requires the two agents to share the same office, so Sandy starts the process of traveling to the "Programmer 1" office that Pat is assigned to.

12:58:21 P.M.: Sandy arrives in Pat's office and immediately starts a conversation with Pat (refer to figure 4.4 on page 73 for the script that Sandy is following). The contact is being made over the "Conversation" channel, and hence Sandy's "Voice" is being used.

12:58:41 P.M.: Pat, who was wasting time, responds to Sandy's request for conversation.

12:58:42 P.M.: The conversation is declared open. Both Sandy and Pat specify what they would like to say and how loudly they would like to say it. Sandy selects *Say Information* about the DESIGN task (utility 7.7) with volume 176 and Pat selects *Gossip* (utility 7.5) with volume 130. Note that Pat selected gossiping not because Pat is not interested in hearing about the DESIGN task, but rather because Pat felt it would be inappropriate to repeat her request for information. Gossiping serves as a useful conversation filler; when an agent has nothing else to say but wishes to remain in the conversation the agent sends a gossip message at a low volume.

12:58:43 P.M.: Sandy is declared the winner of the talking contest and proceeds to transfer her information message to Pat. The total message length is only 4 minutes. By default conversational messages are limited to a maximum length of 5 minutes. Although Sandy

actually has over two hours worth of material to discuss, it would be inappropriate to attempt to pass it on in a single breath. Breaking the conversation into smaller pieces gives other agents an opportunity to speak.

1:03:33 P.M.: The conversation reaches an interlude. Pat once again specifies *gossip* (utility 7.4), volume 130. Sandy continues to *Say Information* (utility 7.6), volume 165. Once again, Sandy "wins" the talking contest and transfers her message.

1:10:20 P.M.: Conversation interlude. Sandy still wishes to *Say Information* (utility 7.4), volume 164. Pat is still on *Gossip* (utility 7.2), volume 130. Note that the utilities expressed by each agent are slowly dropping as the conversation duration increases.

...the conversation continues for a few more exchanges...

2:54:51 P.M.: Sandy decides to terminate the conversation and return to work. She has passed on to Pat all the information available about the DESIGN task up to the scope and thoroughness Pat desired. Sandy's work utility is 4.8 and the utility of gossiping in the conversation is below that. From Sandy's perspective the conversation has lasted long enough and it's time to go back to work. Pat, who has no work to do, is still attempting to gossip. Sandy at this point is walking towards the door of the office.

2:54:52 P.M.: Pat realizes that Sandy is leaving and returns to being idle (utility 2.5). Pat does not yet possess sufficient information to start the INTERFACE SOFTWARE task.

2:56:34 P.M.: Sandy is back in her office and returns to work (utility 4.8).

5:06:34 P.M.: Sandy decides to leave for the day. Sandy succeeded in working 7 hours on the DESIGN task (out of a planned 9 hour effort).

5:09:52 P.M.: Pat decides to *Go Home* for the day.

5:12:52 P.M.: The office building is empty.

The first day of the simulation proved to be relatively straightforward as only two agents did anything of interest. Sandy worked on the DESIGN task. Pat waited until she felt that Sandy had had a chance to generate some information about the DESIGN task, and then sent a voice-mail message to Sandy requesting information. Sandy chose to respond to the voice-mail message in person, and ended up in a conversation that lasted two hours. Once Sandy and Pat had exhausted the things they had to say, Sandy returned to working on the DESIGN task.

At the end of the day it is instructive to compare the task knowledge of the different agents. Chris did not work or participate in any information exchanges, so Chris's knowledge remains unchanged. Figures 5.6 and 5.7 show the task knowledge of Sandy and Pat respectively. Sandy has completed 22% of the DESIGN task and still has plenty of work that can be done on this task without acquiring more information. Pat has some knowledge of the DESIGN task, although it's a little out of date (Sandy worked on the task after Pat last talked with her). Pat is still waiting for more information before she can start working on INTERFACE SOFTWARE; currently her best estimate of when she should be able to start is 10:54:27 the next morning. Sandy, who has a better idea of the current state of the DESIGN task, is estimating a best possible start time closer to noon.

5.2.2 Second Day

8:00 A.M.: Sandy heads into the office with the intent of *Contacting* Pat and passing on more information. The chosen method of contact is "conversation", so Sandy heads directly to Pat's office. Pat heads into the office intending to *Waste Time*. Chris plans on *Sending a Request* to Sandy for information on DESIGN via voice-mail.

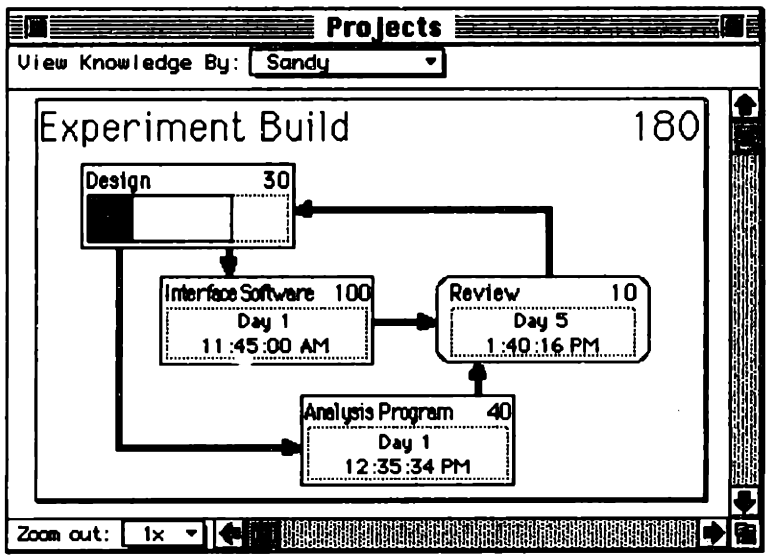


Figure 5.6: Sandy's task knowledge at the end of the first day.

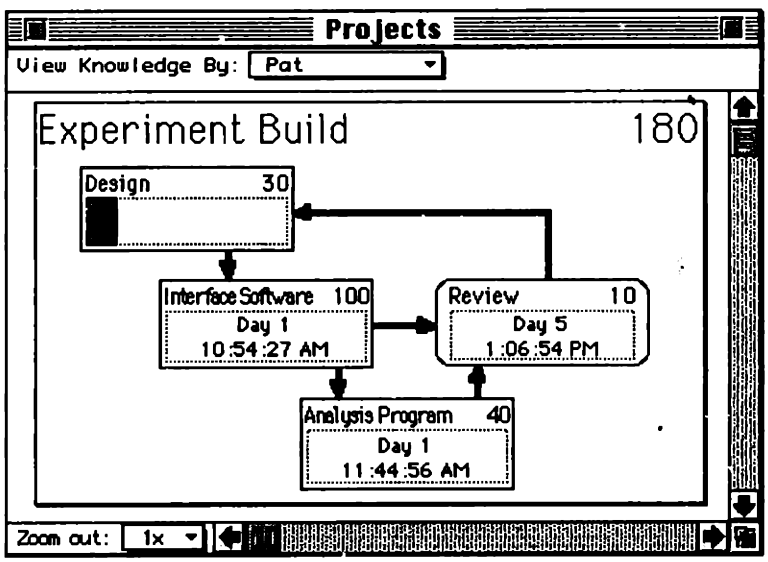


Figure 5.7: Pat's task knowledge at the end of the first day.

8:05 A.M.: Sandy engages Pat in conversation. They begin transferring information about the DESIGN task. Chris sends the request for information to Sandy, and then starts wasting time. Chris has no way of knowing the Sandy is out of her office and will not receive the message for some time.

9:02 A.M.: Sandy and Pat have finished their conversation. Pat still does not have enough information to start work, so Pat returns to wasting time. Sandy heads back to her office to go to work on the DESIGN task.

9:03 A.M.: Sandy arrives in her office and goes to work on the DESIGN task despite the waiting message from Chris.

9:13 A.M.: Having accomplished some work, Sandy notices the waiting message and retrieves it.

9:14 A.M.: Sandy determines that the best way to answer Chris's request is to travel to her office and answer it in person.

9:18 A.M.: Sandy and Chris strike up a conversation. Chris only needs a fairly narrow scope of information about the DESIGN task, and hence this conversation goes rather quickly.

10:05 A.M.: Sandy returns to her office to go back to work on the DESIGN task.

11:05 A.M.: Chris sends a voice-mail message to Pat requesting information about the INTERFACE SOFTWARE task. This message is a bit premature because Pat has not started working on that task. If Chris had a truly global mental model of the tasks and their dependencies, Chris could have determined from her conversation with Sandy that the INTERFACE SOFTWARE could not have acquired enough information in order to start. However, Chris's limited task model does not include the dependency between DESIGN and INTERFACE SOFTWARE. I might add that Chris doesn't need this information to start work on the ANALYSIS PROGRAM task; Chris just doesn't have anything better to do.

11:06 A.M.: Pat receives Chris's message, and decides to answer it in person.

11:10 A.M.: Pat starts a conversation with Chris and passes on an updated estimated of the start time of the INTERFACE SOFTWARE task.

11:50 A.M.: After gossiping with Chris for 40 minutes, Pat returns to her office.

12:35 A.M.: Chris goes home.

1:52 P.M.: Pat sends voice-mail to Sandy requesting more information about DESIGN. From Pat's perspective, Sandy should have had ample time to generate the information that Pat needs.

1:53 P.M.: Sandy decides to go talk with Pat. At this point, Sandy has *not* received the voice-mail message from Pat; it's available, but the utility of listening to voice-mail is lower than the utility of contacting Pat. Pat and Sandy briefly exchange information before Sandy returns to her office. During this conversation Pat refrains from asking about DESIGN because Pat assumes that Sandy has received the message. Sandy does not volunteer information about DESIGN because Sandy has not received Pat's message and hence doesn't realize that Pat is waiting for the information. The *DiFS* simulation currently does not have a mechanism for one agent to ask another whether or not a message was received.

2:05 P.M.: Sandy sends voice mail to Chris about ANALYSIS PROGRAM.

2:07 P.M.: Sandy finally receives the voice-mail message from Pat and decides to go talk with Pat personally. Sandy spends the next hour and a half telling Pat about the DESIGN task.

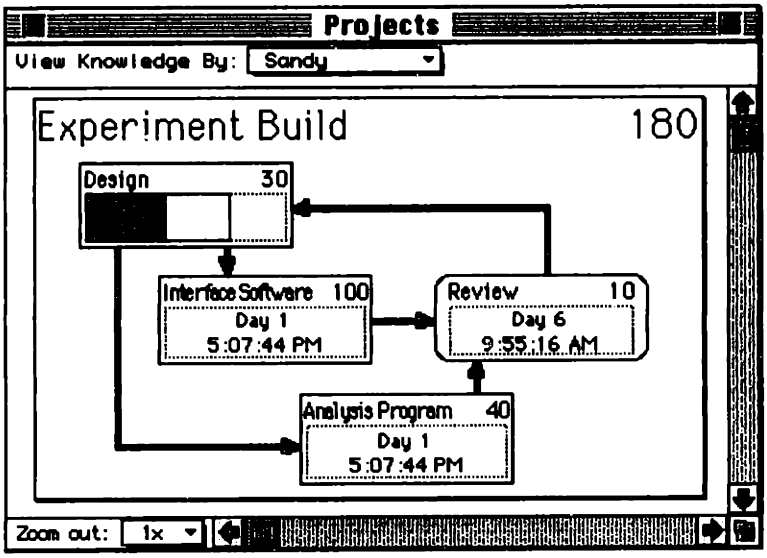


Figure 5.8: Sandy's task knowledge at the end of the second day.

3:46 P.M.: Sandy returns to her office to go back to work on DESIGN. Pat begins working on the INTERFACE SOFTWARE.

5:06 P.M.: Pat and Sandy head home.

At the end of the day, Pat has finally acquired enough information to start working on the INTERFACE SOFTWARE task. Sandy has made some progress on the DESIGN task, although not as much as she originally planned upon (primarily because of the long conversations). Chris received some useful knowledge, but left too early in the day to receive the rest of the information that Sandy has generated. Figures 5.8, 5.9, and 5.10 show the task knowledge of Sandy, Pat, and Chris respectively.

5.2.3 Remaining Days

The remaining days in the life of the simulation pass much the same as the first two days. Day 2 (the third day) starts off with a great deal of communication between Sandy, Pat, and Chris as each tries to be updated on the status of each other's tasks. In fact, all three of them end up in Chris's office where they share a three-way conversation and discuss the state of each task. Sandy ends up sending Chris the formal documentation generated as a part of the DESIGN task as well as passing on information in person; this combined approach minimizes the amount of time she has to spend communicating. Chris starts work on the ANALYSIS PROGRAM. Sandy is still making progress on DESIGN and Pat is working on INTERFACE SOFTWARE.

Day 3 (the fourth day) kicks off with more conversational exchanges of information between Sandy, Pat, and Chris (who all end up in the same office). The first group decision meeting to discuss the REVIEW task occurs on day 8. Figures 5.11, 5.12, and 5.13 show the task knowledge of Sandy, Pat, and Chris respectively after the first two weeks (10 days) have passed.

The entire design project is finally complete as of 11:41 A.M. on day 19. The chosen stopping criteria was when Sandy, the agent responsible for the overall project, knew that all tasks had completed. In fact, the simulation terminates with a phone call between Chris and Sandy as Chris relays the information that the ANALYSIS PROGRAM task is complete. Figures 5.14, 5.15, and 5.16 show the task knowledge of Sandy, Pat, and Chris respectively at the end of the entire design project. Note that Sandy's knowledge of the world indicates that she is aware that every task has completed, whereas neither Pat nor Chris are aware that the other has finished her task.

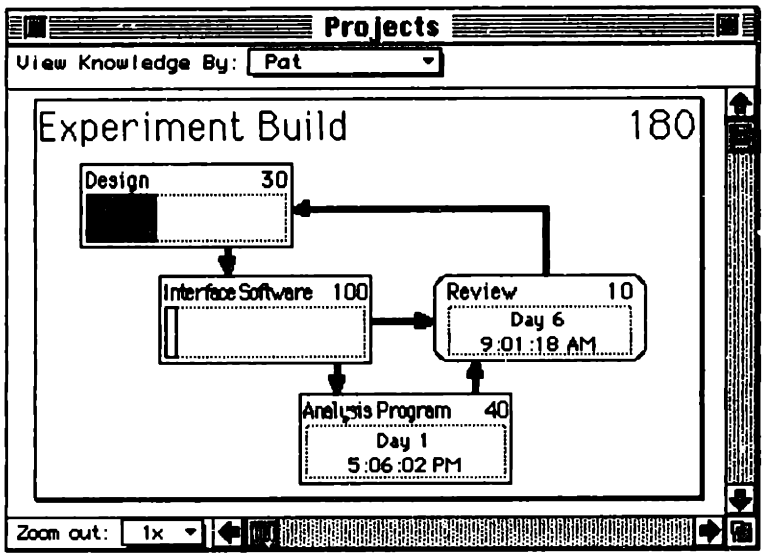


Figure 5.9: Pat's task knowledge at the end of the second day.

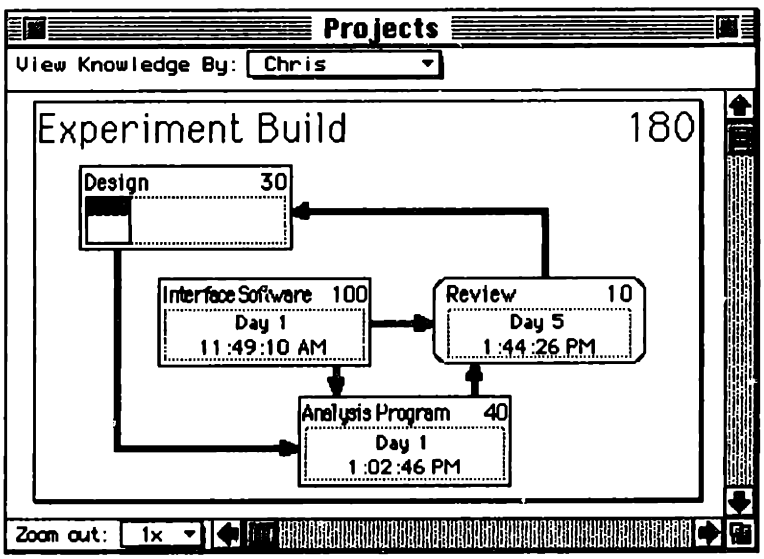


Figure 5.10: Chris's task knowledge at the end of the second day.

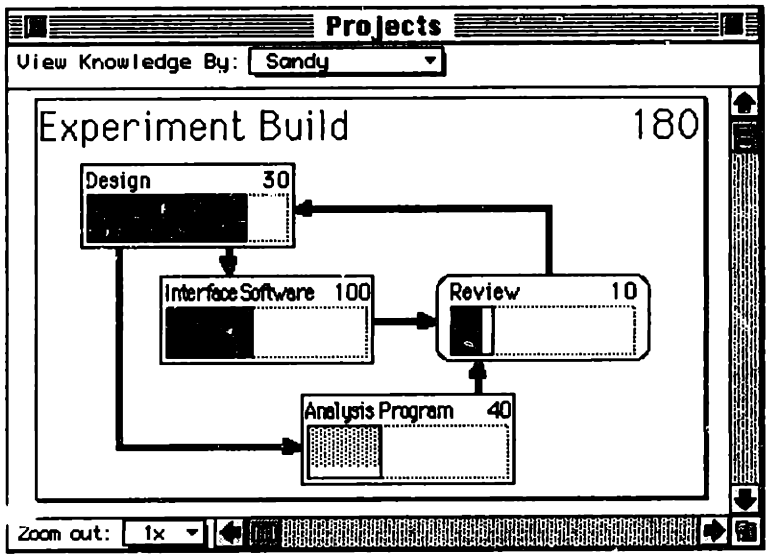


Figure 5.11: Sandy's task knowledge at the end of the tenth day.

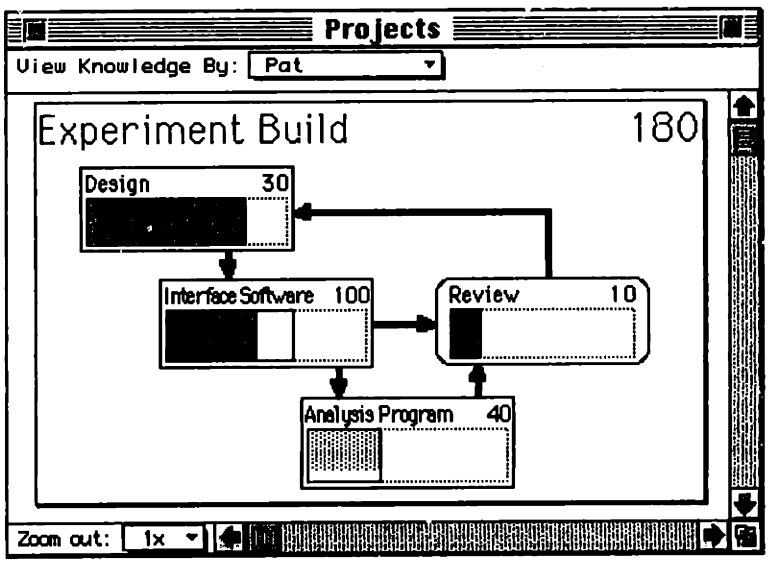


Figure 5.12: Pat's task knowledge at the end of the tenth day.

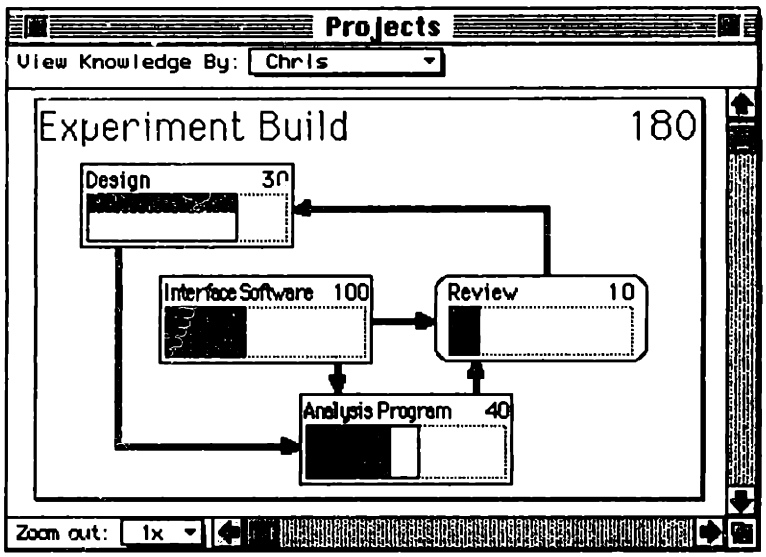


Figure 5.13: Chris's task knowledge at the end of the tenth day.

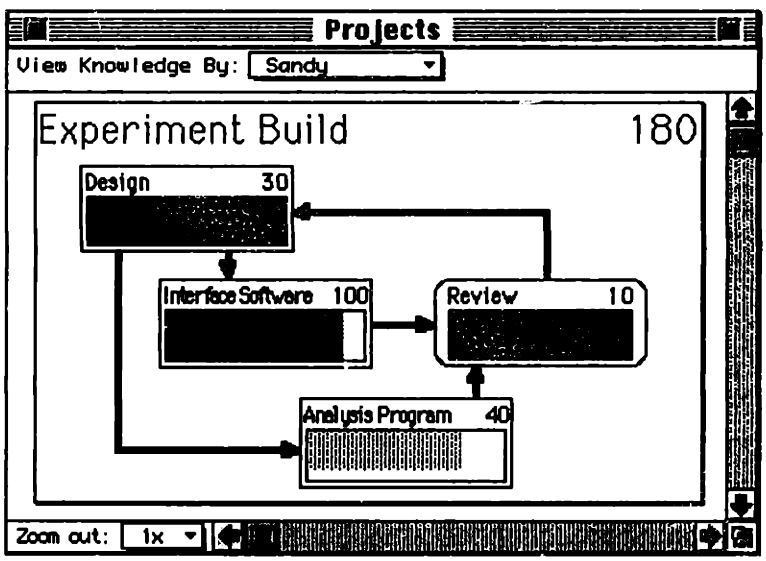


Figure 5.14: Sandy's task knowledge at the end of the project.

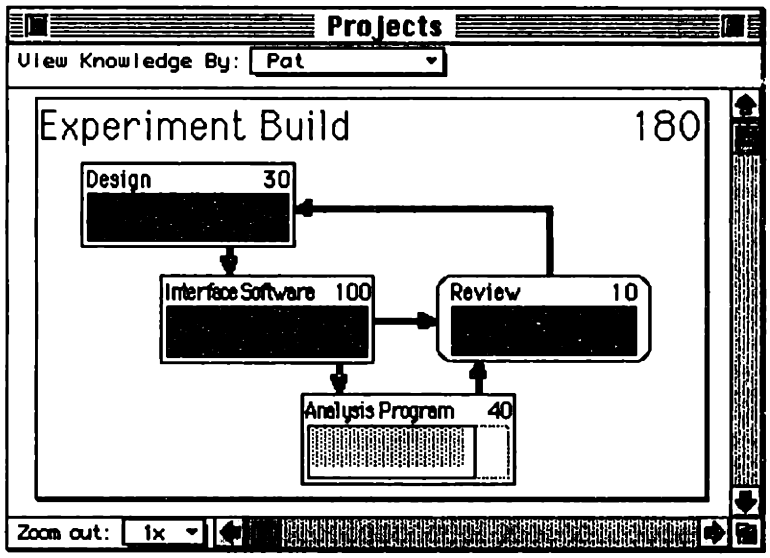


Figure 5.15: Pat's task knowledge at the end of the project.

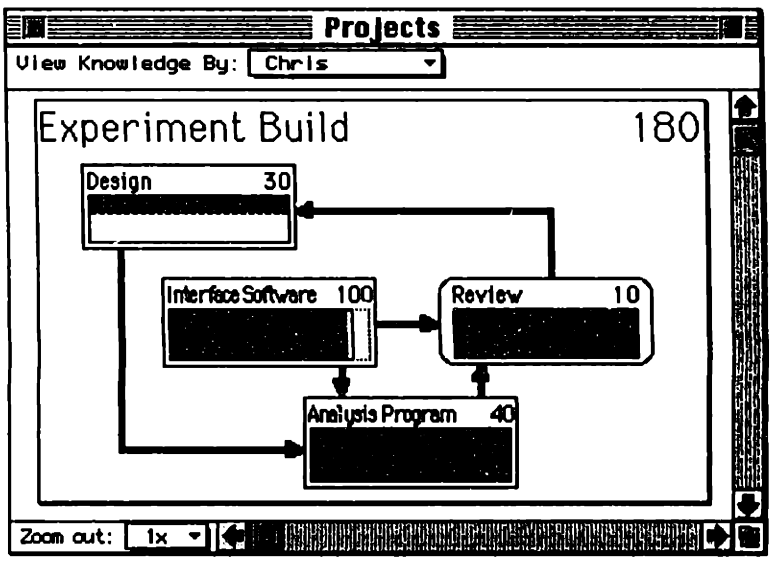


Figure 5.16: Chris's task knowledge at the end of the project.

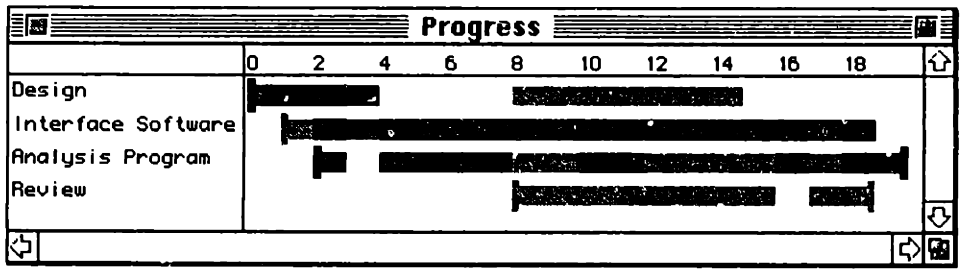


Figure 5.17: Time trace of task progress. Darkness of bar on each day is proportional to hours of work put into that task on a particular day. A black bar equals four or more hours of work.

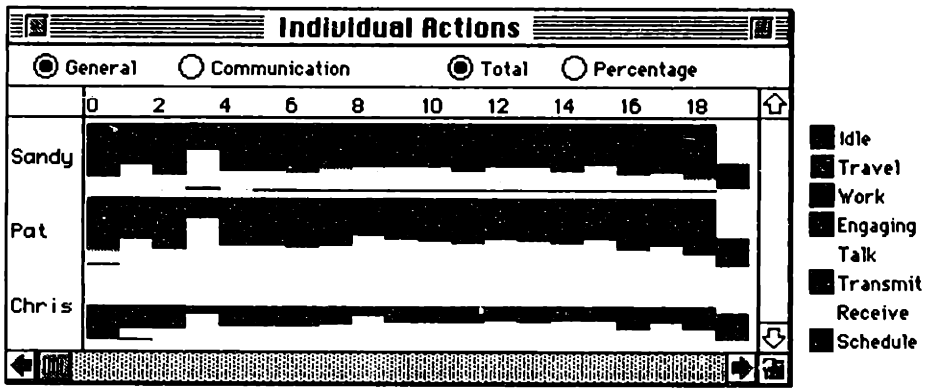


Figure 5.18: Time trace of agent actions. The amount of time each agent spent doing each type of action per day is recorded. A full vertical bar represents a nine-hour work day. The majority of time is spent either idle, in work, or communicating. The original diagram is in color.

5.3 Results of the Simple Example

At the termination of the simulation run we can examine results in both a graphical format and from textual output files. Figure 5.17 records the hours of work spent on each task per day. It clearly illustrates when tasks started and stopped; for example, there is a four day gap in the DESIGN task where Sandy had run out of things to do and had to wait until the REVIEW task could start. In this particular run of the simulation, the ANALYSIS PROGRAM task was the final one to terminate.

A second method of looking at time-trace results is to consider what each individual did during the course of the simulation. Figure 5.18 displays the types of activities each agent engaged in during each day. Note how these activities match up with those recorded in figure 5.17. Sandy spent the first four days working on the DESIGN task and communicating information to the other agents. Pat and Chris started working in earnest on their respective tasks as of day 2. This figure shows clearly that Chris is only working 50% time and that almost half of that time is spent communicating.

In conjunction with the time traces, figure 5.19 shows the overall percentage of time spent by the entire team of agents during the course of the simulation, whereas figures 5.20, 5.21, and 5.22 show the individual percentages of action time. These figures clearly show that the bulk of the agent's time is spent either communicating with each other or working individually. Traveling between offices and sending and receiving messages make up the remainder (with a very small amount devoted towards scheduling meetings or starting conversations).

The types of messages being exchanged by agents are recorded when they are talking (that is, in synchronous conversation, not asynchronous). Figure 5.23 shows a time trace of the types of messages exchanged during conversations by the agent and days they occurred on. This

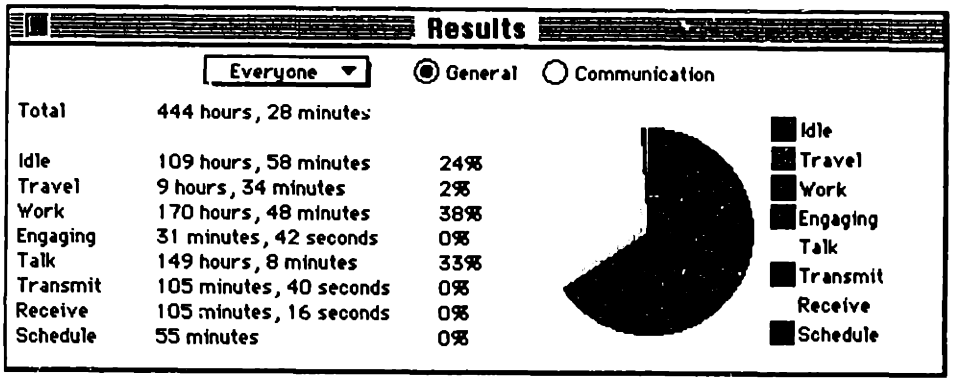


Figure 5.19: Pie chart of time spent on agent actions. The individual time each agent spent on each type of action has been collected together and displayed in a single pie chart. The original diagram is in color.

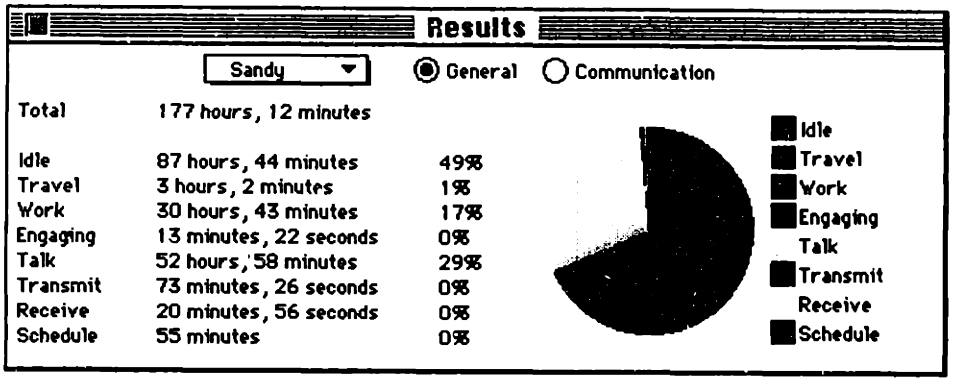


Figure 5.20: Pie chart of time spent on Sandy's actions. The original diagram is in color.

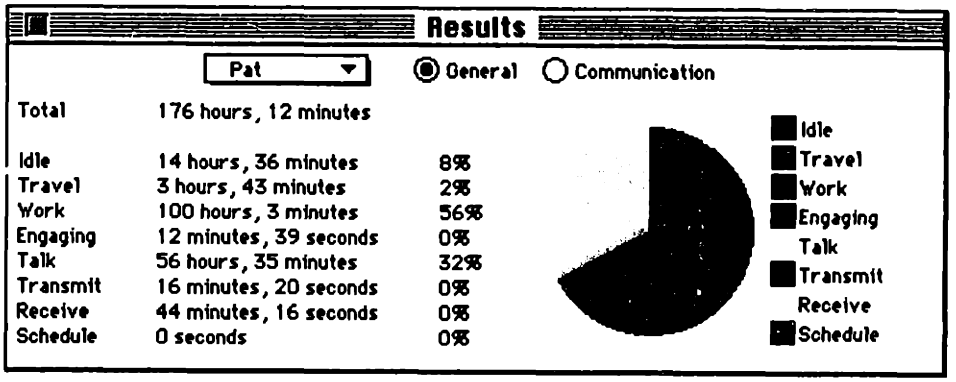


Figure 5.21: Pie chart of time spent on Pat's actions. The original diagram is in color.

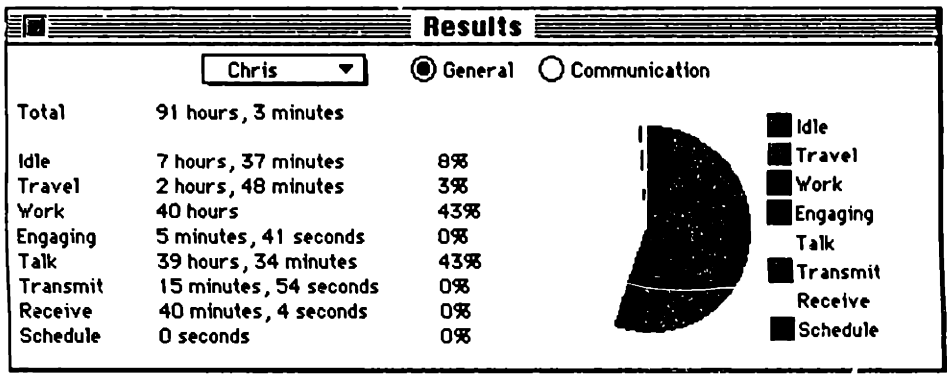


Figure 5.22: Pie chart of time spent on Chris's actions. The original diagram is in color.

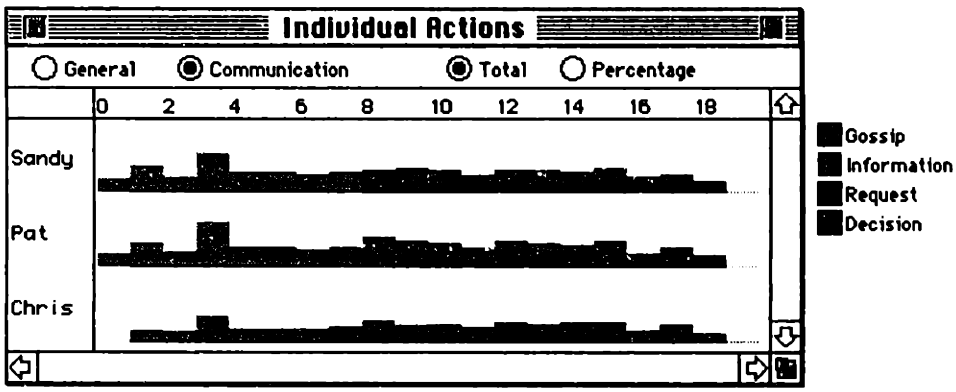


Figure 5.23: Time trace of agent communication actions. The amount of time devoted in conversation to a particular type of message by agent is recorded. A full vertical bar represents a nine-hour work day; as most agents spent considerably less than that in conversation, their vertical bars are shorter. The original diagram is in color.

figure uses the same scale as figure 5.18. To help visualize types of communications that are going on, the scale of the figure can be changed to fill out the vertical bars. Figure 5.24 shows this effect; here the amount of time spent exchanging each type of message has been scaled by the total amount of time spent in conversation for that day and agent. Note that the displayed message types represent only the actual messages that were spoken in the conversation, not the ones that agents attempted to speak but were not sent because another agent specified a higher volume. Figure 5.24 clearly shows the days that the agents gathered for joint discussions of the REVIEW task (days 8 through 18).

Figure 5.25 displays a pie chart for the percentages of the different types of messages that were exchanged by all agents during the course of the simulation run. The chart demonstrates that a low percentage of the agent's time was spent gossiping, and that the bulk of the time was spent exchanging information.

Output data files from the simulation may be used to extract other interesting results and statistics. For example, a total of 11 meetings were scheduled and held (regarding the REVIEW task), averaging one hour and five minutes each. 54.5 hours were spent in 55 conversations, with either 2 or 3 agents per conversation. The longest conversation lasted two hours (it was probably a scheduled meeting) and the average conversation duration was about an hour. 13 phone calls took place, averaging 22 minutes apiece. This disparity between the number of conversations and telephone calls is the result of the high cost of transmitting information over the phone lines (83.3% of the speed of a face-to-face conversation) and the close proximity of the agents.

A total of 88 asynchronous communication events were recorded, consisting of 52 voice-mail

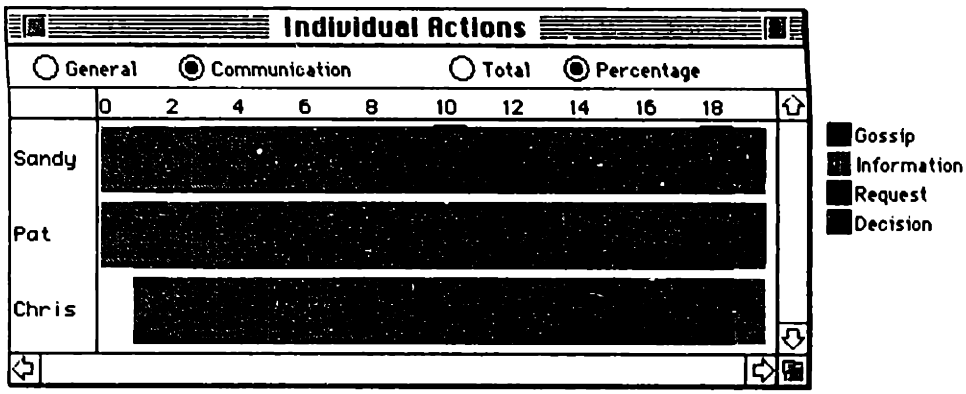


Figure 5.24: Scaled time trace of agent communication actions. The amount of time devoted in conversation to a particular type of message by agent is recorded. Message time is scaled to fill the entire day; hence the height of the colored bars represents the percentage of time spent that day exchanging a particular type of message. The original diagram is in color.

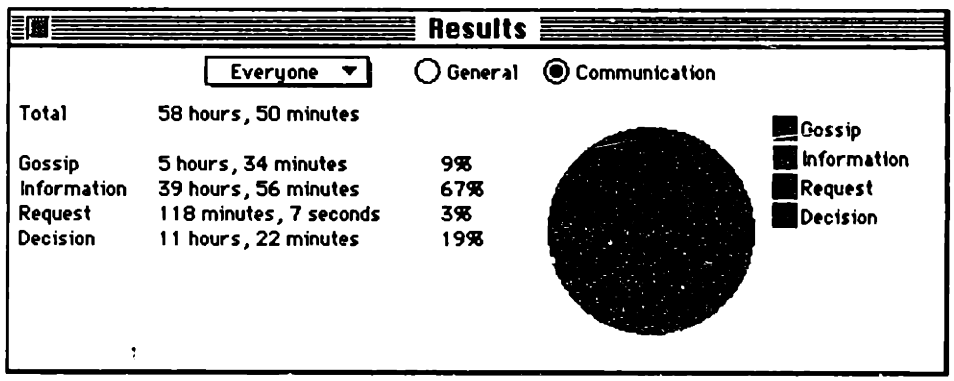


Figure 5.25: Pie chart of time spent on different message types in conversation. The total time each type of message was represented in synchronous communication during the simulation run has been summed. The original diagram is in color.

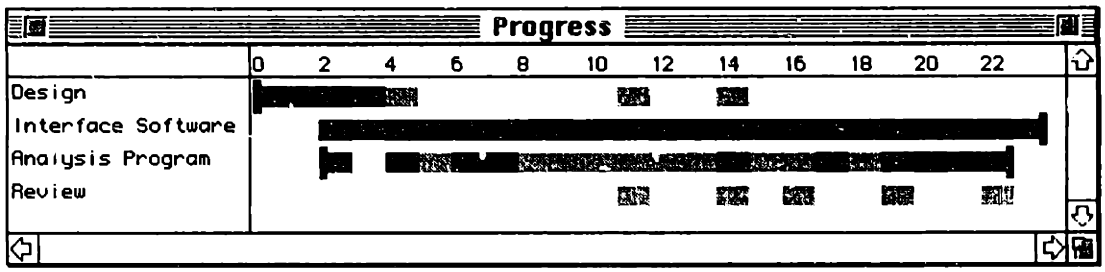


Figure 5.26: Time trace of work with increase in INTERFACE SOFTWARE task size. Darker regions represent more hours of work done per day.

transmissions and 36 formal mail transmissions. Most of these transmissions were short (voice-mail is limited to two minutes), but the formal mail was used to transfer the small quantity of formal documentation that was generated during the course of this project (formal mail only takes 40 seconds to send, so it's a big win for the sender).

5.4 Variations on the Simple Example

After running the simulation once or twice, it's natural to start asking "what if?" questions. What if the task sizes are a little different? What if the nature of the dependencies change? What if different types of communication are available? This section will explore just a few of the many possible "what if?" questions.

5.4.1 Variation in Task Size

Estimating the sizes of tasks is a tricky business, particularly for tasks that involve writing software. Let's assume that the INTERFACE SOFTWARE task was underestimated and that the real work time is 120 hours (instead of 100 hours) and that the real comm time is 60 hours (instead of 40 hours). This is an increase of 20 work hours and an increase in true communication time of approximately 8.5 hours (based on dependency scope and thoroughness). That should translate into a little over three additional days of work.

Running the simulation with these changes results in a new finish time of 11:51 A.M. on day 23 (4 days later than the original). Examining the time trace of the tasks (figure 5.26) reveals that the start of the REVIEW task was delayed by a few days. This matches well with the rough prediction.

Curiously enough, the increase in size of the INTERFACE SOFTWARE task did not apparently free up any of Chris's time (see figure 5.27). Chris was the limiting factor in the original design, so one would suspect that increasing the work to be done by Pat would free up some of Chris's time. In fact, the extra communication required between Pat and Chris effectively eliminates any extra time that Chris would gain from the longer project duration.

5.4.2 Variation in Structure

Consider a more substantial variation in the task structure. Assume that it is important that the project be finished in a shorter amount of time than 20 days. One approach to shortening the project would be to divide the INTERFACE SOFTWARE task into two subtasks and assign a unique person to each subtask. This has the potential of reducing the duration of the INTERFACE SOFTWARE task by 8 days or so. If simultaneously the person assigned to the ANALYSIS PROGRAM task is increased to full-time, it might be possible to reduce the overall task duration to around 13 days.

The division of the INTERFACE SOFTWARE task will create two tasks that are strongly coupled to each other. Assume that the INTERFACE SOFTWARE task is divided into one task

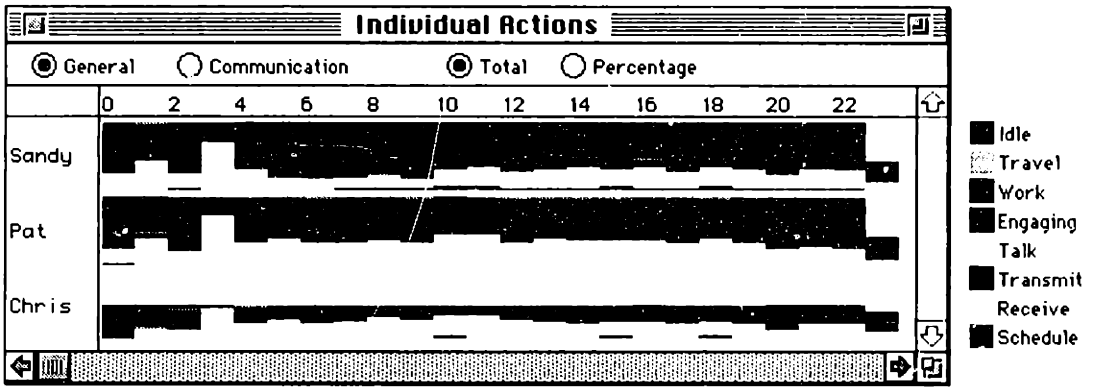


Figure 5.27: Time trace of agent actions with an increase in INTERFACE SOFTWARE task size. The original diagram is in color.

that handles the user interaction and display (INTERFACE GRAPHICS), and a second task that will handle dialog boxes, file input and output, and experiment ordering (INTERFACE I/O). The INTERFACE I/O task is assumed to be the portion of the original INTERFACE SOFTWARE task that drove the ANALYSIS PROGRAM task.

Figure 5.28 shows the layout of the divided set of tasks. The total work and communication time associated with the original INTERFACE SOFTWARE task has been divided between the INTERFACE GRAPHICS task and INTERFACE I/O task. The dependencies between INTERFACE SOFTWARE and other tasks have been duplicated, except for the driving dependency from INTERFACE SOFTWARE to ANALYSIS PROGRAM. As ANALYSIS PROGRAM really depends only upon the file structure of the software, the dependency from INTERFACE I/O was left in, but the scope was broadened. The coupling between the two INTERFACE tasks is a tight interdependency with broad scope and high required thoroughness. This will add about 30 hours of communication time to the overall project requirements.

To balance the staffing of the project an additional team member, Sam, was added and Chris was set to be a full-time worker. Figure 5.29 shows the redesigned office layout, where Sam and Chris were assigned into the same room. Sam is assigned responsibility for INTERFACE GRAPHICS and assists on the REVIEW task. Pat is assigned responsibility for INTERFACE I/O and assists on the REVIEW task.

Figures 5.30 and 5.31 are time traces of task progress and agent activity during the course of the divided project. The project duration is 16 days, 4 hours. This is a reduction of almost 3 days from the original project, or a 15% decrease in project duration. The cost of this decrease is an increase in total billable hours. If we assume that idle agents work on projects outside of the scope of the simulation, then the original project took 333 hours of billed agent time. The divided project takes 455 hours of billed agent time, or a 37% increase in billing hours for the 15% decrease in project duration. The increase in speed comes at a great cost in communication time between Pat and Sam.

5.4.3 Variation in Communication

Imagine the "office of the future" where all employees telecommute rather than come to a central office building. How would that affect our sample project? This can be simulated by removing all face-to-face communications between individual agents. Furthermore, assume that in the office of the future, the standard mail, telephone, and conversation communication channels are replaced by a videoconferencing system and a fax machine that can handle both formal and informal documents.

The efficiency parameters of the video conferencing system is assumed to be a little better than a telephone, but still not as good as a face-to-face discussion (message sizes increase by

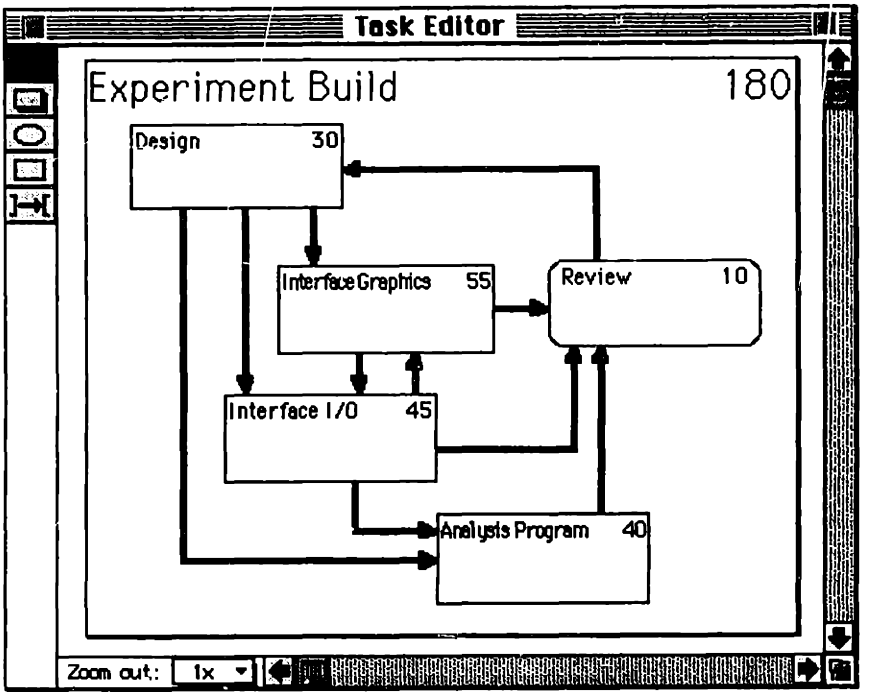


Figure 5.28: Structure of the divided project; INTERFACE SOFTWARE has been split into two parts.

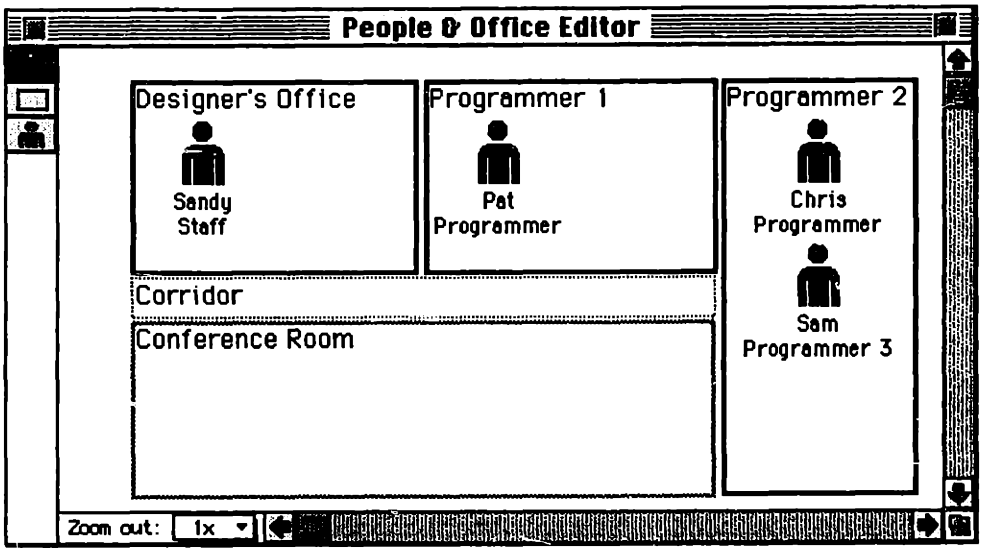


Figure 5.29: Office assignments of divided project with an additional team member.

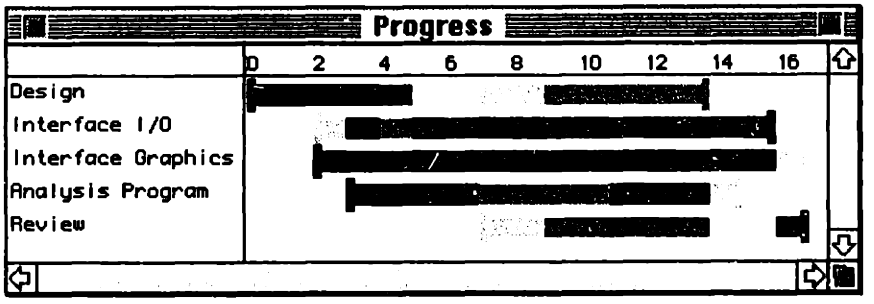


Figure 5.30: Time trace of divided project.

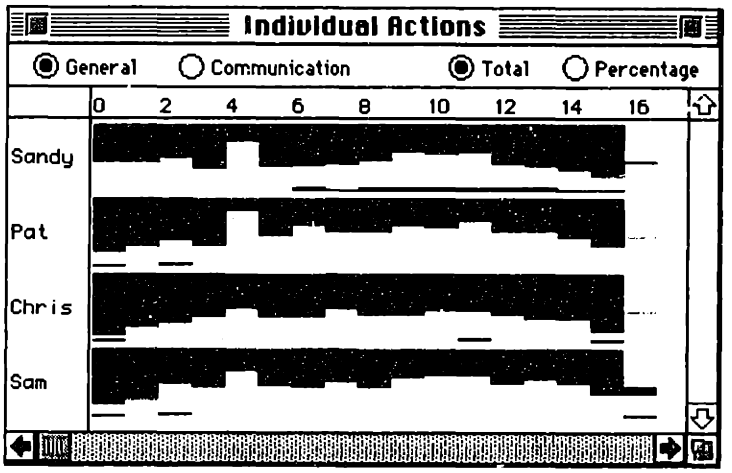


Figure 5.31: Time trace of agent activities in divided project.

10%). It is also assumed that video conferencing is difficult to set up and requires two minutes of effort to contact another agent. Video conferencing does allow more than two agents to participate (otherwise there could be no team meetings), but it does not allow the easy joining and leaving by other participants. This is in opposition to normal conversation where agents in the room may join or depart the conversation as they choose.

The fax system handles all formal and informal documents and requires a small quantity of time to transmit them (about 25% of nominal message size) as well as a large amount of time to compose informal documents (about 200% of nominal message size). However, agents read faxes quickly (200% of normal rate), so there are some potential advantages to sending formal information via fax channels.

The teleconferencing project is complete on day 21 at 9:51 A.M. This represents an increase of only two days over the original project. Figures 5.32 and 5.33 show the time traces of what happened during the course of the work on the project. Figure 5.34 displays the breakdown of how individuals spent their time. Compare this figure with figure 5.19 on page 101. Note that the number of billable hours increased only slightly, from 333 hours to 340 hours. The travel time decreased by a third and the asynchronous communications dropped by 15%. The only big increase was in the "Engaging" activity, which is the time spent by an agent opening up a synchronous communication. This increased because of the extra time requirements to start a videoconference.

The output data file for the videoconferencing example demonstrates some of the differences between telecommuting and face-to-face discussions. For example, there were 44 scheduled meetings with telecommuters versus 11 for the normal office. But the average duration of a meeting dropped from an hour and five minutes to just 24 minutes. 152 video discussions took place, but over 355 were attempted. Because it is not possible to add an additional agent to an ongoing video discussion (and because there were only three agents), the odd agent was left in the awkward position of disrupting ongoing conversations or being ignored (a real video-conferencing system would probably have a way of adding additional callers).

5.5 Complex Example

5.5.1 Project Description

Figures 5.35 and 5.36 are an example of a more complex engineering project; a flight simulator development project. This example is drawn loosely from a real flight simulator development project. In this example a team of five engineers and managers work together to develop a simple flight simulator. The simulator consists of hardware and software. The hardware side consists of

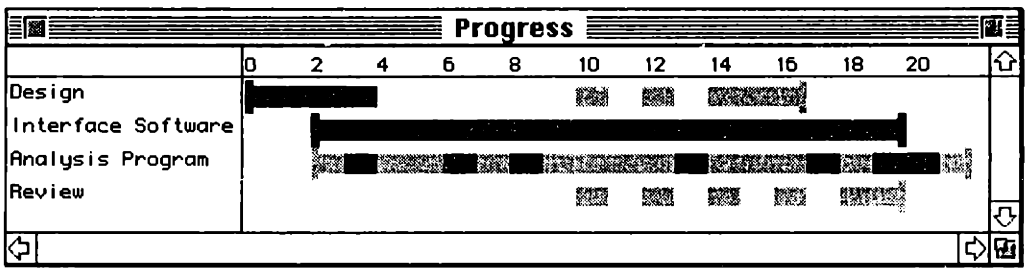


Figure 5.32: Time trace of task progress in teleconferencing project.

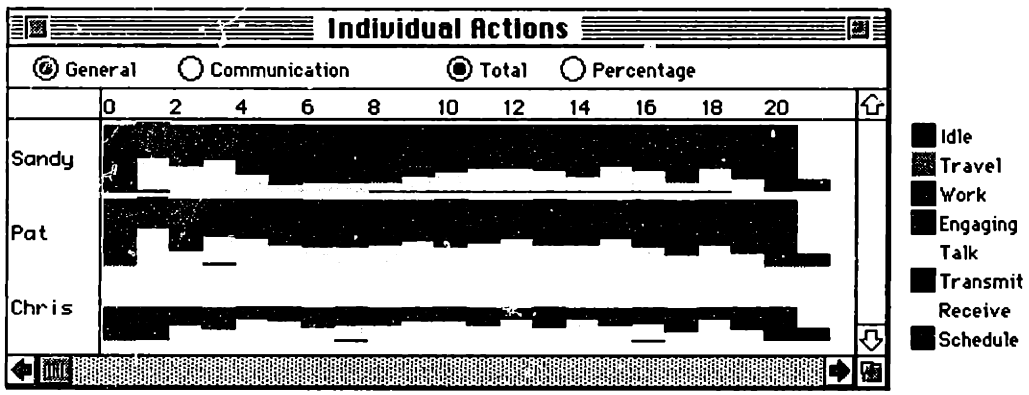


Figure 5.33: Time trace of agent activities in teleconferencing project.

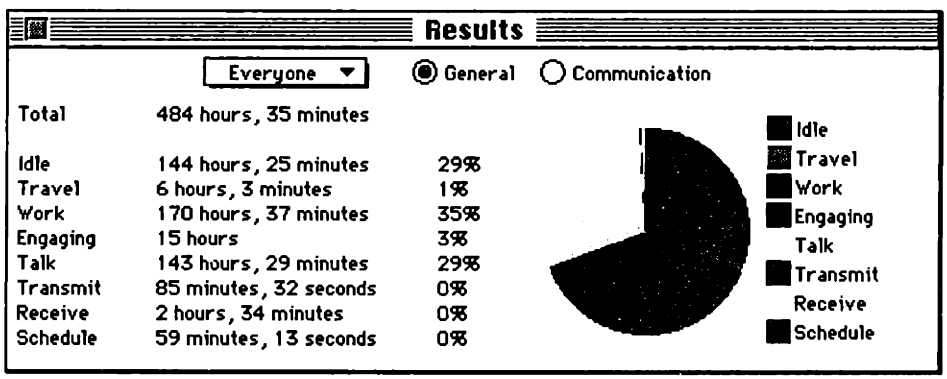


Figure 5.34: Pie chart of collected agent activities in teleconferencing project.

aircraft components mounted in a room (control yoke, ailerons, display hardware), and circuitry to connect the components to a computer. The software side consists of modifying an existing flight simulator package to accept hardware inputs and creating two virtual instruments to be displayed to the pilot during flight.

The development program consists of four logical parts: conceptual development, software implementation, hardware implementation, and testing and debugging. Task work time is given in each box. Task communication time is roughly 25% to 40% of the task work time. Dependency strength is shown in the Design Structure Matrix (DSM) in figure 5.37. The thickness of the circle shown in each matrix location is proportional to the integral of the completion function for each dependency. Although this figure does not carry scope or thoroughness information, it does give a sense of the effect of the various dependencies.

The project personnel are divided between a high-level manager (Edward) who also tracks the software development, two engineers (Sandy, a software engineer, and Chris, a hardware engineer), and two technicians (Joe, a computer programmer, and Sue, a hardware technician). Job assignments for these individuals are shown in figure 5.35; names in bold refer to the agent who is responsible for the task. All individuals are assumed to work full time at normal working efficiency and communication efficiency.

5.5.2 Results

The SIMULATION DEVELOPMENT project finishes on day 36 at 1:55 P.M. Figures 5.38, 5.39, and 5.40 show the standard time traces for how long the project takes and what the individuals do during that time.

The initial CONCEPT task slowed project progress at the beginning of the simulation. The CONCEPT task is a tightly coupled collection of three tasks. The time traces show that the three involved agents (Sandy, Chris, and Edward) are not fully engaged with work during this period. They are not working to their maximum potential because they are required to schedule group decision meetings. Ideally they'd work apart for an hour, work together for an hour, work apart for an hour, and so forth. This type of "tight" scheduling of meetings is not allowed in the *DiFS* simulation. If there were only *two* agents involved in these tasks or if all three shared a common office, the project would proceed faster because they could casually gather to discuss the group decision task.

A similar coordination problem occurs during the last few days of the simulation. Here the problem is the coordination of the four tasks SOFTWARE DEBUG, SHAKEDOWNS, HARDWARE DEBUG, and REVIEW. The REVIEW task is small and yet coupled tightly to the SHAKEDOWNS task. It forces the final days of work to be spread out so that appropriate meetings can be scheduled.

Figure 5.39 clearly points out that Joe is a key player in finishing this project on time. The other agents appear to have large amounts of free time. This suggests that perhaps the number of agents working on the simulation could be reduced.

5.5.3 Smaller Project Team

SIMULATOR DEVELOPMENT PROJECT!VARIATION

In an attempt to reduce the project duration, team leader Edward was deleted from the staff. Sandy was assigned all of Edward's responsibilities and took over as team leader. Figures 5.41, 5.42, and 5.43 show progress in the "small team" version of the SIMULATOR DEVELOPMENT project.

The small-team project duration was 34 working days, or a reduction of 2.5 days. The billable hours dropped from 680 to 628, or a reduction of 8%. This is an example of a project where the extra team members actually slowed the overall project simply by requiring more communication, more meetings, and generally more work. Of course, in a real project the team leader often functions in multiple capacities, capacities that have not been modeled in this particular project.

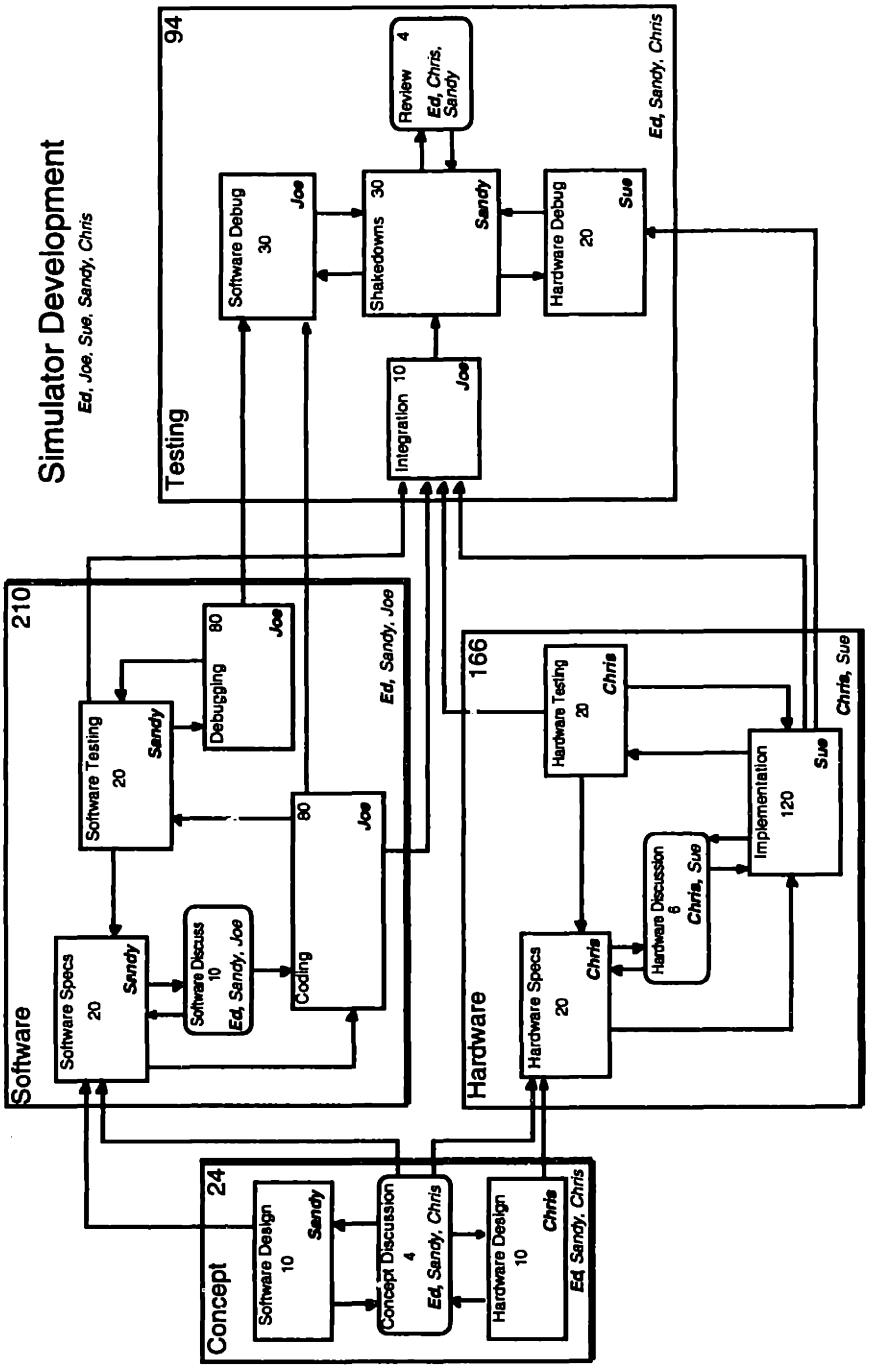


Figure 5.35: SIMULATOR DEVELOPMENT project.

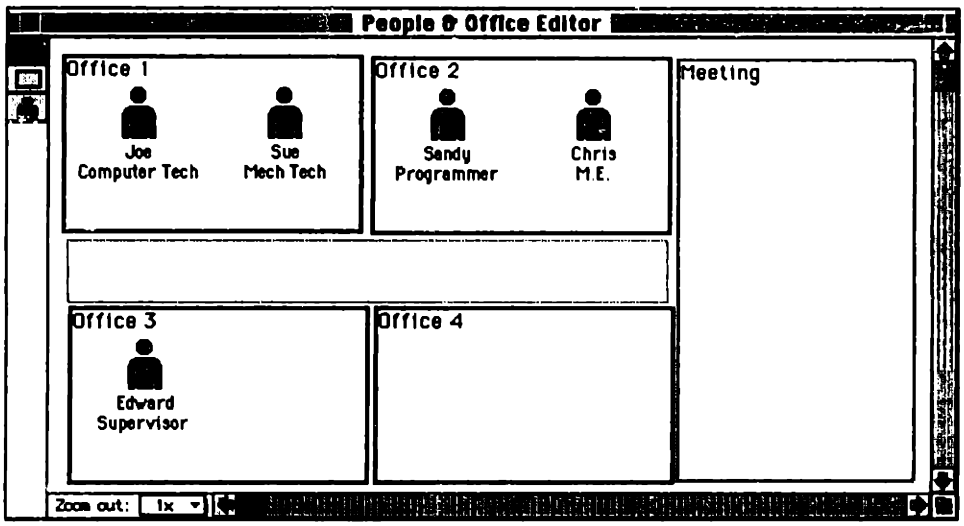


Figure 5.36: SIMULATOR DEVELOPMENT office environment.

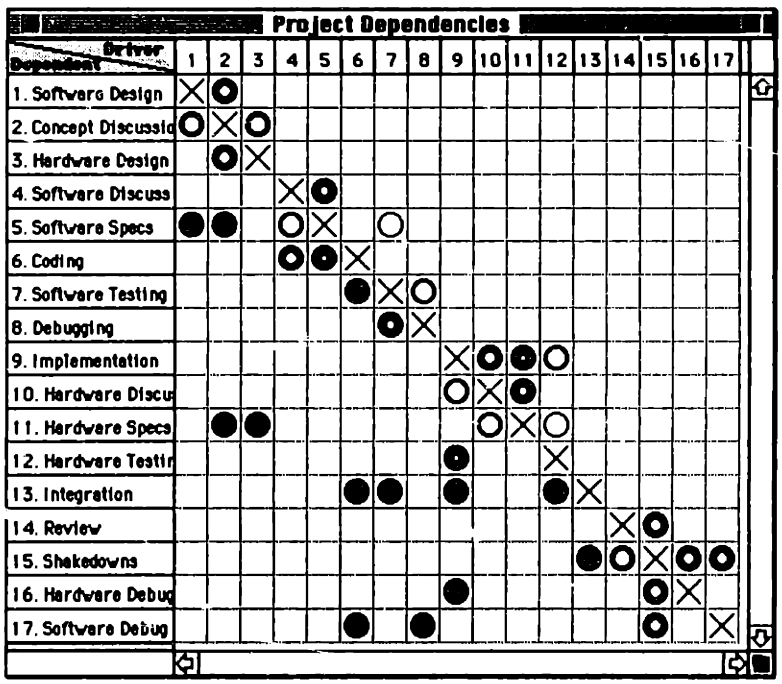


Figure 5.37: SIMULATOR DEVELOPMENT design structure matrix. Circle thickness is proportional to the integral of the dependency completion function.

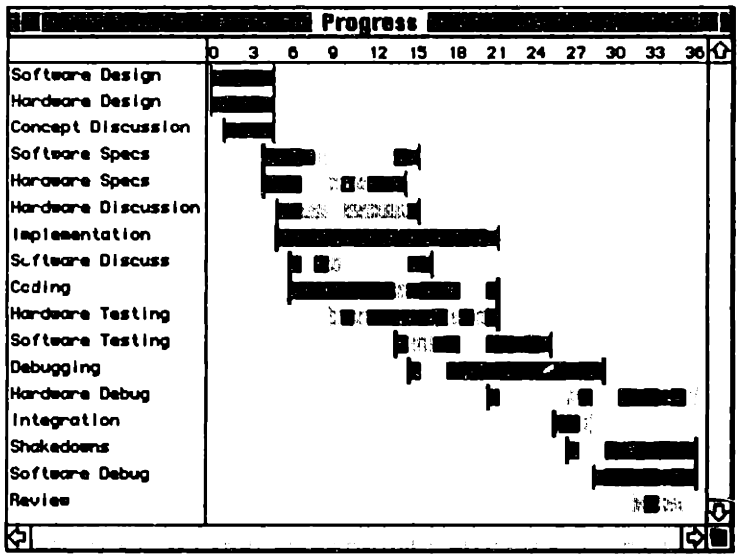


Figure 5.38: Time trace of the task progress in the SIMULATOR DEVELOPMENT project.

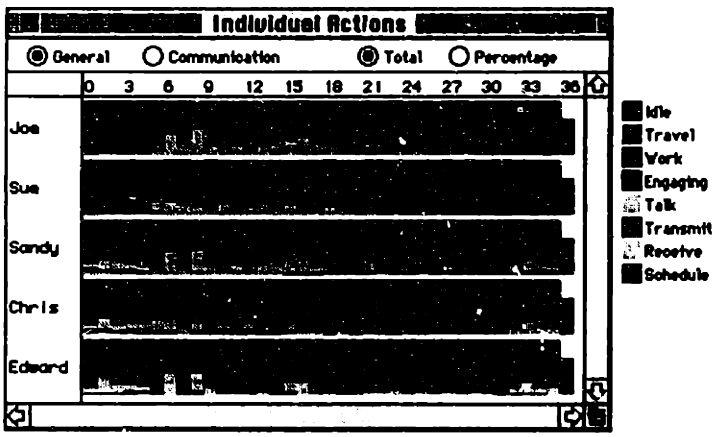


Figure 5.39: Time trace of the agent activity in the SIMULATOR DEVELOPMENT project.

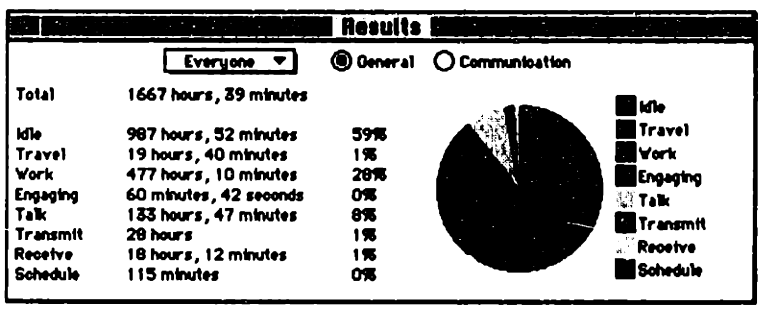


Figure 5.40: Pie chart of agent activity times for the SIMULATOR DEVELOPMENT project.

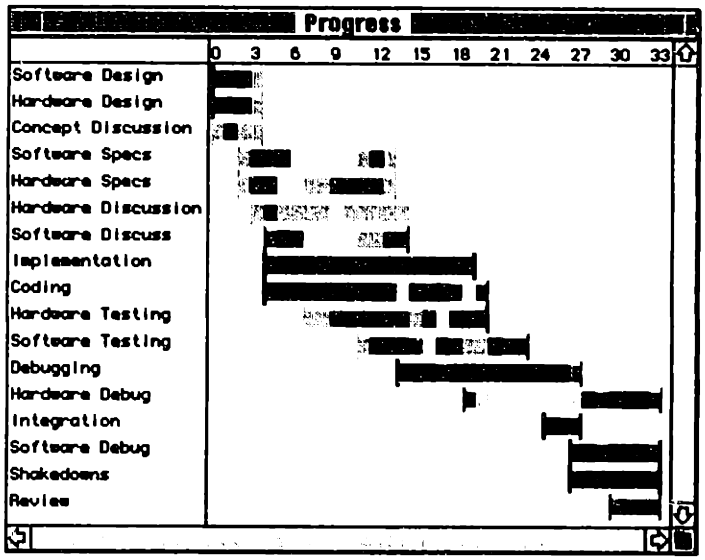


Figure 5.41: Time trace of the task progress in the small team SIMULATOR DEVELOPMENT project.

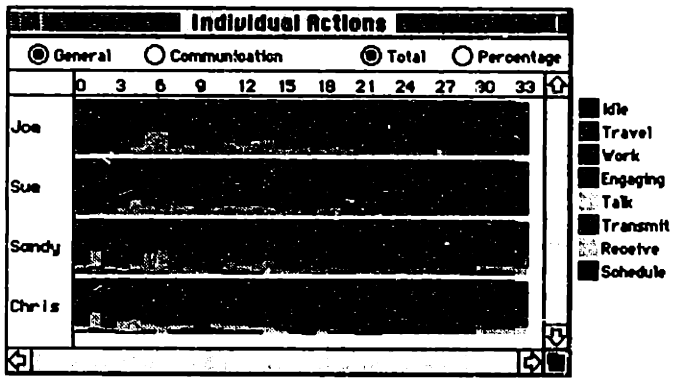


Figure 5.42: Time trace of the agent activity in the small team SIMULATOR DEVELOPMENT project.

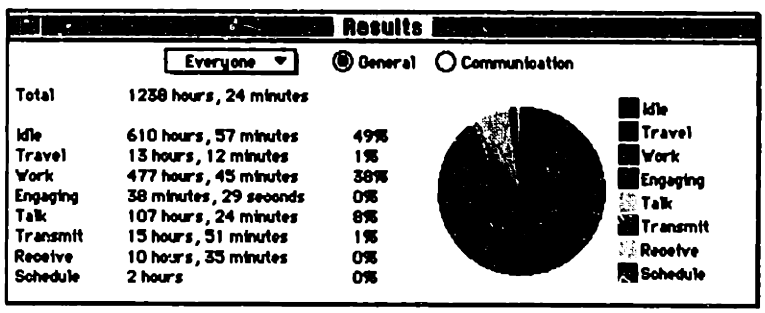


Figure 5.43: Pie chart of agent activity times for the small team SIMULATOR DEVELOPMENT project.

5.6 Summary

This chapter has presented examples of *DiFS* simulation models. The human-factors project model was examined in detail, both by what happens during a regular simulation run and what happens when the project varies size, topology, or communication methods. A larger model of a flight-simulation project demonstrated that the *DiFS* simulation can model a more complex project. It also demonstrated how removing team members from a project may improve project performance.

The remaining chapters of this thesis explore experimental methods with the *DiFS* simulation (chapter 6), a validation study (chapter 7), and applications of the simulation to studying management questions (chapter 8).

Chapter 6

Experimentation

Prior chapters have described the *DiFS* model and the *DiFS* simulation. The preceding chapter (chapter 5) demonstrated some typical simulation runs using a variety of different project setups. Many of these simulation runs were perturbations of each other, demonstrating what happens when small changes are made to the project model. But there are a number of unanswered experimental questions:

1. How can we use the simulation to run controlled experiments?
2. Is the simulation robust to small variations in project input?
3. Does the simulation behave in a reasonable and predictable fashion?
4. How do we know that the human behavior model is well behaved?

This chapter attempts to address these questions, although part of question 3 will be addressed in the validation experiment of chapter 7. This chapter is divided into three sections: a discussion of how experiments can be run with the *DiFS* simulation, a discussion of the effects of variations in agent behavior, and a discussion of how perturbations to the project model or agent behavior affect project performance.

6.1 Running Experiments

This section discusses how the simulation is "exercised"; that is, how a person can run experiments using the *DiFS* simulation to extract useful results. Running an experiment with the *DiFS* simulation requires: (1) a planned set of trials (or projects) to execute, (2) a stopping criteria for when a trial has finished, (3) recording appropriate performance metrics for each trial, and (4) analysis of the data.

Setting up and running a particular project simulation can be done completely by hand, although this is a time consuming procedure. A faster way of running multiple trials is by using an AppleScript [6]. AppleScript is a language that interacts with Macintosh programs via high-level events. The *DiFS* simulation responds to a variety of high-level events that can be used to open and close files, modify task and agent parameters, set breakpoints, run the simulation, extract simulation results, and store them to a file. AppleScript support in *DiFS* allows the user to write a short program that systematically varies task parameters, runs multiple trials, and stores the results. Section A.3 contains a description of the AppleScript support in *DiFS*.

To run an experiment, the experimenter selects a suitable stopping criteria and one or more performance metrics. The standard stopping criteria used throughout this thesis is when the agent responsible for the top-level management task is aware that all tasks in the simulation have finished. This stopping criteria is normally combined with the performance metric of project duration, where project duration is measured in terms of working hours since the start

of the simulation. Because the agents in the simulation work nine hours per day, the project duration may be readily converted into the total number of days required to finish the project.

The project duration stopping criteria is used in the majority of experiments reported in this thesis primarily because it is easy to visualize. Other stopping criteria and performance metrics are possible. For example, another stopping criteria would be the completion of a particular task. A matching performance metric would be the duration of that task. Another useful performance metric is the billable hours measure; that is, the number of hours that agents within the simulation have worked. This is measured within the simulation as the difference between the total number of hours logged by agents during the simulation run less the total number of hours logged as idle time. The assumption is that an idle agent is one who could profitably spend his/her time working on a project that is out of the scope of this simulation.

The standard experimental procedure is as follows: First a project model is created. Second, the parameters the experimenter wishes to test are selected and suitable values for those parameters are chosen. Third, the experimenter writes a short AppleScript that systematically modifies the parameters, runs simulation trials, and stores results to a data file. Fourth, the data file is statistically analyzed, normally in a spreadsheet package like Microsoft Excel. For the record, the complex project example presented in section 5.5 on page 107 takes approximately 18 seconds to run on a Power Macintosh 6100/60.[1] A typical experiment that varies a single factor to one of three different levels and runs 10 repetitions per level will take 9 minutes to execute. The largest experiments have run up to 12 hours.

Statistical data results will be presented where appropriate for the different experiments. As the performance metric typically used is the total project duration, most charts will plot the mean project duration plus or minus two standard errors. The standard error is the standard deviation of the estimated *mean* of a set of trials; it is calculated as $s_e = s/\sqrt{n}$ where s is the standard deviation of the population and n is the number of trials. The interval $(\overline{y} - 2s_e, \overline{y} + 2s_e)$ is a 92% confidence interval; that is, 92% of the time the actual mean should lie within this range. Thoughtful, well-written information on significance tests, standard errors, and experimental design may be found in Box, Hunter, and Hunter [8].

6.2 Sensitivity to Agent Choice

The *DiFS* simulation postulates a world where individual agents do exactly one thing at a time. This is a substantially different approach than one utilized by System Dynamics and related fields (see [2, 29]). It introduces a tricky question: Will different arrangements of agent actions result in substantially different results? A small variation in project conditions could potentially result in substantially different chains of activities.

This section addresses the issue of simulation stability in the face of agent behavior that has been slightly randomized.

6.2.1 Stochastic Variation in Utility

Computer agents in *DiFS* can only do a single action at a time. Hence the sequence of their actions may potentially strongly affect the overall duration of a particular simulation run. This sequence of actions is determined by the agent's personal behavior model and memory and involves a large number of calculations of the relative utilities of different actions. A small variation in the utility calculations could result in a different action being taken or the same action having a slightly shorter or longer duration. This introduces a different state of knowledge in the agent and the two "streams of action" quickly diverge.

Having variations in the actions that agents take is not necessarily a bad thing for a simulation of engineering behavior; all of us at one time or another have experienced days where

[1] During the 30+ simulated days, approximately 15,600 agent actions are selected. The total simulation run took about 18 seconds, so almost 1000 agent decisions need to be made every second of real computer time. Optimizing the run time of the code was a high priority.

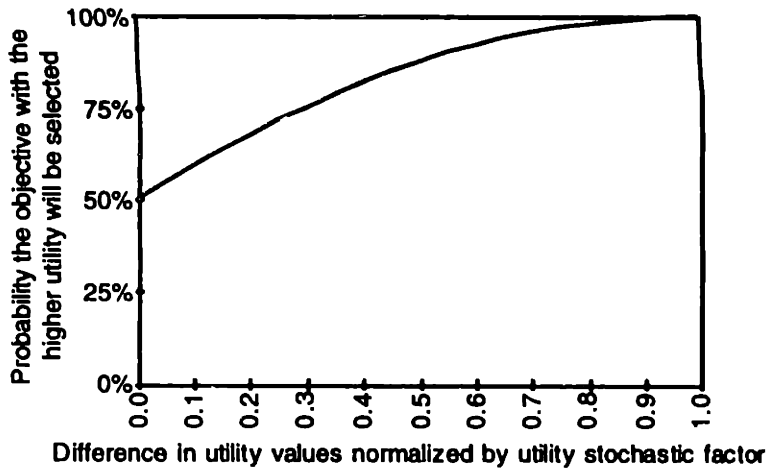


Figure 6.1: The probability that the objective with the higher utility will be selected.

catching or missing a telephone call resulted in a significantly different day than expected. The action variation is only a source of trouble if it is large or strongly dependent upon project parameters. If slight variations in a single task size result in large, unpredictable variations in project duration, then the simulation would be unusable.

The approach used is to add stochastic variation to the agent's calculation of the utility of objectives. The stochastic variation (described in section 4.5.4 on page 85) introduces a small amount of randomness into an agent's calculation of utility for each of the possible objectives the agent is considering. Specifically, a utility perturbation is randomly selected from the range $(0, U_{SV})$ and is added to utility that the agent calculates for each objective. Instead of always selecting the objective with the highest utility value, now the possibility exists of the agent selecting a slightly less useful objective. Hence any two runs of the simulation will result in a different series of actions being selected by agents. Figure 6.1 is a plot of the probability that an agent will select the objective with the "true" higher utility given that a small perturbation (between 0 and U_{SV}) has been added to each. For example, if the utility of working on task A is 3.5 and the utility of working on task B is 3.8 and $U_{SV} = 0.6$ then the normalized difference is $(3.8 - 3.5)/0.6$ or 0.5, resulting a probability of selecting task B of 87.5%.

The controlled nature of the stochastic variation of the agent's utility calculations is valuable: no parameters in the project change from trial to trial, no personality factors are altered. Each simulation run has the same agents selecting slightly different actions each time. Our intuition suggests that introducing this small amount of variability would result in a small variation in project duration.

Figure 6.2 is a histogram of 1000 repetitions of running the human-factors project (refer to figure 2.1 on page 29 and the first part of chapter 5) with a utility stochastic factor set to 0.5 (or $U_{SV} = 0.5$). The mean project duration was 178.6 hours (or day 19, around 3:30 P.M.), with a standard deviation of 3.3 hours and a standard error of 0.1 hours. For reference, the project duration with $U_{SV} = 0$ is 173.7 hours (day 19, 10:48 A.M.). The histogram clearly shows the bimodal distribution of run times. This bimodality arises from the scheduling of meetings and because one agent only works in the morning. Work in the REVIEW task is accomplished in team meetings, and those meetings are normally scheduled at 8:00 A.M. each morning until the task is accomplished. During these meetings the agents in the simulation have an excellent opportunity to exchange information. The other contributing factor is the half-time work schedule of one of the agents; if that agent is holding up the completion of the project, then clearly the project will not be able to finish while that agent is out of the office. Hence the project will prefer to finish in the mornings when that agent is in the office, available for communication, and possibly attending a meeting.

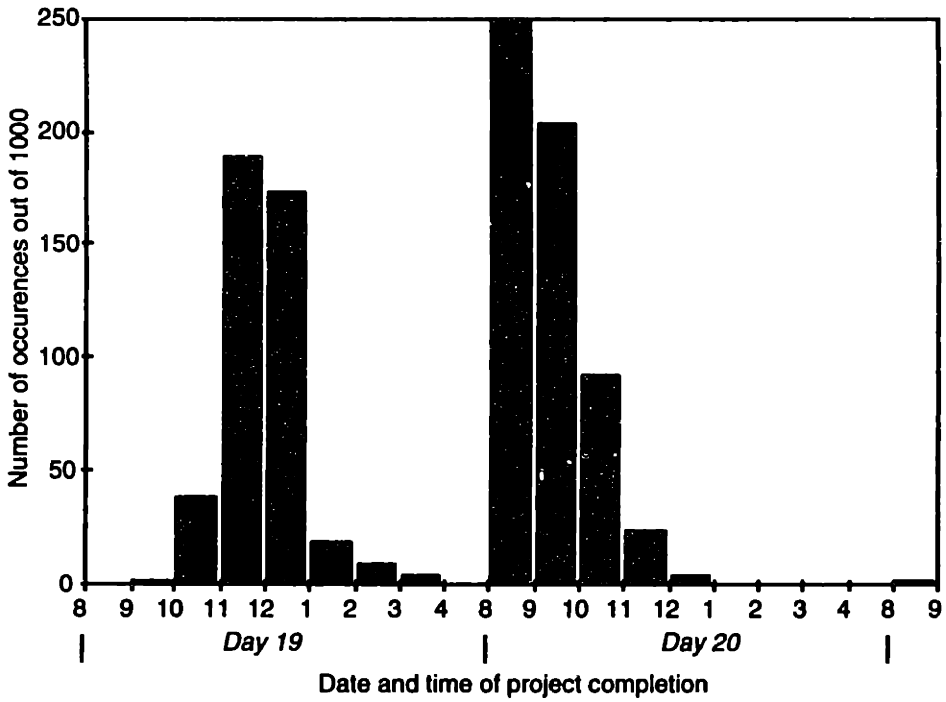


Figure 6.2: Histogram of 1000 repetitions of the human-factors project ($U_{SV} = 0.5$). Non-random trial finishes on day 19 at 10:48 A.M.

6.2.2 Magnitude of Stochastic Variation

Given that the introduction of a small amount of utility variation (U_{SV}) resulted in a two-day wide interval of project duration, it is natural to ask if there is a correlation with utility variation and project duration. The goal is to introduce the "right" level of utility variation; just enough to reflect the real uncertainties that occur with people, but not so much so that the agents behave in an irrational fashion.

Figure 6.3 was created by running the human-factors project at different levels of the utility stochastic factor (U_{SV}). Twenty repetitions of the simulation were run at each of the settings of U_{SV}. Mean project duration plus or minus two standard errors are plotted for each level of U_{SV}. Figure 6.4 displays the raw data plotted in figure 6.3. This figure clearly shows the banding effect produced because this particular project is more likely to finish in the morning hours.

The standard deviation of project duration does not vary greatly as U_{SV} is increased (although it does appear to increase at large values of U_{SV}). The mean project duration does increase as U_{SV} is increased, particularly when $U_{SV} > 0.5$ or so. Similar experiments conducted on different projects have yielded similar results; i.e., the project duration increases when $U_{SV} > 0.5$ and the standard deviation usually falls between 3 and 6 hours. As a result of these experiments I have selected the value $U_{SV} = 0.5$ as a standard value to use when running tests. Arguably one could select a lower value and get reasonable results, but I decided to keep it this high to insure that a good variety of agent activities would be represented in the simulation runs.

6.2.3 Scaling Effects

Allowing stochastic variations in utility calculations resulted in a distribution of project durations with a mean of about 180 hours and a standard deviation of 3.3 hours. If this standard deviation is a function of the project size, then we will have trouble experimenting with large

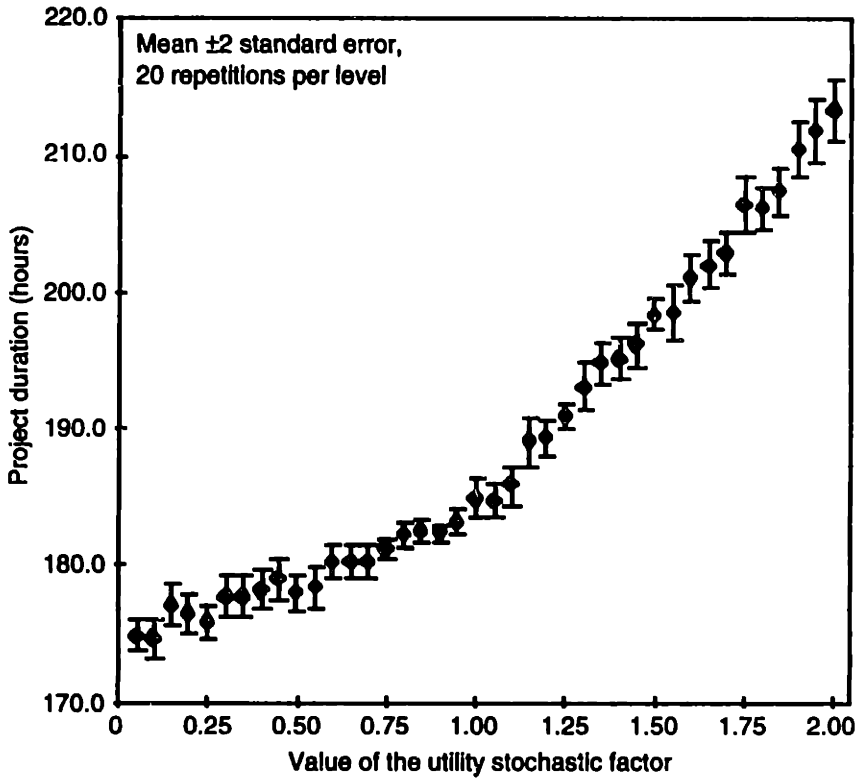


Figure 6.3: Effect of the utility stochastic factor (U_{SV}) on the human-factors project duration. The project duration standard error is 3.3 hours. The standard error does not vary significantly as the utility stochastic factor is changed.

projects because many trials will be needed to accurately measure project durations. We can address this issue by scaling the size of the tasks within the human-factors project (refer to figure 2.1 on page 29 and the first part of chapter 5) and measuring how this changes the project duration. Figure 6.5 is a plot of trials run at different project scales, from 50% to 300% of the normal size. For each data point all work and communication times within the project were scaled by the scaling factor and 20 repetitions were run.

Figure 6.5 shows two nice effects. The first is that the project duration scales linearly with the size of the project. Fitting a linear curve to this graph gives the equation $T_P = kT_0$ where T_P is the project duration, k is the scaling factor, $T_0 = 175.6$ hours, and $R^2 = 0.9994$. The second nice effect is that the standard deviation of project duration does not change significantly as the size of the project is increased.

Stochastic variation in the utility calculations affects the order and type of activities that agents perform. It introduces randomness into the project duration calculations, but the standard deviation of that randomness is well-behaved and appears to represent the natural behavior changes of agents based on the time of day and given limits of information transfer. The advantage of the stochastic variation is that multiple runs provide a distribution of simulation results, allowing a manager to estimate project duration and expected variance of project duration.

6.3 Parametric Variations

Before large simulation runs can be undertaken with confidence, the question of simulation *stability* needs to be addressed. Introducing stochastic variation in the utility calculations has demonstrated that utility calculation variations result in stable trials. From the experimental point of view, the next question is "Do small perturbations of the project model result in

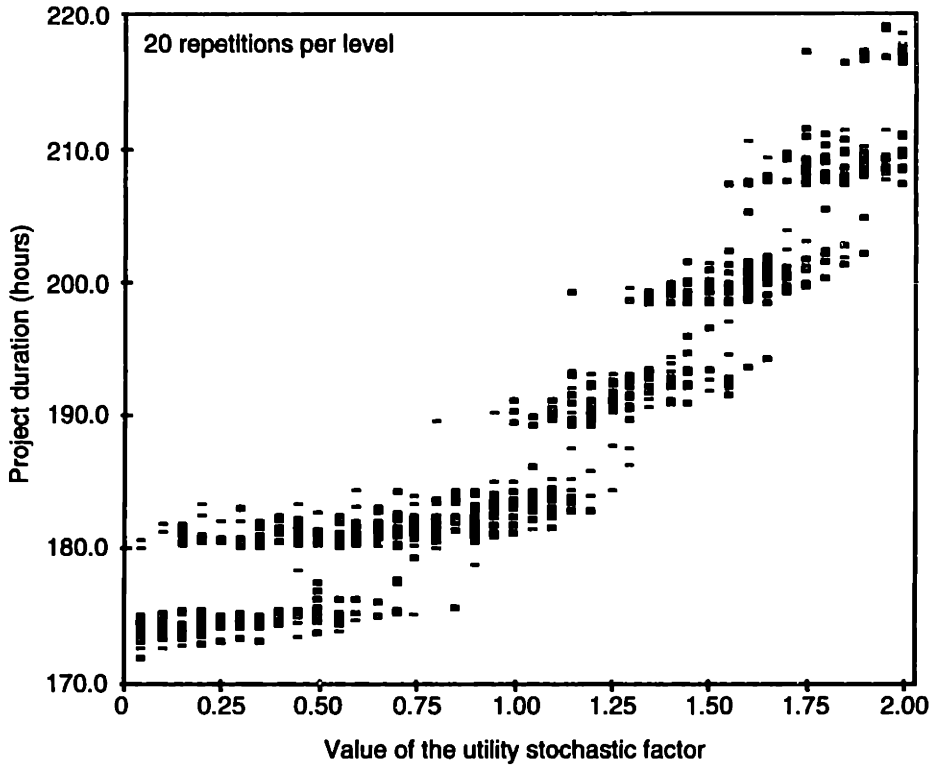


Figure 6.4: Scatter plot of the effect of the utility stochastic factor (U_{SV}) on the human-factors project duration. Each data point represents a single simulation run.

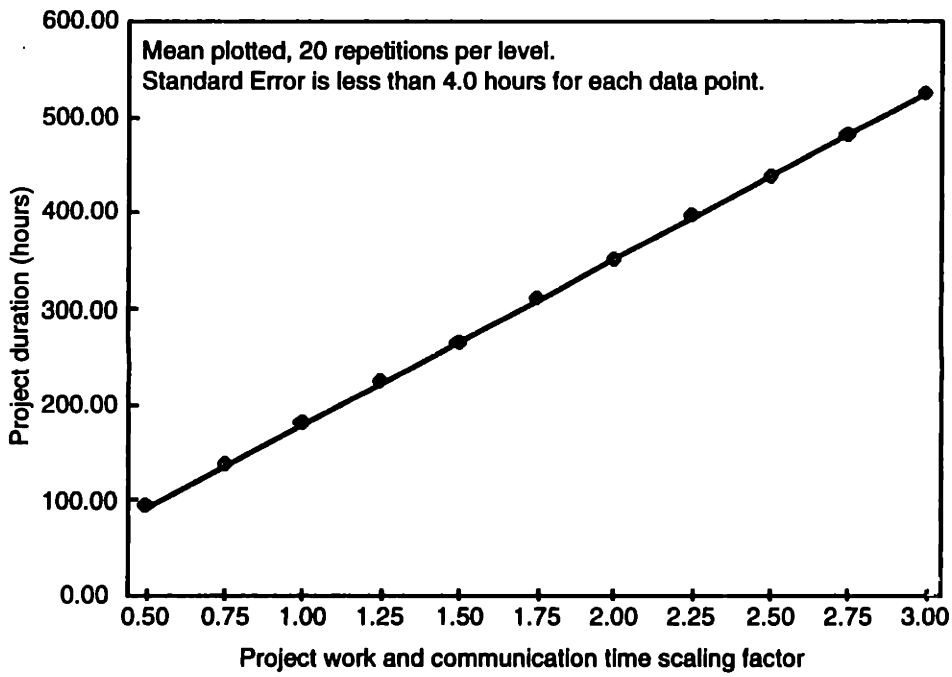


Figure 6.5: Project duration of the scaled human-factors project with $U_{SV} = 0.5$. The work and communication time for each task within the project have been multiplied by the scaling factor. Standard error bars are not plotted for this figure because they would not be visible.

consistent, small variations in the output metrics?" If this is true and if one believes that the model itself is an adequate reflection of reality (a question to be discussed in chapter 7), then it should be possible to use the simulation to make predictions about project performance.

There are two types of project variations to be considered: topological and parametric. A topological variation is a change in the actual structure of the project model. This can be a change in the number of tasks, dependencies, communication methods, or agents. It can also be variations in office layout, room assignments, or job assignments. A parametric variation is a change in the value of a continuous parameter specified in a task, dependency, agent, or communication method.

Parametric variations can be made arbitrarily small, as opposed to topological variations that can not be made arbitrarily small. Hence parametric variations are a good candidate to examine for simulation stability. A small change in the value of a parameter should result in a small change in simulation results.

This section discusses two types of parametric variations: task size and behavior parameters. Other types of parametric variation have been explored; a number of these are discussed in chapter 8.

6.3.1 Task Variation

In theory a small variation in the size of an individual task should result in a small variation in the amount of time required to complete the entire design project. Moreover, the project duration should vary smoothly as a function of the time added to an individual task.

Figure 6.6 shows the results of an experiment that varied task work time and recorded project duration. Each of the four tasks within the human-factors project were individually varied between -50% and $+50\%$ of their nominal size (i.e, work time). The communication time of the tasks was not varied. The stochastic utility variation (U_{SV}) was set to 0.5 and 20 repetitions of each task trial level were recorded. This experimental approach (individually testing a single task at a time) is not statistically efficient and does not examine interactions between tasks, but it is simple to set up and yields clean results without assuming a great deal about the underlying response surface.[2]

Figure 6.6 shows that for small variations in task size, the simulation predicts consistently small variations in project duration. This graph can be misleading because it plots the change in project duration versus the percentage change in task size, but the tasks vary in size from 10 to 100 working hours. Figure 6.7 is a graph of the same data that plots the change project duration versus the absolute change in task size.

These figures demonstrate that project duration is not a linear function of the variation in task size. Increasing the work in the INTERFACE SOFTWARE task increases the project duration, but decreasing the work in the same task does not decrease the project duration. A similar case holds for the ANALYSIS PROGRAM task, except that it needs to be decreased a little bit before the curve levels off. Referring to the sample runs in chapter 5, we note that the agent responsible for the INTERFACE SOFTWARE task is doing it essentially in parallel with the agent responsible for the ANALYSIS PROGRAM task. Moreover, these agents have nothing else to do, accomplish their tasks in roughly the same amount of time, and the entire project waits upon the completion of both of these tasks (a classic case of parallel tasks). In this case we would expect that the project duration would depend upon the slower of the two tasks; hence decreasing the hours in one of the two will result in the other being the slower task. Increasing the work time for one of the two will result in it being the limiting task and hence increase the hours for the overall project.

The general conclusion from this experiment is that the simulation is well behaved with respect to variations in the work time for any given task. Other studies (not presented in this thesis) have demonstrated the same result for variations in parameters associated with

[2] A partial factorial experiment would enable us to test interactions of task sizes, but task interactions are not the focus of this chapter.

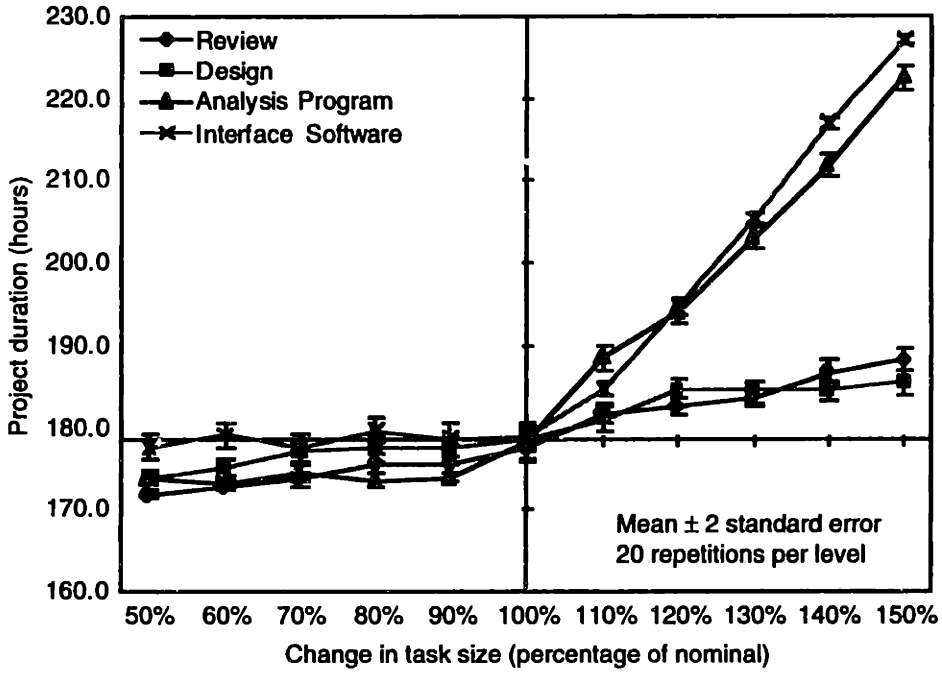


Figure 6.6: Project duration as a function of percentage variation in individual task size.

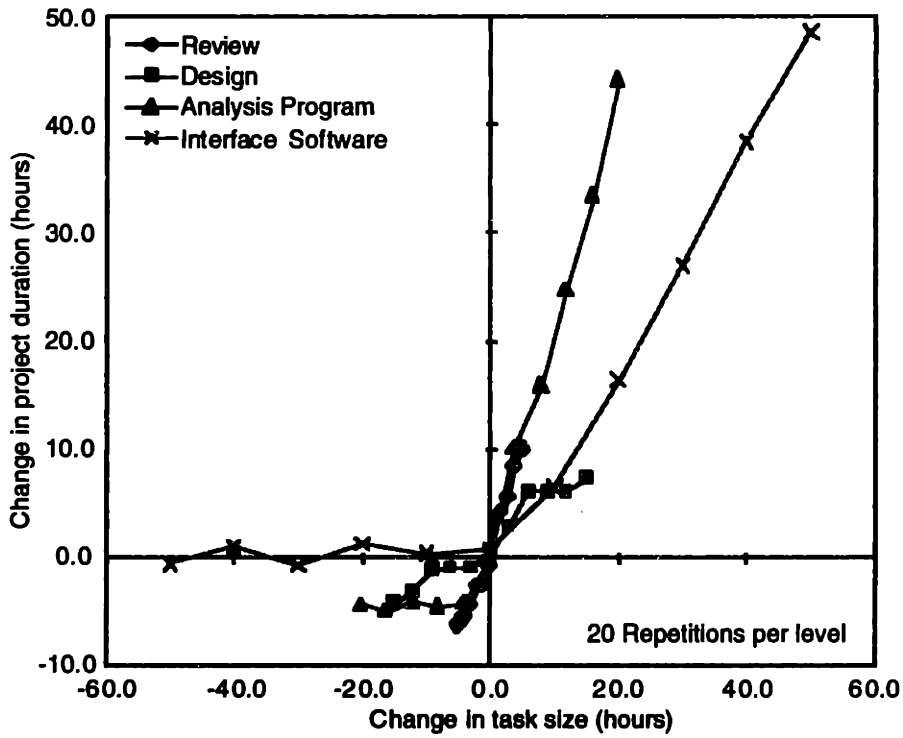


Figure 6.7: Project duration as a function of absolute variation in individual task size.

dependencies, agents, and communications. In all cases we have ended up with well behaved results. A discussion of the managerial interpretation of parametric variations is located in chapter 8.

6.3.2 Agent Behavior Variations

The personal behavior of individual agents is controlled by their calculation of the utility of possible objectives. These utility calculations are made using a set of equations and parameters (see chapter 4 and appendix B). Section 6.2 of this chapter has already demonstrated that a small amount of random variation added to the utility calculations results in a distribution of project durations. It is reasonable to expect that variations in the personal behavior parameters would result in different agent behavior and changes in the project performance. For a given project model, it should be possible to optimize the personal behavior parameters to give the best possible project performance.

Optimizing personal behavior parameters falls into what is know as *Response Surface Methodology* [8, 63, 64]. In a statistical setting, the normal approach is to estimate the local gradient for a particular setting of the behavior parameters and then move "up the hill" until a locally optimal point is reached. Gradient estimation is normally done by a full or partial factorial experiment design that varies each of the possible parameters and estimates their effect. In the *DiFS* personal behavior model there are 32 parameters that directly affect agent behavior (see appendix B.2); a full factorial gradient calculation would require the examination of 3^{32} combinations (assuming that each parameter is set to one of three different levels). Even a partial factorial examination that considered only the first order effects of each parameter and the second order interactions between parameters would require at least $1 + 2(32) + 4(32^2) = 4161$ separate trials.

The solution to the complexity problem is to use an orthogonal array approach [61]. By ignoring the interactions between behavior parameters, the orthogonal array approach only requires $1 + 2(32) = 65$ separate trials to examine each behavior parameter at three separate levels. The L_{81} array is an orthogonal array that specifies three levels of 40 factors and hence for our problem provides 16 degrees of freedom for error estimation at the cost of an extra 16 trials. The price of the orthogonal array approach is that interaction effects are not measured and hence the calculated gradients must be used conservatively.[3] Appendix C discusses statistical calculations with orthogonal arrays.

Figures 6.8 and 6.9 are experimental results for several of the personal behavior parameters obtained by running an L_{81} experiment on the human-factors project. Appendix B contains a discussion of the precise meaning of these parameters. 20 repetitions of each trial has been run. Parameters have to be varied by small steps because large steps may result in pathological agent behaviors. But small parameter steps typically result in small changes in project duration and hence many repetitions must be run in order to get statistically significant results.[4]

A visual inspection of figures 6.8 and 6.9, and of the other behavior parameters tested is sufficient to select new values of the behavior parameters. For example, figure 6.9 suggests that lowering the value of *Dependency Bonus* from 1.0 to 0.4 should result in a large reduction of project duration.

A total of nine parameters were selected and modified according to the results of this experiment. The new parameter values were tested by running 100 repetitions of the simulation. The mean project duration calculated from this test was 177.9 hours with a standard deviation of 3.3 hours. The old project duration (calculated from 1000 repetitions) was 178.6 hours with

[3] It is possible to set up orthogonal array experiments that specifically estimate interaction effects. Given the number of behavior parameters and the number of plausible interactions, I chose to ignore interactions and just take small steps along the estimated gradient.

[4] Because we have run many trial repetitions, a better experimental method would be to randomly assign orthogonal array columns to the tested parameters for each repetition of the trial block (81 runs). This would help reduce unwanted second-order interactions between behavior factors.

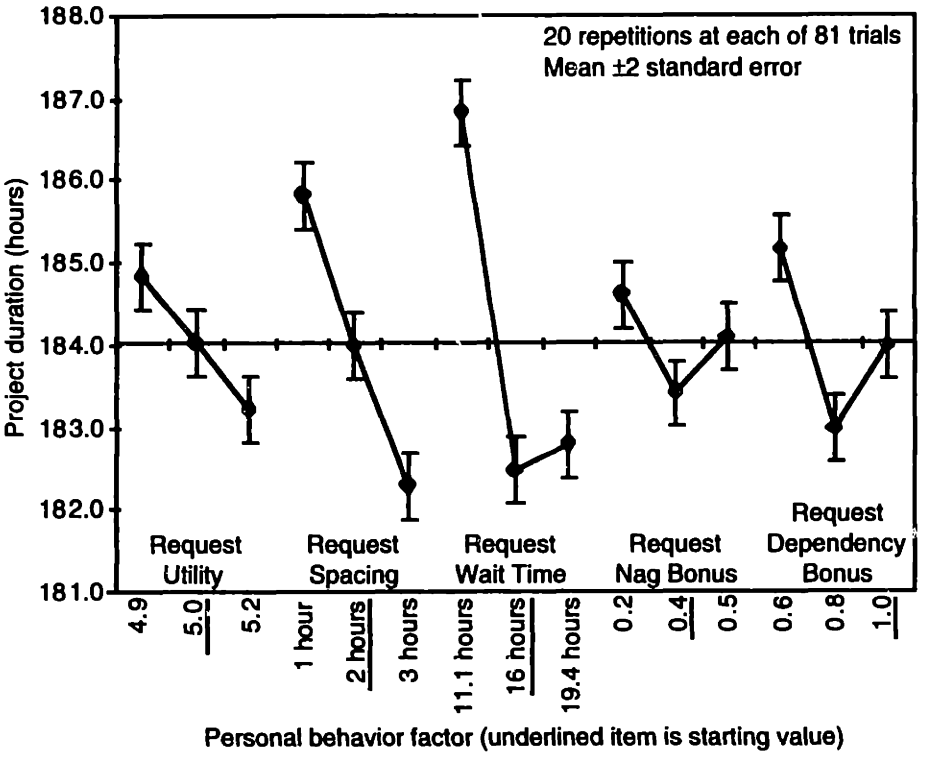


Figure 6.8: Change in project duration due to variations in personal behavior parameters that control requesting information.

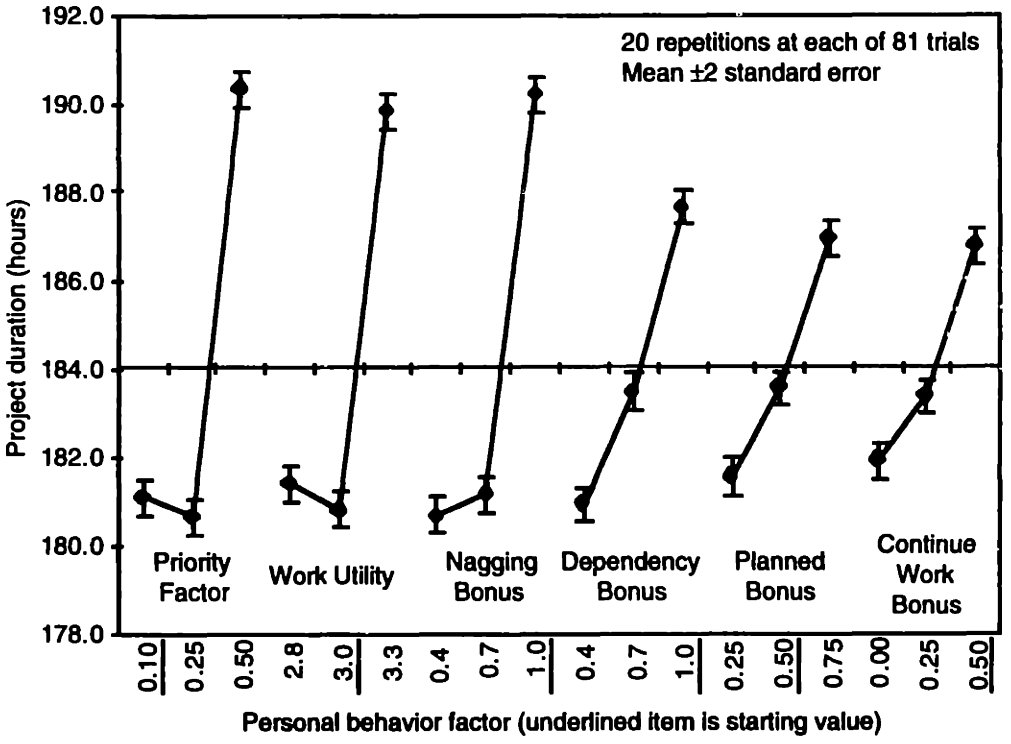


Figure 6.9: Change in project duration due to variations in personal behavior parameters that control working on solo tasks.

a standard deviation of 3.3 hours. The new project duration is statistically lower than the old one ($p < 0.05$).

The improvement in project duration after modifying the agents behavior is not impressive, particularly as the individual parameters shown in the graphs suggest much larger performance improvements. This is the result of two factors. First, the behavior parameters are highly interdependent whereas the orthogonal array experiment ignores all interactions between parameters. Hence second-order interaction effects will mitigate improvements. Second, the values for the behavior parameters were originally selected by running a long series of optimization experiments. Periodically during the course of the software development I would spend a day or two running these behavioral experiments and attempting to improve agent behavior parameters. During those runs I often improved performance significantly. Hence the behavior parameters used in this sample experiment are already fairly good; there aren't large gains in performance left to be made using the current behavior model.

To conclude this section I should point out that optimizing behavior parameters for a single project (that is, set of tasks, dependencies, agents, and communication methods) does not imply that the behavior parameters are optimal for all types of projects. In fact, experimentally one could attempt to find optimal behavior sets for different types of projects and compare the *types* of behavior that are optimal for different types of projects.

6.4 Summary

This chapter has discussed how to run experiments with the *DiFS* simulation, the robustness of the simulation to variations in personal behavior, and the effects of variation in task and behavior parameters.

The standard approach of running experiments in *DiFS* is statistical in nature. A project is created with tasks, agents, dependencies, and communication equipment. Using an AppleScript, the project can be run and re-run with systematic variations in project parameters. Several metrics can be stored for analysis: the overall duration of the project, the total number of person-hours logged, or the billable hours logged. An experimental trial is normally stopped when the individual responsible for the top-level management task is aware that every task in the project has completed.

Section 6.2 discussed the effect of adding stochastic variation to the agent's calculations of the utility of possible objectives. The effect of adding a small level of stochastic variation is to randomize slightly the objectives that agents pick; instead of a deterministic simulation, the variation in objective selection results in completely different sequences of actions. However, these different sequences of actions appear to be well-behaved in the sense that the standard deviation of the project duration is generally no larger than five or six hours.

The final section of this chapter discussed how parametric variations in project parameters may be tested. Small variations in task work time resulted in small, consistent variations in project duration. Variations in personal behavior parameters were used to optimize the agent's personal behavior and to reduce the overall project duration.

The next chapter discusses another aspect of model validity: is the *DiFS* simulation sufficient to represent engineering projects? That is, can a real engineering project be modeled in the *DiFS* simulation?

Chapter 7

Validation Experiment

This chapter describes the activities performed in a corporate validation study of the *DiFS* simulation. The intent of this study is to demonstrate that the *DiFS* simulation is an effective method of modeling engineering projects and can be used to make accurate predictions.

This document is divided into five sections: an introduction to the problem and the company, the data analysis performed on the corporate data, how the *DiFS* model was created from the corporate data, a comparison between the simulation and corporate data, and a discussion of experiments performed.

7.1 Introduction

This section discusses what I attempted to accomplish with the validation study and describes the project chosen for the study.

7.1.1 Purpose

The information-flow model and *DiFS* simulation were created based on experience gathered by visiting companies, talking with engineers/managers, and reviewing literature. There are three verification issues that need to be addressed: sufficiency, plausibility, and predictive power.

A **sufficient** model is one where all the important parts of a design project can be captured. That is, can we create an accurate model of a design project given the intent and limitations of the modeling method? If important factors cannot be included, then the model needs to be extended. The hypothesis of the information-flow model is that the interrelated nature of the design process may be represented by the required information transfer between the tasks that make up an engineering project. The sufficiency question can be addressed by attempting to model engineering projects in *DiFS*.

Plausibility refers to whether or not the model behaves in a "reasonable" fashion. Running the simulation should give consistent and sensible results. It would not be plausible to have a simulation where computer agents never do any work. It would be plausible if they don't work because the experimenter has specified that individual agents place the highest priority on wasting time. The plausibility question can be addressed by running the *DiFS* simulation on many input cases with different types of behaviors and checking to see that agents behave rationally and effectively. Much of the plausibility question has been addressed during the course of creating the simulation model. A standard debugging technique was to run the simulation on a variety of test cases. When pathological behavior was located, its source was tracked down and fixed.

Predictive power is the final verification of the model. No matter how accurate or entertaining a model, if it can't make useful predictions, then it has limited appeal and utility. From a verification standpoint, the predictive capability of a model is hard to validate because

it is difficult to set up a controlled experiment with engineers working on similar design projects under slightly different circumstances and requirements. Hence validating the predictive power of a engineering process model is normally limited to demonstrating that perturbations of the model give plausible results.

To test these three issues I have worked with a local company to model one of their design projects. This validation study primarily addresses the issues of sufficiency and plausibility, with a few predictive experiments described at the end. The creation of the model of the company's design project itself addresses sufficiency. Running the model to see if it behaves in a similar fashion to what really happened addresses plausibility. Predictive power is examined via sensitivity studies.

7.1.2 Project Description

For the purposes of a validation study, a design firm local to the Boston area kindly agreed to provide access to engineers and project documentation. This firm keeps accurate timesheets on the activities of their employees, as well as excellent project documentation. A single design project was selected for the validation study, based on the criteria of a "reasonable" size (i.e., 3–4 full time people working over 4+ months). I deliberately selected a project that had been completed in the past year.

Details of the actual project are under non-disclosure, so I cannot describe everything. All members of the design team will be referred to by their roles. The company will be referred to as "the company"; the client company (who paid for the design) will be referred to as "the client". Individuals at the various companies who were not directly related to the project will be referred to by their approximate roles at the companies.

The purpose of this particular design project was to design a line of water filtration units. The client had developed a new type of filter that they needed an attractive, inexpensive housing for. The contract called for the company to create two designs: an inexpensive version and a higher-cost version that was larger and had extra features such as a flow indicator. Interesting mechanical engineering issues that arose during the course of the design are (1) designing the path water would take through the casing and filter, (2) providing a flow controller that automatically compensates for varying head pressures, and (3) creating an attractive flow indicator. Styling was considered of high importance because the filtration units would be sold to home owners.

The project lasted approximately 6 months. Roughly speaking, it went through the following phases: a preliminary design phase of concept generation and brainstorming, an embodiment phase, a detailed design phase, and finally a post-detailed design phase with interactions with the tooling vendors. During this time, the company created a number of demonstration models including industrial design mockups, an alpha prototype to test flow characteristics, and a beta prototype. A fair amount of experimentation was necessary to create the flow regulator and flow indicator; this experimentation also went through a similar set of phases.

The company started with a detailed estimate of activities to be performed. That time estimate that ultimately proved to be accurate. Approximately 25 employees of the company participated at one time or another on the project. Only four of the participants could be considered to be "assigned" to the project: a program manager who dealt with billing issues and the client, a team leader who was responsible for the engineering aspects, a staff engineer, and an industrial designer. Other individuals participated in the discussions, helped out with models and experimentation, did some CAD work, consulted on managerial issues, and so forth.

7.2 Data Analysis

I was given full access to the information generated about this particular design project. This section discusses the available information and how it was analyzed. The net result of the

data analysis was a division of the project into tasks, or chunks of work to be accomplished by individuals or by groups. For purposes of confidentiality, the details of each chunk are limited.

The steps followed in the data analysis were (1) gather information, (2) construct a timeline of significant events, (3) code the timesheet data by the type of work being done and phase it was done in, and (4) recode the timesheet data into chunks that are similar to *DiFS* tasks. This section contains a description of each of these four steps. Section 7.3 contains a description of the construction of the *DiFS* model based on the coded data.

7.2.1 Sources of Information

I had access to three sources of information about this project: project notebooks, timesheets, and the team members themselves. I started by meeting with the project manager and then met with the engineer and industrial designer (for approximately 2 hours each). They told me about the project, showed sample parts/drawings, and discussed what had happened during the course of the design.

After interviewing the team members, I acquired the project notebooks and timesheets. The project notebooks consist of three or four large three-ring binders with detailed notes, design sketches, and memorandum from the project (as well as final drawings, etc). The timesheets consist of 1448 entries recorded by individuals working on the project. Each timesheet entry consists of 4 elements: the date, the name of the person, a short text description of the activities performed, and the number of hours spent. The level of detail recorded in each timesheet entry varies from person to person and from day to day. On some days an individual might record as many as 10 different activities performed; on others he/she might just say "Worked on Drawings: 8 hours".

7.2.2 Project Timeline

The first step in the data analysis was to read all of the project notebooks and create a working timeline of the activities that occurred during the course of the design. Meetings with the client were noteworthy, as well as dates that drawings were sent to vendors or quotes were accepted. Fortunately, after each important meeting with the client, a memo would be written and the important points discussed summarized. These dates were cross-referenced with timesheet entries. Approximately 13 important milestones were identified, ranging from the initial project kickoff meeting to the final review meeting with the tooling vendors. The principle project milestones are as follows:

- Preliminary kickoff meetings with team members and client engineers.
- Brainstorming sessions held at the company and at the client company.
- First Design Review. A series of preliminary concepts were presented with the intent of identifying the most promising forms to follow up on. Concepts were presented as foam models and layouts.
- Second Design Review. Two detailed concepts presented to the client. The intent of this meeting was to agree to the final shape of the design; however, consensus was not reached at this stage.
- Sales Meeting. A discussion and presentation of detailed concepts to the client's sales force.
- Third Design Review. Last minute discussions with client company to get everyone to agree to the desired design (also referred to as the "Emergency Meeting", as explained below).

- Alpha Quote Package. Drawings for the alpha prototype delivered to vendors for quotations and fabrication. The principle purpose of the alpha prototype to test flow regulation and flow indication ideas.
- Preproduction Quote Package. Drawings for the preproduction prototype delivered to vendors for quotations and fabrication.
- Alpha Parts Received. This set off a round of experimentation on flow regulators and indicators.
- Tooling Quote Package. Drawings sent to potential vendors and discussions regarding modifications that could improve the manufacturability, reduce price, etc.
- Preproduction Parts Received. Several preproduction models were manufactured; these were used for experimentation by both the company and by the client company.
- Internal Design Review. A company-wide discussion of the final drawings.
- Tooling Vendor. The final tooling vendor was selected and a review of the drawings conducted.

This project timeline was used to categorize the phases of the design process and to identify when certain dependencies would have to be satisfied.

One important event occurred approximately halfway through the project. The company believed they had finished the preliminary design work, and had begun work on detailed drawings. In fact, not all of the individuals at the client company had agreed to the final design. When this became evident, more preliminary design work was necessary in order to get agreement from all members of the client company. This series of miscommunications resulted in the loss of approximately 60 person-hours of detailed design work, changes to the project schedule, and a number of emergency phone calls and discussions. The meeting held to discuss this problem is called the "Emergency Meeting" because it was held on short notice.

7.2.3 First Coding

The first coding step was to categorize timesheet data by person, type of activity, and phase of the design. The purpose of this categorization was to prepare the raw timesheet data for the creation of a *DiFS* model.

As mentioned before, a total of 25 individuals participated in this design project at one stage or another. Six individuals had significant roles on the project; the remainder played minor roles on the project. Individuals with minor project roles are lumped into two pseudo-people. Table 7.1 is the list of the modeled individuals. The two pseudo-individuals are DR and GM. Individuals who helped with the experimentation, drafting, or engineering are lumped together into DR. Individuals who participated in managerial activities are collected into GM. These two pseudo-people account for approximately 5% of the total work/discussion time recorded. As it worked out, GM participation was so low that I eventually combined GM work with the program manager (PM).

Note that none of the individuals at the client company are represented, nor are any individuals at the various vendors for alpha, preproduction, or tooling parts. No data were available for these individuals, so they have not been explicitly modeled. The ramification of this decision is that all interactions between individuals at the company and non-modeled individuals are represented by work tasks.

Table 7.2 is a sample subset of the database. This sample has been modified in two ways: by date and by person. The original database contains the actual day of the year recorded. In this sample, all dates have been replaced by an ordinal scale that does not preserve the spacing between dates. The names of the individuals have been replaced by the appropriate code (as given by table 7.1).

Table 7.1: Reduced set of individuals.

Initials	Description
PM	Program & Client Manager
TL	Team Leader
EN	Engineer (+some drafting)
ID	Industrial Designer
DR	Draftsman
T1	Technician #1
T2	Technician #2
GM	General Management

For the first level of categorization each timesheet entry was assigned values in each of four categories: type, class, subclass, and phase.

Type: The type of a timesheet entry was either Work (w) or Discussion (d), depending on the description of the entry. Work entries are reserved for solo work (i.e., only one individual participating). Discussion occurs whenever groups of people are working together. Note that as client personnel are not modeled, entries such as the first item (a phone call to the client) are categorized as work. Some entries contained references to multiple activities performed. In these cases, if one activity clearly took the greater portion of time, its type was selected. Otherwise, the first activity mentioned was used.

Class: The class of a timesheet entry refers to the general category of work being performed. The following class categories were used: client, design, engineering, experiment, fabrication, milestone, management, and model. The *milestone* class is special; it refers to particular milestone meetings that occurred during the course of the project, as well as the work directly associated with those milestones (traveling, writing up summaries of the discussions, etc.

Subclass: The subclass of a timesheet entry categorizes the class of the timesheet entry. Each class has its own list of valid subclasses. For example, *client* has eight elements: administrative, billing, design discussions, experiment discussions, fabrication, general work agreement, specifications, and status updates. *Design* has 18 elements that break down mostly upon component lines and checking of drawings. *Engineering* has eight elements: finite element models, fluid dynamics, focus group videotapes, materials research, patent searches, standards review, tooling discussions, and welding discussions.

Phase: The phase of a timesheet entry refers to when in the design process it occurred. As with subclass, the phase of an entry is a function of the class of the entry. *Design* and *model* were divided into: concept 1/2/3, alpha 1/2, preproduction, tooling, and revision. *Experiment* followed a different path (based on what prototypes were available), namely preliminary, setup, alpha, and beta. Everything else was divided by concept, alpha, preproduction, tooling, and revision.

After coding, the database contains 310 unique combinations of type, class, subclass, and phase. Including people, a total of 497 unique combinations are represented (out of a total of 1416 timesheet entries).

7.2.4 Second Coding

The preliminary coding of the data categorizes what happened, when it happened, and what type of work was going on. These data are too detailed to directly translate into a model. The preliminary data was recoded into a form that could be directly translated into a project model.

Table 7.2: Sample timesheet entries.

Day	Name	Description	Hours
1	PM	call about CLIENT gwa†	0.3
2	PM	calls with CLIENT on authorization & deposit, internal kickoff with core team	2.0
2	PM	review CLIENT gwa, modify, review with CEO calls with CLIENT Pres. prepare for kickoff at CLIENT	2.0
2	TL	mtg w/PM and ID re CLIENT kickoff tomorrow	1.0
2	TL	reviewing CLIENT materials: proposal, design specs, background	2.8
2	ID	CLIENT head tech — brief w/PM and prep for meeting	4.0
3	PM	travel to and from CLIENT	2.5
3	PM	prepare for and attend kickoff meeting & lunch with CLIENT	5.5
3	TL	kickoff mtg at CLIENT, incl 2.5 travel	8.5
3	ID	kick off mtg, rev current products and inv. molding / mat'ls	8.0
4	PM	call with CLIENT on next meeting, debrief and review planning tasks with PM, brief ID	0.8
4	TL	compile and review notes from CLIENT, compile minutes from mtg yesterday	4.0
4	TL	minutes memo	2.0
5	PM	revise CLIENT gwa, pass to CEO for comment, call to CLIENT	2.0
5	TL	complete memo re: meeting minutes	1.5
5	TL	revise CLIENT development plan and minutes memo per PM	3.8
5	TL	mtgs w/PM re development plan	1.0
5	TL	compile notebook w/ref materials from CLIENT	1.0
6	PM	revise and fax new gwa following review with CEO	1.0
6	TL	revise development plan, compile hours for staffing, fight with excel	3.5
7	TL	revise development plan to fit prototype schedule and for pts sheets	5.0
8	PM	see TL several CLIENT items	0.7
8	TL	continued work on development plan and discs w/PM	2.5
8	TL	arrange COMPANY b'storm session	1.0
8	TL	combine CLIENT and COMPANY product specs	0.5
8	TL	disc w/CLIENT PRES. re: product spec	0.4
9	PM	discuss scheduling, CLIENT brainstorming session, planning with TL	0.5
9	GM	brainstorming	2.0
9	DR	brainstorm	2.3
9	TL	B-storm general concepts, + prep + cleanup	3.0
9	TL	compile design spec from COMPANY and CLIENT specs	1.5
9	TL	prepare b-storm agenda	1.0
9	TL	review purifier units from CLIENT	1.0
9	ID	B-storming session TL	2.5
9	T1	System level brainstorm session	2.0
9	DR	Brainstorming session	2.3
9	EN	Brainstorm session on purifier	2.0

† "gwa" = General Work Agreement.

Dependencies between the project model were determined by examining the original coded data as well as the secondary coded data.

The preliminary coding of the data suggested a form for the underlying model. Each of the classes of activities that were engaged in during the course of the design generally divided themselves logically along "phase" lines. More importantly, the reason these activities divided themselves by phase was because of the milestones that occurred during the course of the project. The individual subclasses within each class of work were not as important as the person who did the work and in what phase it occurred.

The second coding of the data created a consistent set of phases across all classes, ignored the subclasses, and combined a few classes together (specifically, engineering was merged with fabrication and client/management were combined). In addition, a number of timesheet entries that had previously been categorized as "discussion" were given the type "joint work" to indicate that several people had spent time working as a team on a particular problem.

Using the second coding of the data, table 7.3 is a summary of the hours logged as work. Table 7.4 summarizes hours logged as joint work, and table 7.5 summarizes hours logged as discussion.

7.3 Model Creation

This section of the document contains a discussion of how the coded data was translated into a *DiFS* model, a description of the resulting model, and a discussion of how the creation of this model prompted several improvements to the *DiFS* software.

7.3.1 Steps

The coded data was translated into a *DiFS* model by a series of three general steps. First, an appropriate collection of work tasks, group decision tasks, and management tasks was created. The individual tasks were based directly off of the corporate data as summarized in tables 7.3 through 7.5. Hence task hours and personnel assignments were already known. Second, parameters were assigned to the tasks based on engineering judgment and available data. Third, dependencies between these tasks were created and checked for appropriateness.

The tasks within the *DiFS* model were created as follows:

1. The overall engineering project was divided into management tasks by phase, with only the Kickoff meeting and Emergency meeting placed outside of the management tasks. This grouping of tasks by phase has some basis in fact because most of the phases are terminated by a milestone meeting representing a step in the life of the project. However, the primary reason to divide the project into management tasks by phase was to visually separate the project into recognizable chunks.

2. Within each phase, instances of solo work by person/class (as in table 7.3) that contained more than a small number of hours were translated into a work task. A "small number of hours" was loosely defined to be around five, but was subject to the interpretation of the individual coding the data. Leftover hours were swept into larger tasks with a matching class.

3. Joint work reports (as in table 7.4) were translated into group decision tasks. This translation was done more loosely than the work-task translation because many of the items listed in the timesheet entries as "person X worked with person Y" do not have a matching "person Y worked with person X". The artistic license taken in this stage is partially justified by the hours spent discussing the project with the engineers and reviewing the project documentation.

Table 7.3: Work hours by person, phase, and class.

Phase	Class	PM	TL	EN	ID	DR	T1	T2	Total
Brainstorm	design		1.0						1.0
	M		13.7						13.7
	mgmt	7.3	27.6						34.9
Design	design		47.5	51.0	55.5	17.3			171.3
Review 1	experiment		1.0	4.0					5.0
	fab		1.0						1.0
	M	0.9	2.7	7.0					10.6
	mgmt	2.1	1.0						3.1
	model				3.5		18.9	23.9	46.3
Design	design	1.3	34.5	51.3					87.1
Review 2	experiment		8.9	30.5			46.9		86.3
	fab		6.8						6.8
	M	1.0	6.2						7.2
	mgmt	1.8	17.8						19.6
	model						11.9	37.5	49.4
Sales Mtg.	design		11.2	33.0	25.3	43.0			112.5
	experiment		3.1	13.8			13.0		29.9
	fab		10.5	2.0					12.5
	M		10.5						10.5
	mgmt		6.5						6.5
	model				11.0		1.2		12.2
Embody 1	design		33.0	60.5	33.0	34.0			160.5
	experiment			1.0		2.5			3.5
	fab		12.4	3.0					15.4
	M		3.0	1.5					4.5
	mgmt	2.0	6.8						8.8
Embody 2	design		17.6	60.5	38.0	1.5			117.6
	experiment		2.3	17.5			8.0		27.8
	fab		8.0						8.0
	M		8.8						8.8
	mgmt	1.4	8.4						9.8
Preprod 1	design		30.0	104.5	11.8				146.3
	experiment		21.1	24.8			24.4		70.2
	fab		21.1	6.0					27.1
	mgmt	1.4	36.2						37.6
	model		2.0						2.0
Preprod 2	design		10.7	16.3	8.0				35.0
	experiment		3.0	33.5			3.5		40.0
	fab		7.7	6.0					13.7
	M		4.0						4.0
	mgmt		13.9						13.9
	model		8.1					24.5	32.6
Tooling	design		25.4	80.0	5.0				110.4
	experiment		8.8	41.0				26.6	76.4
	fab		23.5	7.5					31.0
	M		1.5						1.5
	mgmt	0.6	26.6						27.2
	model		1.3					3.5	4.8
Revision	design		28.3	41.8					70.1
	experiment		5.8	3.0				13.1	21.9
	fab		10.0	4.8					14.8
	mgmt	4.3	13.1						17.4
	model		10.0					1.7	11.7
Total		22.8	590.5	688.8	242.3	95.8	130.2	130.8	1901.1

Table 7.4: Joint work hours by person, phase, and class.

Phase	Class	TL	EN	ID	DR	T1	T2	Total
Design Review 1	design	3.0		4.0				7.0
Design Review 2	design	1.5	1.0					2.5
	experiment	4.0	4.5			2.0		10.5
Sales Mtg.	design	10.7	6.0					16.7
	experiment				1.5			1.5
Embody 1	design	4.8						4.8
Embody 2	design	4.5		7.0				11.5
	experiment	1.0						1.0
Preprod 1	design	3.8	4.0					7.8
	experiment	1.7				0.4		2.1
Preprod 2	design	1.5	2.5		1.0			5.0
	experiment	2.0	2.0		2.5		1.7	8.2
Tooling	design	1.0	5.0					6.0
	experiment						2.3	2.3
Revision	design	1.0						1.0
Total		40.5	25.0	11.0	5.0	2.4	4.0	87.9

4. Milestone meetings (a form of group decision task) were created to match the appropriate milestones observed. Milestone meetings that were held at the client company were assigned a 6 hour duration; milestone meetings held at the design company were assigned a 4 hour duration. The longer duration at the client company reflects the "all-day" nature of most of the milestone meetings.

5. Communication time associated with each task was estimated to be between 16% and 33% of the task's work time. The percentage varied to reflect the type of activity being performed; for example, experimentation tended to have low percentages, whereas engineering activity tended to have high percentages. Assignments of communication time unfortunately cannot be justified by looking at the company database.

6. Other task parameters were set consistent with client documentation and available notebooks. Thoroughness of documentation was associated directly with activity type and available documentation. When in doubt, parameters were left at their default levels.

7. Dependencies between the tasks had to be estimated. Project documentation, the first level of data coding, and engineering judgment were used. Generally the presence of a dependency could be inferred from the available data. When in doubt, it was assumed that a dependency did not exist. Scope, thoroughness, and completion curve of the dependency were set as best as could be estimated.

8. After creating all of the dependencies, the simulation was verified to check for inappropriate completion curves or missing dependencies. Only a few problems were identified and corrected at this stage. They generally consisted of tasks in the later section of the project not being sufficiently serially dependent upon upstream tasks.

7.3.2 Model Description

Figure 7.1 is a screen capture of the office layout for the company. This office layout is based loosely on the actual configuration at the host company, but many intermediate offices and rooms have been deleted to conserve space. The shop area at the bottom of the figure actually represents a separate floor in the building.

Figures 7.2 through 7.6 show the completed *DiFS* model. Because of the size of the model (108 tasks and nearly 2000 hours of work), it had to be divided between five figures. They connect from left to right. The management tasks represent phases in the design, and the

Table 7.5: Discussion hours by person, phase, and class.

Phase	Class	PM	TL	EN	ID	DR	T1	T2	GM	Total
Brainstorm	M	14.8	20.0	2.0	21.5	4.6	2.0		2.0	66.9
	mgmt	2.2	1.7							3.9
Design	design		10.9	5.3	1.0	0.5			0.3	17.9
Review 1	experiment		2.0	1.5						3.5
	M	7.6	9.4	7.5	7.0					31.5
	mgmt	1.3	6.2	0.5	0.8	0.5				9.3
Design	design			3.0	2.5	1.0				6.5
Review 2	experiment		5.6	7.5						13.1
	fab		1.0							1.0
	M	4.0	7.6	4.0	8.0					23.6
	mgmt	2.4	6.9	1.0					1.0	11.3
	model							0.5		0.5
Sales Mtg.	design		8.8	6.5	1.0				1.5	17.8
	experiment		3.7	5.5		1.2				10.4
	M	8.5	7.1		8.0					23.6
	mgmt	1.2	4.9							6.1
	model		1.0		2.0					3.0
Embody 1	design		5.9	7.0	4.0	0.4				17.3
	M	2.0	5.8	3.5	4.0					15.3
	mgmt	2.4	2.4	3.5						8.3
Embody 2	design		7.4	11.0	3.0	0.3				21.7
	experiment		2.5	2.0						4.5
	M		1.8							1.8
	mgmt	3.1	5.9	4.0						13.0
Preprod 1	design		7.1	6.5	8.5			0.5		22.6
	experiment		3.7	5.5			0.2			9.4
	fab		2.2	1.0	0.3					3.5
	mgmt	1.3	6.7	1.0						9.0
Preprod 2	design		2.5	3.0	1.0				2.0	8.5
	experiment		1.6	5.0						6.6
	M	3.9	7.5		1.5					12.9
	mgmt	1.5	1.2	0.5						3.2
	model		1.2					3.1		4.3
Tooling	design		23.5	5.5	7.0				0.5	36.5
	experiment		2.5	5.5			0.8	1.2		10.0
	fab		0.8							0.8
	M	2.0	2.5			1.8			3.7	10.0
	mgmt	2.5	3.8	4.0						10.3
	model							6.7		6.7
Revision	design		4.3		1.0					5.3
	experiment							0.7		0.7
	M		6.0	4.5						10.5
	mgmt	12.6	10.2	2.3				0.2		25.3
	model		0.7							0.7
Total		73.3	216.5	119.5	82.0	10.3	3.0	12.9	11.0	528.4

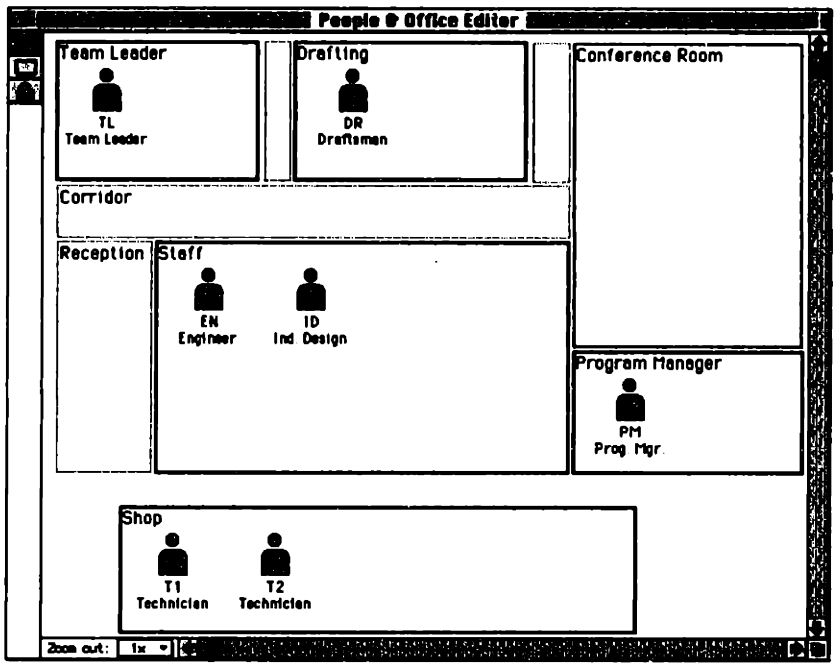


Figure 7.1: Office environment.

names of the individual tasks represent the type of work that was being done in them. In the case of work tasks, the name of the responsible person is indicated along with the hours of work required. Group decision tasks and milestone meetings do not indicate the responsible agents, but they generally have a predictable set of people involved (if a group decision task couples two work tasks, then those two people will be assigned to it). The team leader (TL) was responsible for almost all of the group decision tasks. The program manager (PM) was assigned responsibility for the entire design project (although it potentially would be appropriate to assign this responsibility to the team leader).

Communication channels between agents in the model were assumed to be standard office channels: face-to-face discussions, telephone calls, office mail, and answering machine messages. In fact, the close proximity of the agents working within the office building encourages primarily face-to-face discussions.

7.3.3 Required Software Modifications

One of the purposes of an experimental validation study is to point out limitations in the modeling approach and behavior algorithms. Two problems became apparent during the validation modeling effort. First, a new type of meeting needed to be created. Second, project start time estimation had to be improved.

The new type of meeting created is referred to as a milestone meeting. During the course of the design activity the client and the company would meet to discuss important issues. These meetings (or summits) always took place in a single day. The *DiFS* simulation originally had no provisions for representing an important gathering of individuals that occurs on a single day. Attempting to represent the meeting by a group decision task with tight dependencies generally would result in a meeting split over several days. Hence the milestone meeting was introduced. The only distinction between this type of meeting and the group decision task is that the individual responsible for the milestone meeting attempts to schedule it to occur in a single day.

The second problem was a human rationality one: the estimation of when a task will start. The long duration and multitude of tasks in the corporate model made it difficult for individuals

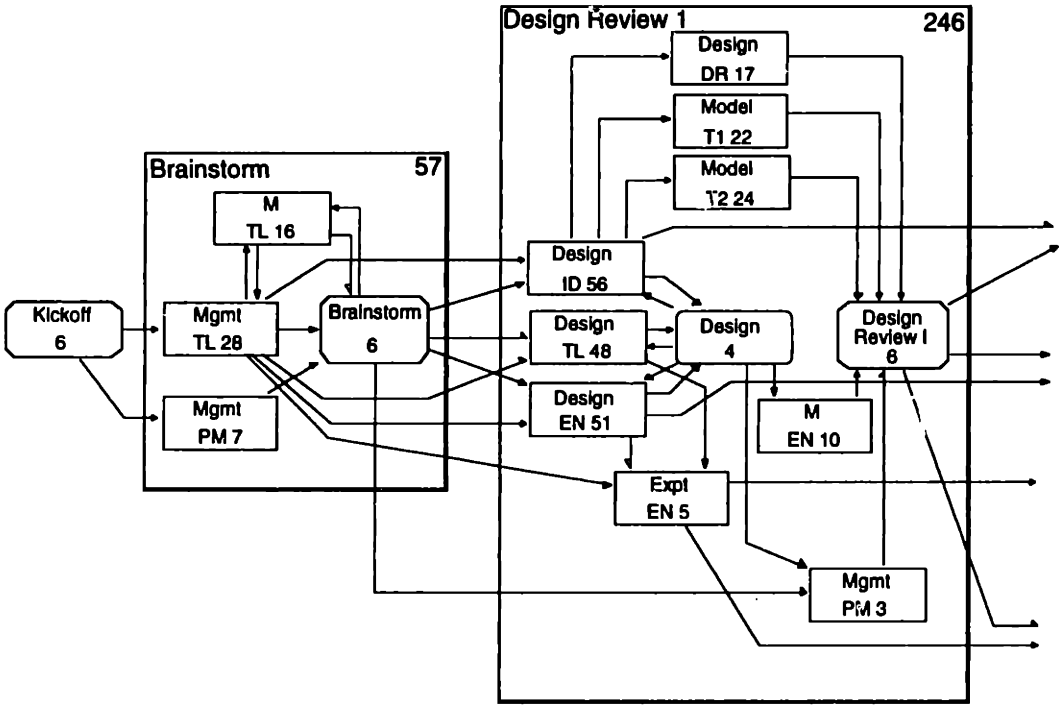


Figure 7.2: Filtration unit, part 1 of 5.

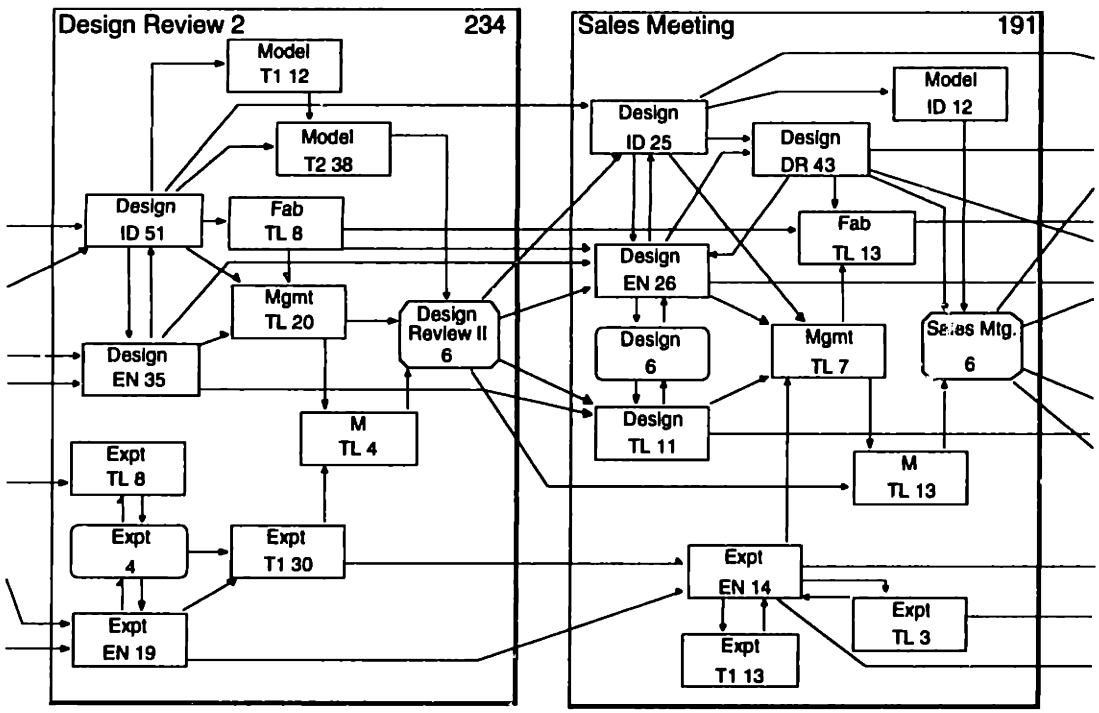


Figure 7.3: Filtration unit, part 2 of 5.

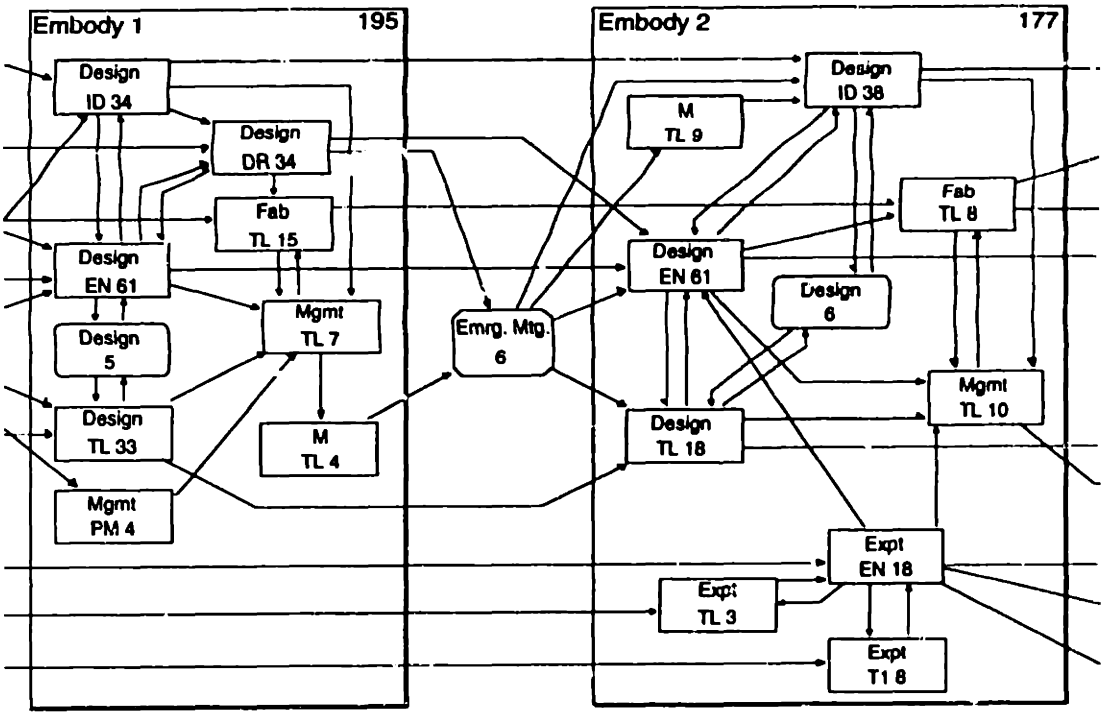


Figure 7.4: Filtration unit, part 3 of 5.

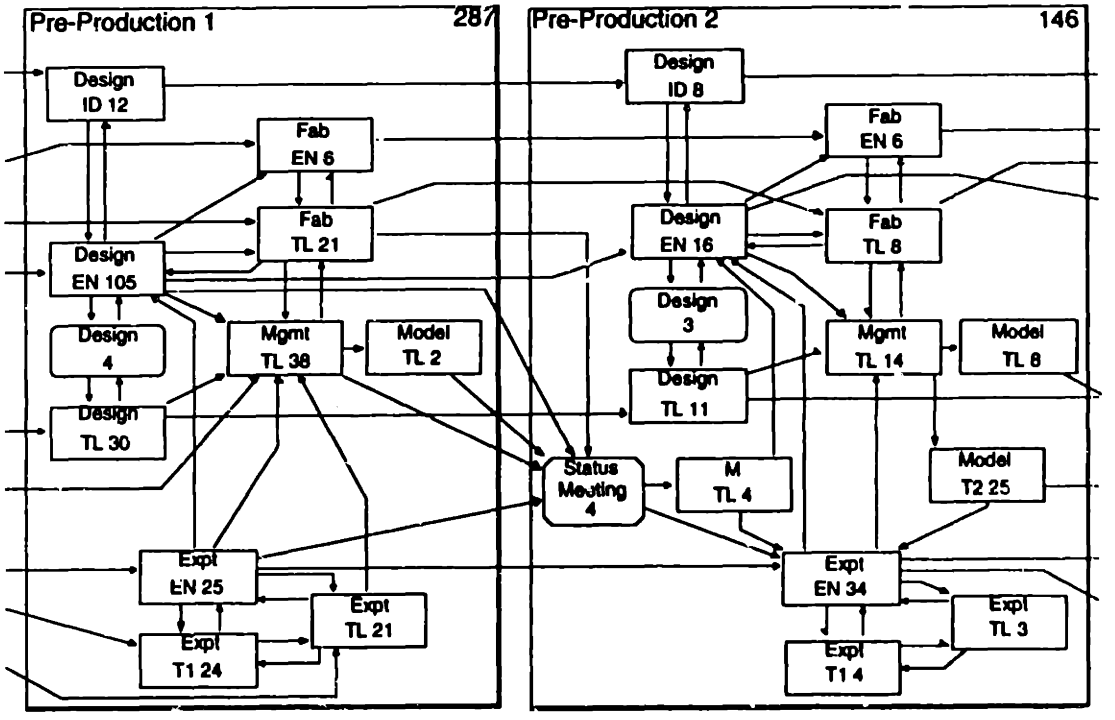


Figure 7.5: Filtration unit, part 4 of 5.

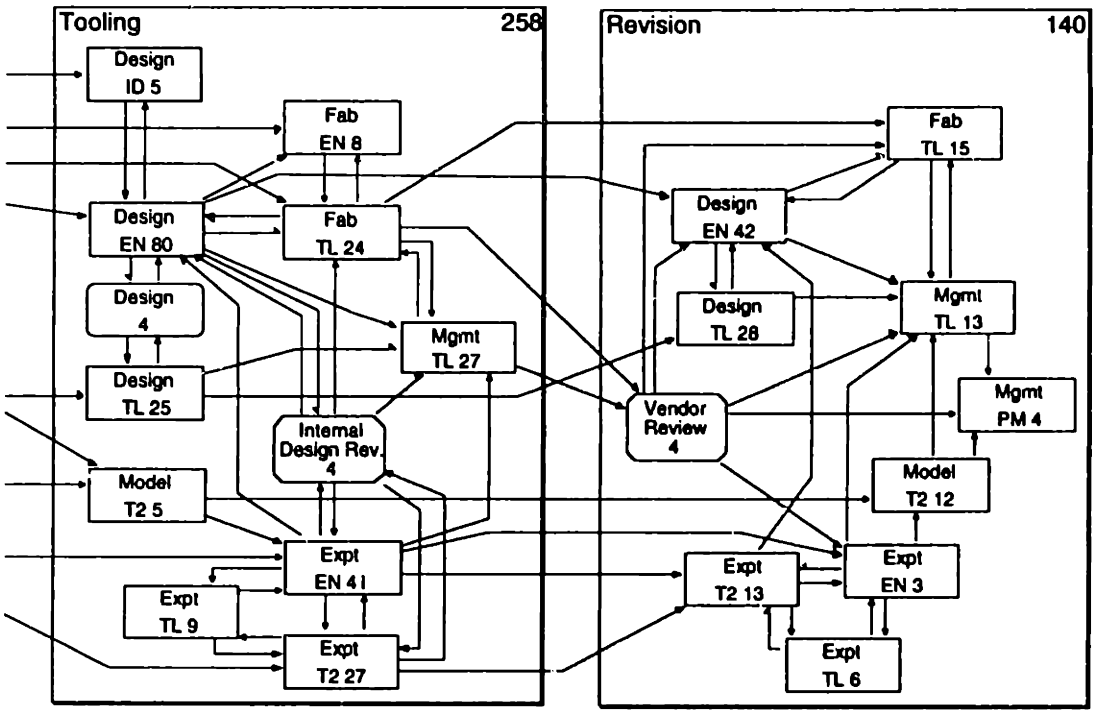


Figure 7.6: Filtration unit, part 5 of 5.

to accurately predict when a task would start and hence when information would be available. Originally individual agents based their predictions of task starting time by asking other agents when upstream tasks would start and translating that information into a start time for a downstream task. Because single agents have access only to the portions of the project model that directly concern them, they could not accurately propagate task start times through the entire model. Moreover, this long range propagation was not initially allowed as it requires a certain amount of agent intelligence when confronted with overconstrained project models. As a result, the first few weeks of work would be largely spent with agents passing start time information back and forth until accurate estimates of start times were arrived at.

The solution to the start-time problem was two-fold; first, before running the simulation an estimate of project start time is calculated based on all of the dependencies in the model and an expected working rate. This global estimate is made available to each agent in the simulation. As a side effect, the global estimate partially verifies that the project model is not overconstrained. Second, during the simulation run, whenever an agent acquires a new piece of information expected start times are propagated throughout all of his/her known dependencies. This extended propagation of information can be made efficient because the global model has already been checked for an overconstrained situation.

Although this change may seem unimportant, it actually has wide ramifications. The accuracy of the start time estimates affects the overall project duration. Too optimistic an estimate, and a great deal of time is wasted requesting information from tasks that haven't started. Too pessimistic an estimate, and a time is wasted by not acquiring information when it is first available.

There are a number of software modifications that have *not* been made that are warranted by the results from this study. A critical one would be to allow the experimenter to assign due dates for the work tasks. To properly implement this change, individuals would need a behavior that causes them to concentrate their efforts on finishing tasks so as to reach the targeted finish date (including propagating backwards through the dependency tree to decide what needs to be done early on). It would also be reasonable to give individuals the ability to stay late and

work extra hard on a task that is behind schedule. Unfortunately, allowing agents to work after 5:00 P.M. would require new project performance metrics because the project duration would no longer be indicative of project performance.

7.4 Model Results

This section compares the results of running the simulation with the corporate data. The creation of the *DiFS* simulation was based upon the recorded corporate timesheets, interviews, and engineering notebooks, *not* based upon the results of running intermediate version of the simulation. This was done deliberately; a modeling method that requires an iterative modeling approach is not useful for making predictions about a novel project.

A typical simulation run of the filtration unit project has a duration of approximately 136 days.[1] The corporate timesheets record a project duration of 146 days over a period of about 6 months (there are gaps in the day sequence at the beginning and end of the project).

7.4.1 Qualitative Results

There are a great number of possible ways to compare the data gathered at the company with the data generated by a single simulation run. This section presents qualitative comparisons of corporate and simulated project results by visually comparing project timelines.

The method of coding timesheet data and translating it into tasks results in tasks that can be categorized by three factors: the person assigned, the phase it occurs in, and the class of activity performed. To compare corporate data with simulated data, one of these factors can be compressed across the tasks and the other two plotted on a timeline that displays a progress trace of when work was done in the collapsed set of tasks. For example, collapsing across phase leads to 18 combinations of person and class.

Figures 7.7 and 7.8 compare the collapsed data by person and class. That is, all tasks that differ only by phase (but have the same person and class) have been collapsed into a single row of the figure. Figure 7.7 is the actual corporate data of person/class information created from the project timesheet database. A black rectangle on a single day represents at least four hours of work done on that particular task on the given day. Figure 7.8 shows the data generated by a single simulation run (because of stochastic variation, re-running the simulation will generate a slightly different figure). These figures are on the same scale so they can be superimposed to look for discrepancies.

A second type of comparison is to collapse across people and look at each class of work by the phase of the design. This is shown in figures 7.9 and 7.10. The third comparison type is to collapse across tasks and look at the time each individual spent during each phase of the engineering project. This is shown in figures 7.11 and 7.12.

Figure 7.13 is a time trace of the activities of each individual on a daily basis. Milestone meetings show up clearly as days with a great deal of communicating. This figure clearly points out that the large majority of the work done in the project was accomplished by just two individuals: the team leader and the engineer. Figure 7.14 shows the same data collected together across all agents and broken down by percentage of simulation time spent in each activity. This figure can be misleading because most simulation agents have limited responsibilities and hence are idle the majority of the time. Figures 7.15 and 7.16 show the time trace and pie chart of conversation topics during the simulation run.

7.4.2 Discussion

There are several discrepancies in these qualitative comparisons that deserve to be explained. Some of these discrepancies can be accounted for by the limited nature of the information

[1] A single simulation run on a PowerMac 6100/60 takes approximately 5 minutes and 45 seconds.

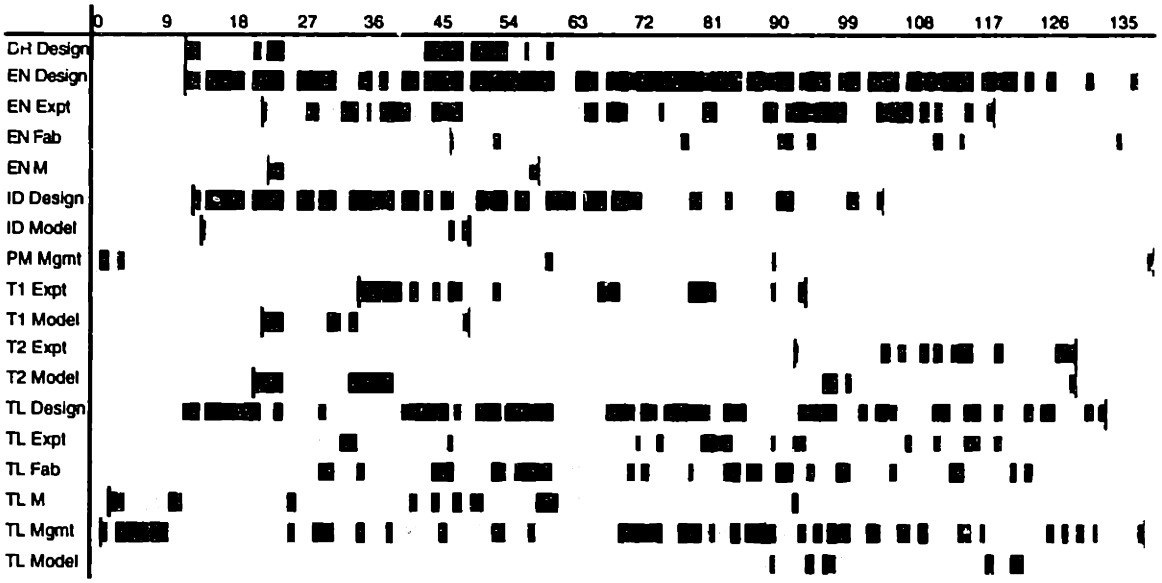


Figure 7.7: Actual progress trace (by person and class).

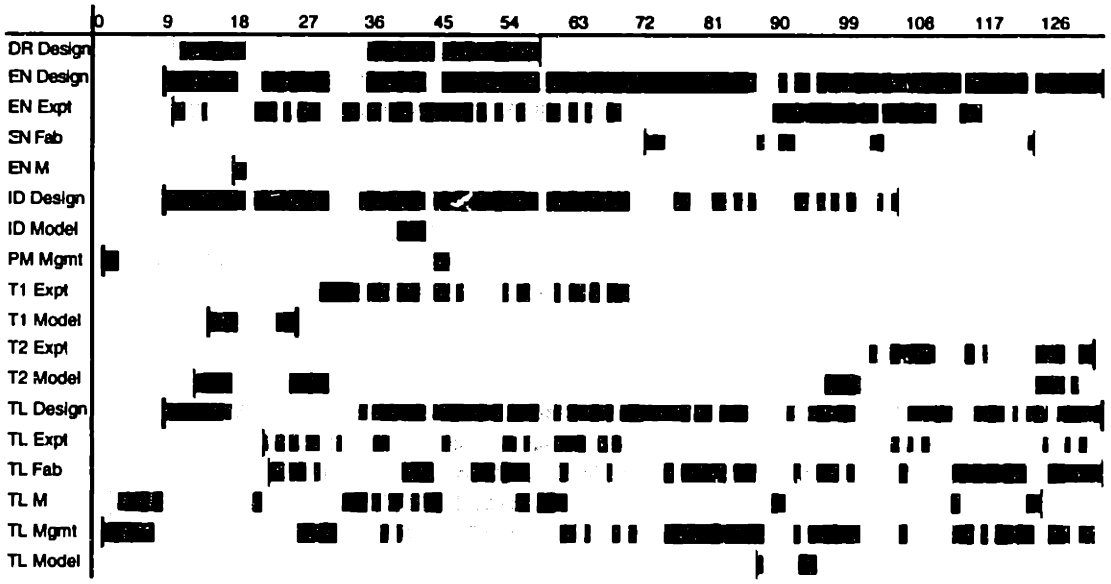


Figure 7.8: Simulated progress trace (by person and class).

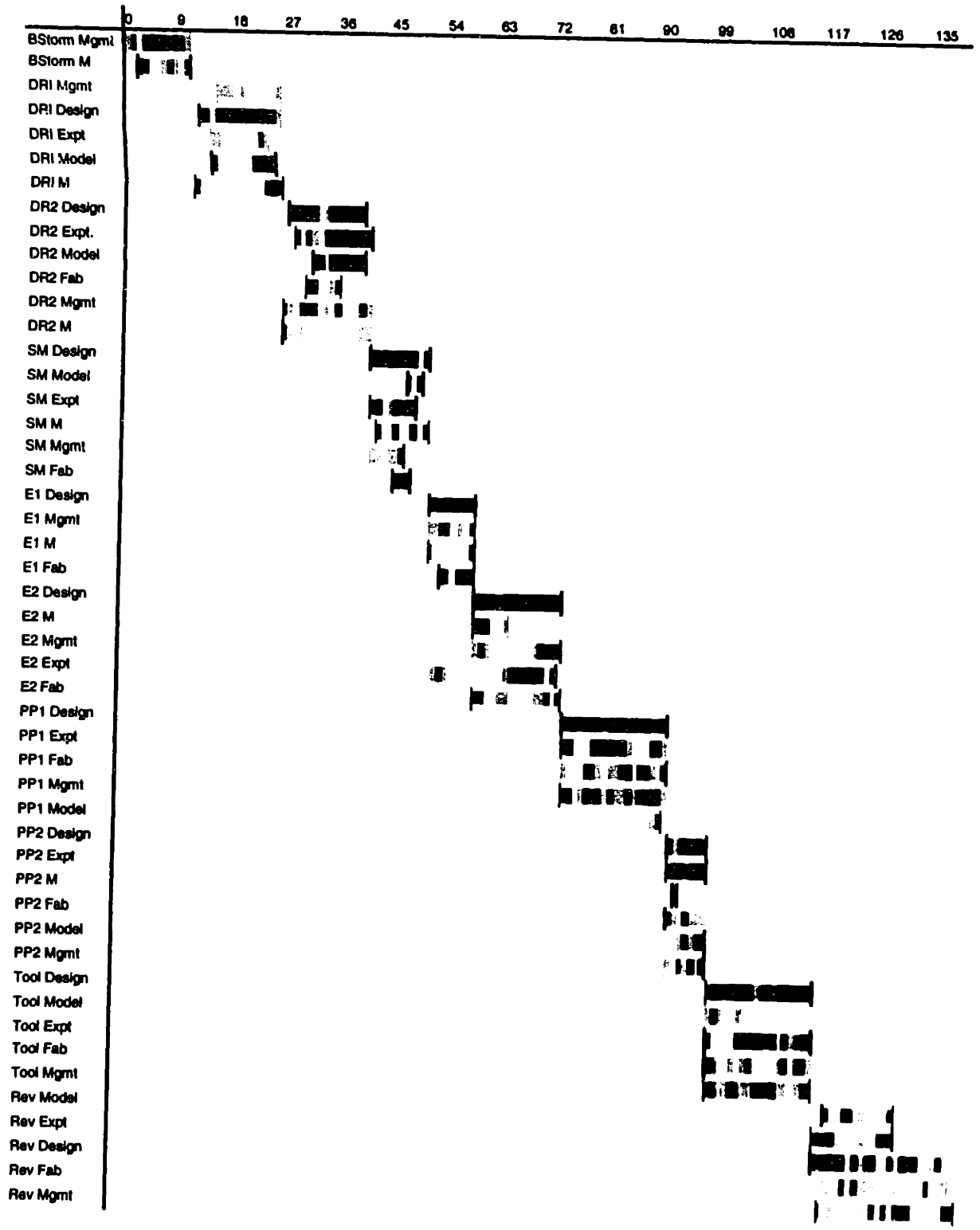


Figure 7.9: Actual progress trace (by class and phase).

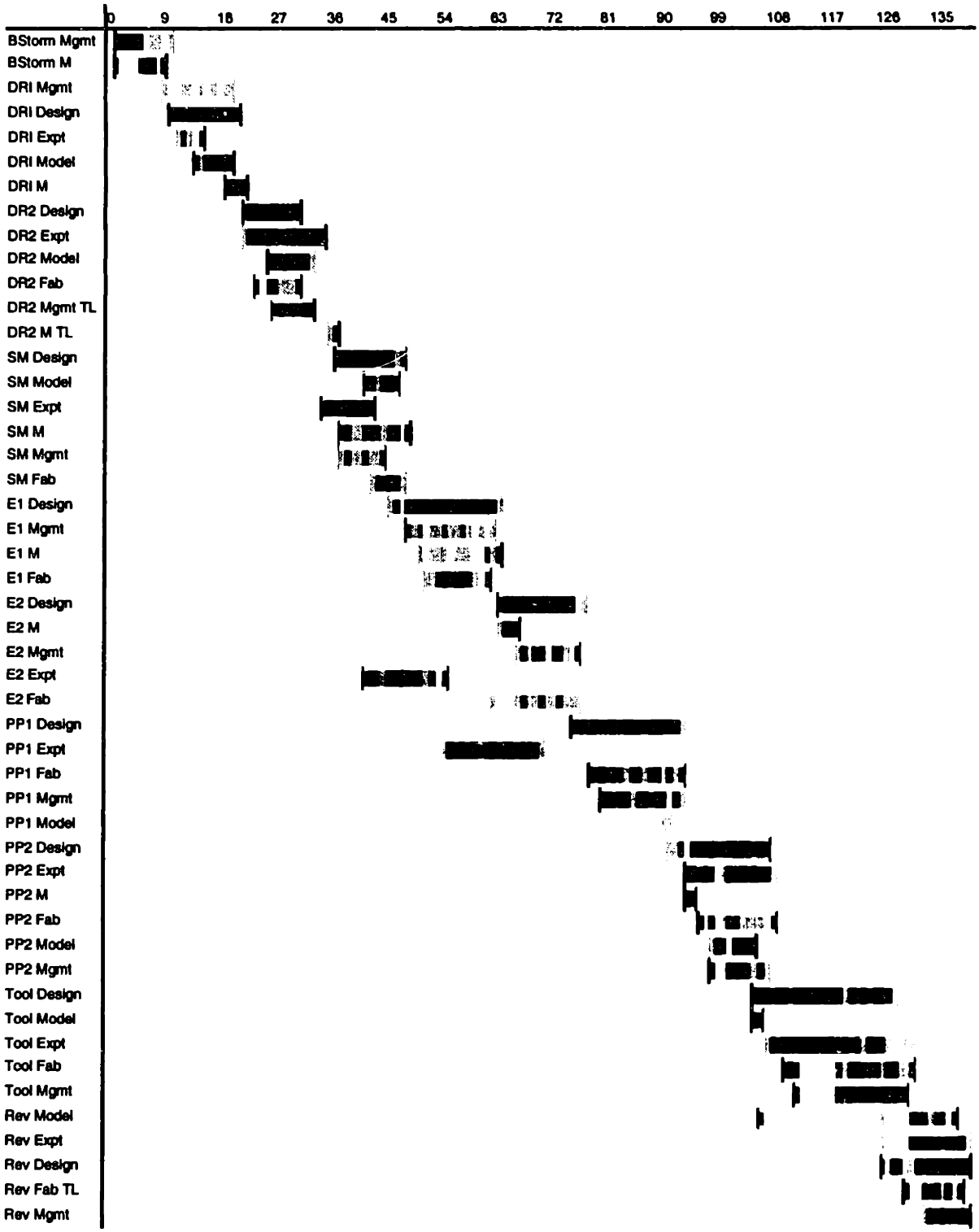


Figure 7.10: Simulated progress trace (by class and phase).

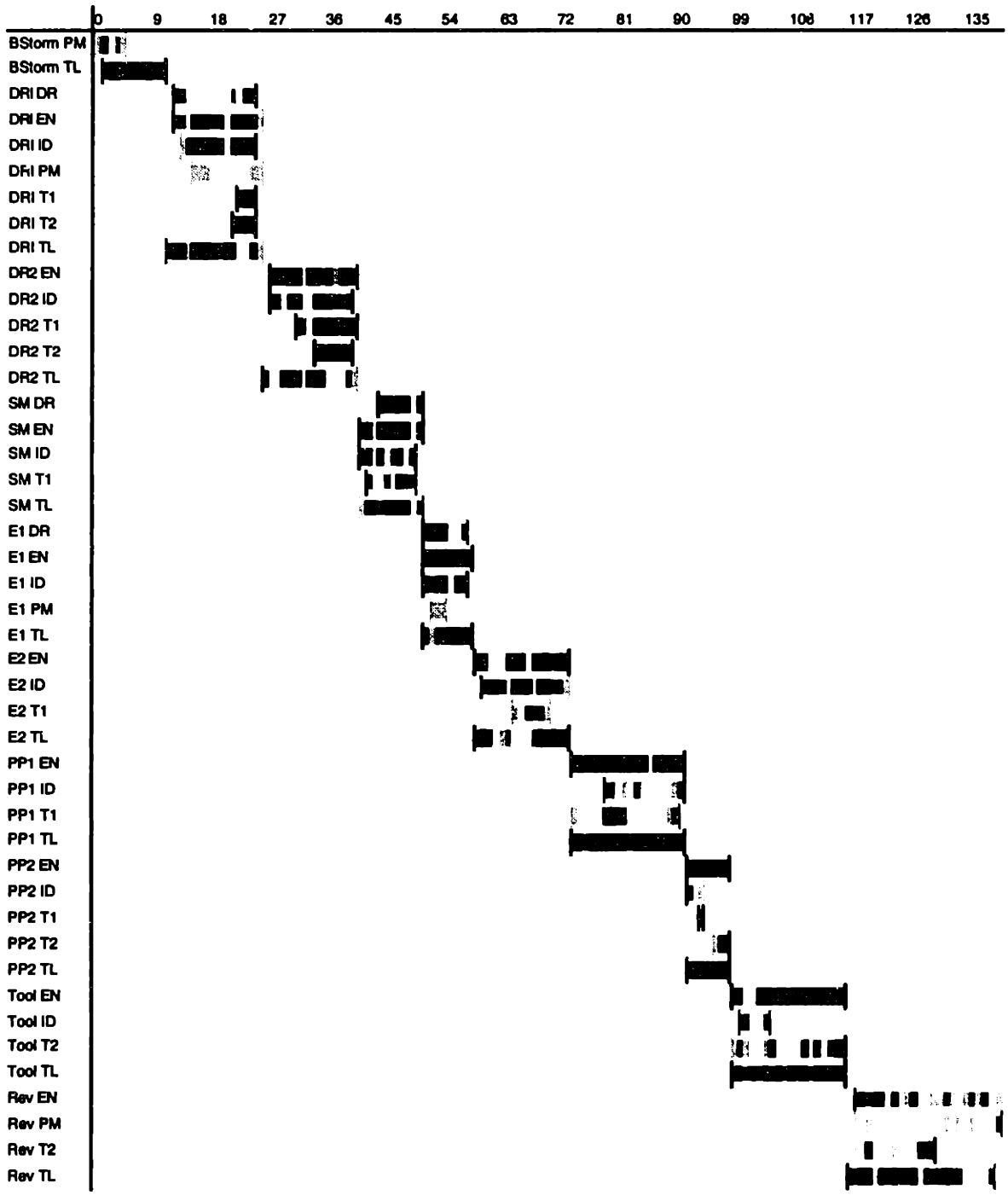


Figure 7.11: Actual progress trace (by phase and person).

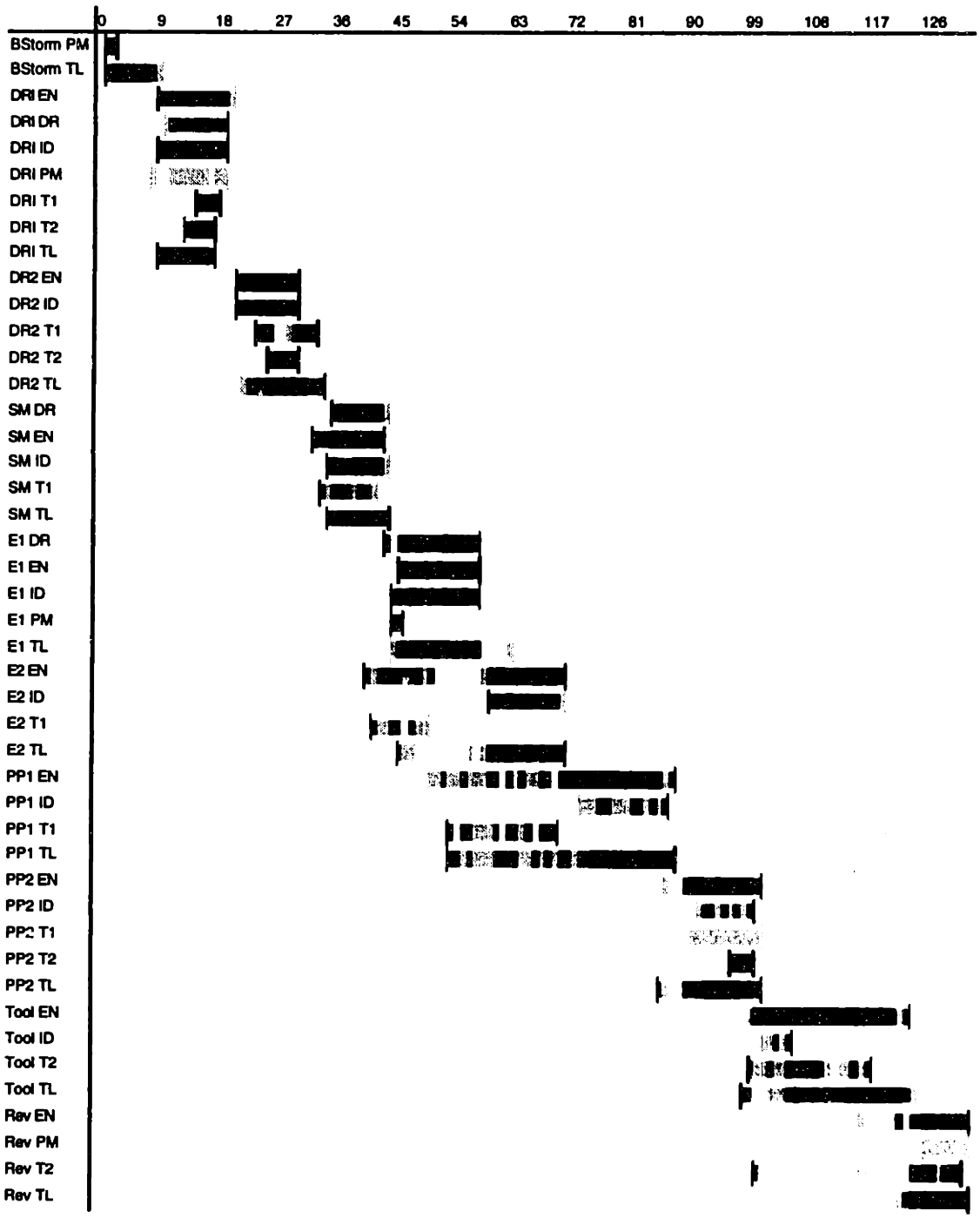


Figure 7.12: Simulated progress trace (by phase and person).

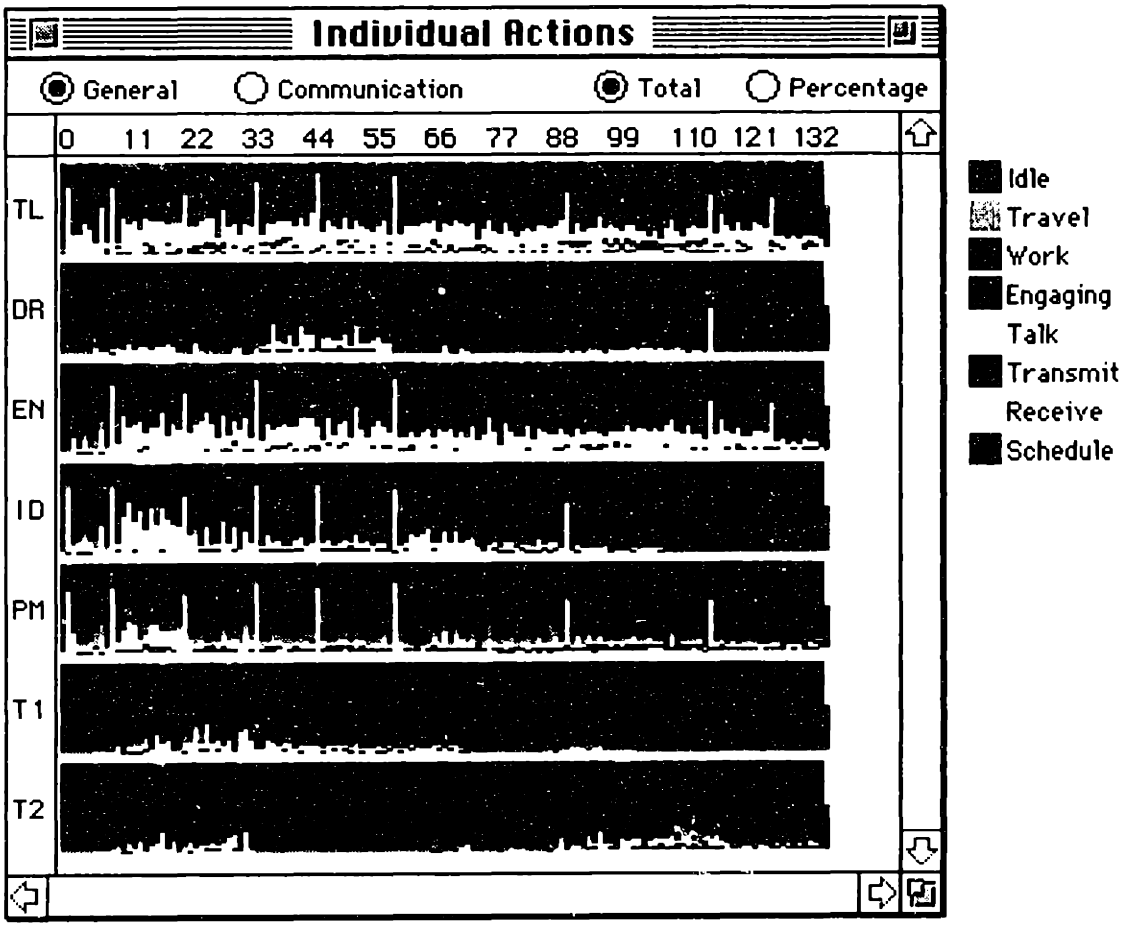


Figure 7.13: Time trace of activities by agent during the simulation. This figure was originally in color. The majority of time is spent either idle, at work, or talking.

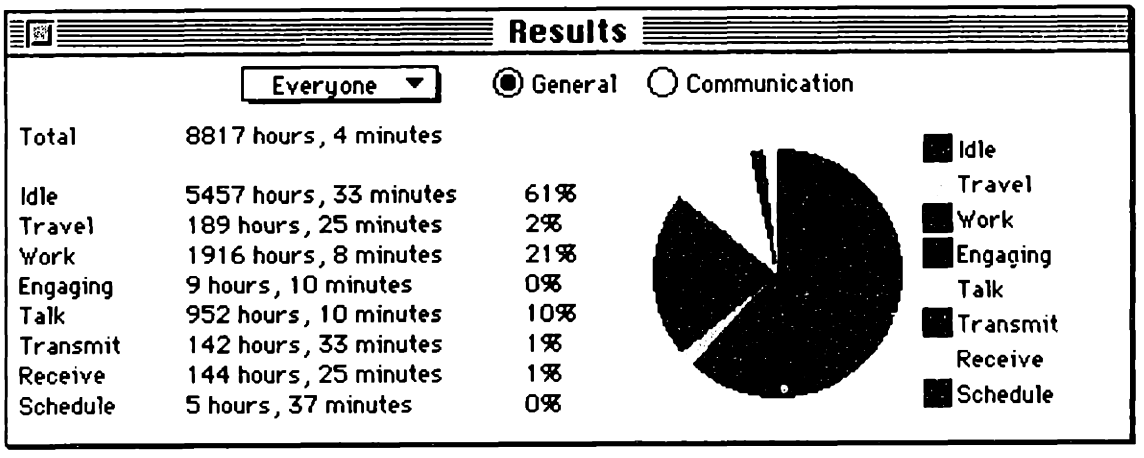


Figure 7.14: Percentage of time agents spent in the simulation doing different types of activities. This figure was originally in color.

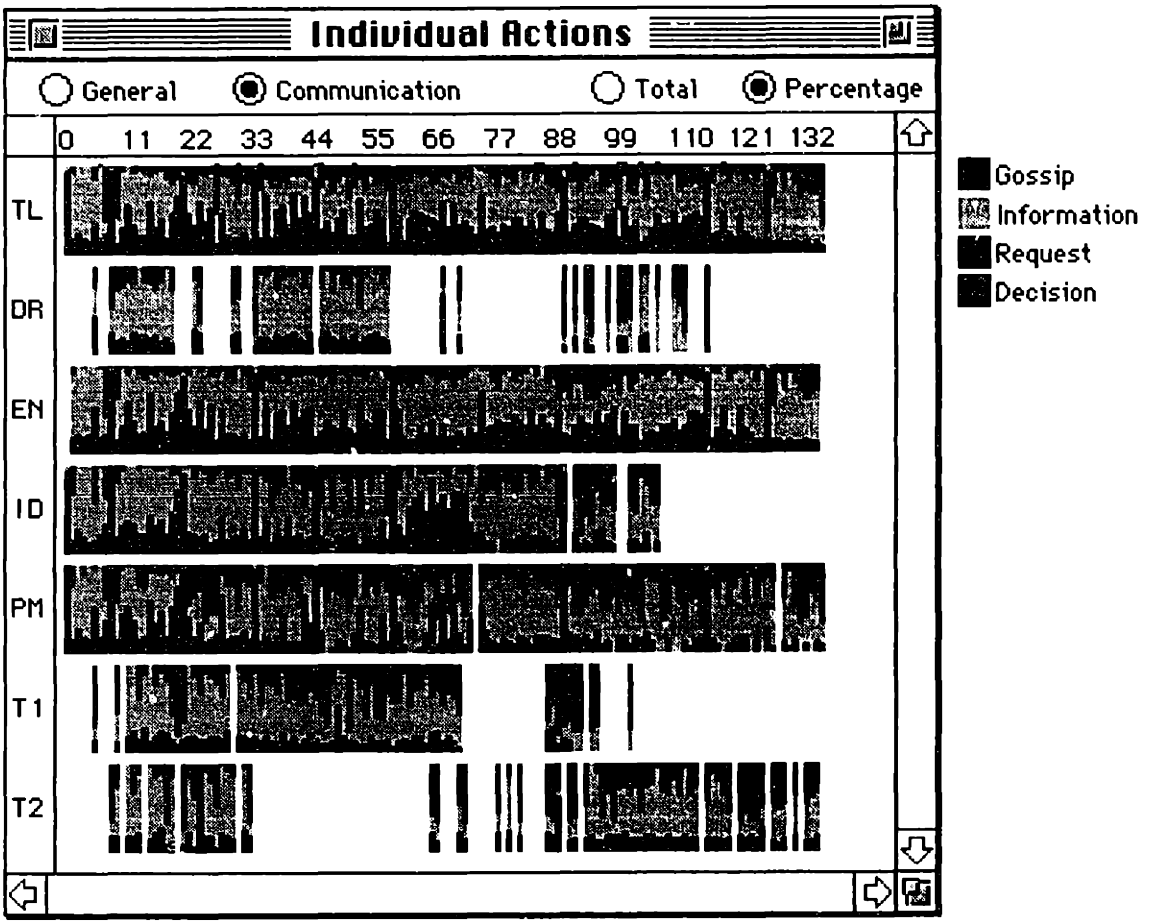


Figure 7.15: Time trace of conversation topics by agent during the simulation. The communication time spent on each topic by day has been normalized, so this figure represents the percentage of time spent discussing a particular topic by an agent on a given day. This figure was originally in color.

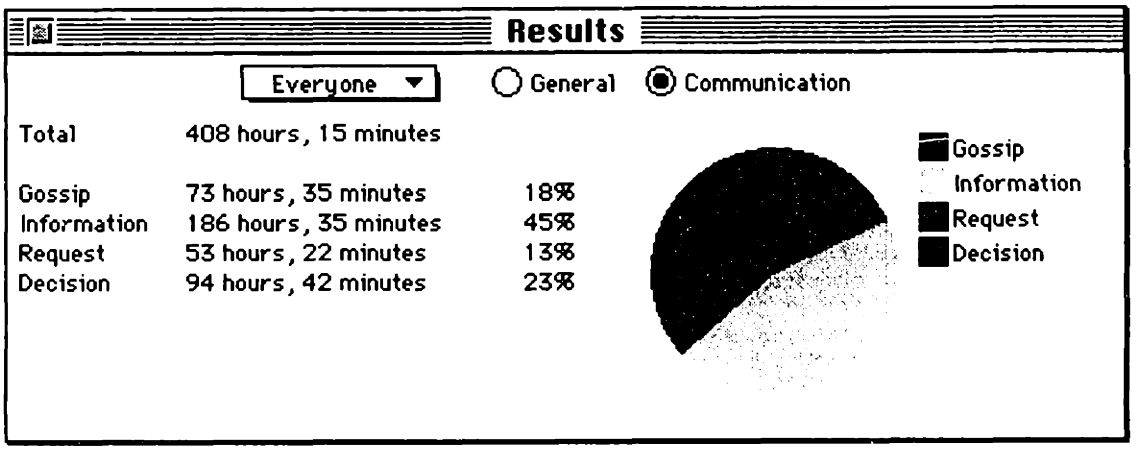


Figure 7.16: Percentage of time agents spent in the simulation talking about different types of topics. This figure was originally in color.

available from the company. But many are a reflection of the simulation and factors that it has a hard time modeling.

The first discrepancy (as seen in figure 7.7) is that the company data has days of very low work. These show up as gaps in the progress bands. The simulation data (figure 7.8) does not have nearly as many gaps. This occurs because the company data contains work that was done on the weekends. Any day that had any work done on it was assigned a unique day number. Since the quantity of work done on a weekend was generally much lower than the quantity done during the week, gaps appear.

The second discrepancy is that work on the *experiment* class is done earlier in the simulation than in the actual project (figure 7.10). This could be easily remedied by inserting a dependency or two that would prevent the *experiment* tasks from starting. However, an examination of company records indicated that the only factor holding up working on *experiment* tasks was the availability of prototype parts, and that these parts were available at the time. In fact, project priorities seemed to be the principle reason more experimentation wasn't done early—the engineers were busy working on more pressing matters. Hence this discrepancy was left in.

The third discrepancy is that the amount of communication recorded by the engineers is smaller than the amount of communication that occurs in the simulation. A total of 528 hours of communication was recorded in the timesheets. A typical simulation run yields 950 hours of synchronous communication and 290 hours of asynchronous. Roughly 23% of the synchronous communication is decision making, giving a total of approximately 1020 hours of combined non-decision communication. Hence roughly twice as much communication occurs within the simulation as was recorded in the actual project.

It is likely that more communication occurred in the actual project than was recorded. Timesheet entries were classified as discussion only if they explicitly mentioned discussions with other people. Short communications may go unreported as well as long communications where the person in question did not feel it necessary to note the specific type of activity being conducted.

Before taking into account the discrepancies, a comparison between corporate data and simulation data shows that the simulation takes approximately 136 days and the corporate data was a little longer at 146 days. These numbers are within 10% of each other. But the corporate data has many gaps and holes in it. Individuals went on vacation, worked on weekends, and had "slow" days (particularly at the beginning and end of the project). Is this good fit purely coincidental? There are two effects that must be accounted for: communication time in the simulation and slow days in the corporate timesheets.

To create a more accurate comparison between the simulated data and the corporate data, we can attempt to compensate for the extra communication time and the time spent working on weekends. The corporate data explicitly records 530 hours of discussion; the simulation reports approximately 1020 hours spent communicating. As a first-cut correction, the communication of each task in the simulation was scaled by 50%. Re-running the simulation resulted in approximately 700 hours of communication (excluding group decision making). At the 50% scaling, the simulation took an average of 126 days to complete.

I should point out that scaling the communication requirements of the model is reasonable because the original communication times were estimated based on available project notebooks and engineering judgment. Compared to the recorded project data, the simulated communication time was too high. Given that the recorded data are a lower bound for the likely amount of communication that occurred, the best I can say is that the initial estimates are not unreasonable, and that it is likely a portion of the recorded work is actually communication, although how much is communication cannot be determined. Scaling the quantity of information to be transferred is useful only for the purpose of matching corporate and simulation results. It also would be valid to leave the communication fixed and scale the quantity of work required in each task; this approach was not used in this discussion.

To account for weekends and "slow days" at work, the hours recorded for each day in the

corporate data were tabulated. Of the 146 days in the corporate data, 16 of them had between 0 and 4 hours of recorded activity and 19 had between 4 and 8 hours of recorded activity. Assuming each of the 16 slowest days is equivalent to 1/4 of a day and each of the 19 slow days is equivalent to 1/2 of a day, then there are 123 normalized days recorded in the corporate data.

When the data has been compensated for the extra communication time and weekend work days, the result is quite close: 126 days for the simulation versus 123 days for the corporate data. This close match is only a result of adjusting the work and communication requirements of the simulation to bring them into line with the corporate data.

Because of the close match between simulated and recorded data it is tempting to claim that the information-flow model has been convincingly demonstrated to be correct. In all fairness, I feel obligated to point out that in fact the majority of the time constraints expressed in this design project are the result of just two individuals: the team leader and the engineer. Figure 7.13 points out that these two were the only agents occupied continuously throughout the project. So although I think one can make a good case for the information-flow model, it is also possible to claim that the project takes this length of time simply because the two agents that are responsible for the large portion of it are working about as fast as they can. A more convincing demonstration of model power would involve a larger number of fully engaged engineers.

The modeling exercise has pointed out a number of weaknesses with the current version of the *DiFS* simulation. The two most important are the scheduling of planned start and stop dates and the addition of extra roles. If the experimenter could explicitly set target dates for the agents in the simulation to work towards, then agents could prioritize their efforts based on their perception of which tasks were on track and which tasks were behind schedule. This would also enable the agents to selectively stay late and work on important tasks with imminent due dates. The second improvement that would aid the experimenter would be the addition of a "consultant" role to the simulation. Throughout the corporate data there are a number of occurrences of one person acting as a short-term consultant to another person on a particular task. These incidents often comprised less than two hours of work and hence are too small to rate being added as separate tasks to the global model. A consultant role requiring a small quantity of discussion between the responsible agent and the consultant agent would capture these occurrences.

7.4.3 Correlations

Upon examining any of the pairs of figures depicting progress traces in the simulation and the real world it is natural to inquire whether or not the traces are correlated in some numerical sense. I have developed a correlation measure that can be used to compare progress traces.

Before comparing project traces with a correlation measure, the question of what constitutes a good correlation must be addressed. Assume a hypothetical 80 hour task that normally takes 10 days to complete. Perfect correlation between simulation and reality would be indicated if the simulated task and real task both started and stopped on the same day and logged the same number of hours per day. No correlation would be justified if the start and stop ranges of the two tasks did not overlap at all. Good correlation would be appropriate if the stop and start ranges missed by just a single day.

Two types of correlation measures are common in the literature but inappropriate for this problem. The first is the correlation coefficient ρ_{xy} from statistics or:

$$\rho(x,y) = \frac{E[(x-\mu_x)(y-\mu_y)]}{\sigma_x \sigma_y} \tag{7.1}$$

where x and y are random variables, $\mu_x = E[x]$, and $\sigma_x^2 = \mathrm{Var}(x)$. A second valid correlation measure would be the $\mathcal{L}^2$ inner product given by:

$$d(x,y) = \frac{\sum x_i y_i}{\sqrt{\sum x_i^2 \sum y_i^2}} \tag{7.2}$$

Both of these correlation measures range between -1 and $+1$, which is a little awkward. Moreover, they do not correctly account for data ordering. For example, assume there are two tasks X and Y and that one hour of work is done per day for N days either on task X or on task Y, ending with a total of $N/2$ hours on X and $N/2$ hours on Y. In this case $\rho(x,y) = -1$ and $d(x,y) = 0$ no matter what the order the work is done in. However, most people would agree that the correlation should be high if the work is done on alternate days and low if all the work was first done in one of the tasks.

The data ordering problem can be avoided by using the running summation of the work done per day. That is, if x_i is the number of hours of work done in task X on day i, then $X_i = \sum_{j=1}^{i} x_j$ is the quantity of work that has been accomplished by the i-th day. A measurement of the distance between two running summations will reflect the correlation between the two tasks while preserving the order of the data. Note that neither $\rho(x,y)$ nor $d(x,y)$ will work for the running summation. Both are strongly biased by the upward trend of the running summation and place more weight on the final elements than on the initial. This can be eliminated by subtracting off a linear term proportional to the number of days that have passed (i.e., $X'_i = X_i - ki$) but that runs into problems of what days are selected to be the starting and stopping points of the correlation.

Of the many possible functions that can measure the distance between running summations, I have selected one that is relatively easy to calculate and avoids scaling issues. Given two work traces x_i and y_i with N days total, and given the running summations $X_i = \sum_{j=1}^{i} x_j$ and $Y_i = \sum_{j=1}^{i} y_j$, then the correlation ϕ between the sequences is given by:

$$\phi(x,y) = 1 - \frac{\max_i |X_i - Y_i|}{\max(X_N, Y_N)} \tag{7.3}$$

The interpretation of ϕ is that it forms the ratio of the maximum distance between the running summations to the total size of the task. Normally the two tasks will have the same overall size, so the denominator may be replaced by $\sum_{i=1}^{N} x_i$ (which is just X_N). The ratio is subtracted from 1 so that the correlation will range from 0 to 1 in the direction of higher values meaning stronger correlation.

The ϕ correlation has nice properties. It avoids the data ordering problem by using the running summation of work accomplished. Two tasks that have starting and stopping ranges that do not overlap have a correlation of 0. Two tasks with work done on alternating days will have a high correlation. Scaling the quantity of work done does not affect the measure; that is $\phi(x^*, y^*) = \phi(x,y)$ where $x_i^* = kx_i$ and $y_i^* = ky_i$ for any constant $k \neq 0$.

Sample ϕ correlations between the simulated project and the timesheet data are shown in table 7.6. The values range all over the scale from 0 to 0.81. As discussed in section 7.4.2, one of the strong factors that influences these correlations is that the number of simulated days doesn't match perfectly with the number of actual days. Hence the tasks that occur later in the project are likely to be shifted a bit relative to each other and hence have lower coordination. Given this shifting effect, it is nice to note that the highest correlations occur towards the end of the simulation run.

7.5 Experiments

Once the corporate data had been entered and simulated, I ran a number of simple studies to gather further insight in the model. The first two studies reported here (sections 7.5.1 and 7.5.2) are efficiency and sensitivity studies conducted using an orthogonal array experimentation method. Section 8.1 starting on page 163 contains a detailed discussion of sensitivity studies and their ramifications. The third study reported here is a variation of the project model that simulates what might have happened if the project had not suffered a communications breakdown in the embodiment phase.

Table 7.6: Calculated ϕ correlations between simulated and actual data compressed across class.

Phase/Person	ϕ	Phase/Person	ϕ	Phase/Person	ϕ
BStorm PM	0.57	SM EN	0.27	PP1 T1	0.00
BStorm TL	0.71	SM ID	0.32	PP1 TL	0.69
DRI DR	0.06	SM T1	0.02	PP2 EN	0.71
DRI EN	0.29	SM TL	0.23	PP2 ID	0.46
DRI ID	0.47	E1 DR	0.52	PP2 T1	0.67
DRI PM	0.34	E1 EN	0.61	PP2 T2	0.81
DRI T1	0.00	E1 ID	0.51	PP2 TL	0.78
DRI T2	0.00	E1 PM	0.00	Tool EN	0.81
DRI TL	0.39	E1 TL	0.51	Tool [illegible]	0.61
DR2 EN	0.40	E2 EN	0.65	[illegible] T2	0.60
DR2 ID	0.25	E2 ID	0.70	Tool TL	0.57
DR2 T1	0.36	E2 T1	0.00	Rev EN	0.51
DR2 T2	0.00	E2 TL	0.56	Rev PM	0.25
DR2 TL	0.43	PP1 EN	0.63	Rev T2	0.55
SM DR	0.00	PP1 ID	0.49	Rev TL	0.55

7.5.1 Agent Efficiency

This experiment measured the sensitivity of the project duration to the work and communication efficiency of an individual agent. The efficiency of each agent was varied between 80% and 120% in 10% increments. An $L_{50}(2^1 \times 5^{11})$ orthogonal array [61] was used for the experiment, which gives 10 replications of each efficiency level and has 21 degrees of freedom left over for the error estimate (1 DOF for the grand mean, 28 DOF for 7 agents at 5 levels each, 50 DOF total).

Figure 7.17 plots the sensitivity of the project duration to variations in work efficiency by agent and figure 7.18 plots the sensitivity of project duration to communication efficiency by agent.

The work efficiency results are unambiguous. The team leader and the engineer are the only two agents whose work efficiency has a significant effect on project duration. For both of these agents a working efficiency of 80% results in a project duration increase of approximately 10%. This gives a sensitivity to work efficiency loss of approximately 0.5; that is, a 2% decrease in work efficiency will result in approximately a 1% increase in project duration.

The communication efficiency results are not as strong as the work efficiency results; this is not surprising for a project with a relatively small number of individuals. There are three individuals whose communication efficiency significantly affects project duration: the team leader, the engineer, and the program manager (the industrial designer is marginal). These results match our expectations; the team leader is responsible for the majority of the group decision tasks and the program manager is continually asking about the status of the various tasks in the project.

7.5.2 Work Sensitivity

This experiment measured the sensitivity of the project duration to variations in the amount of work in a task. The work time of each task was set to 50%, 100%, and 150% of its nominal value. The 108 tasks in the project model were divided into three groups of 36 tasks apiece. Each group was tested using an L_{81} orthogonal array; hence each task had 27 replications of each level of work time. A linear fit was assumed between the variation percentage and the project duration; hence 1 DOF goes to the mean value, 36 DOF to the slopes of the linear relationships, and 44 DOF for error estimation. Refer to appendix C for a discussion of the statistical calculations.

Figures 7.19 through 7.23 plot the work sensitivities for each of the tasks in the simulation.

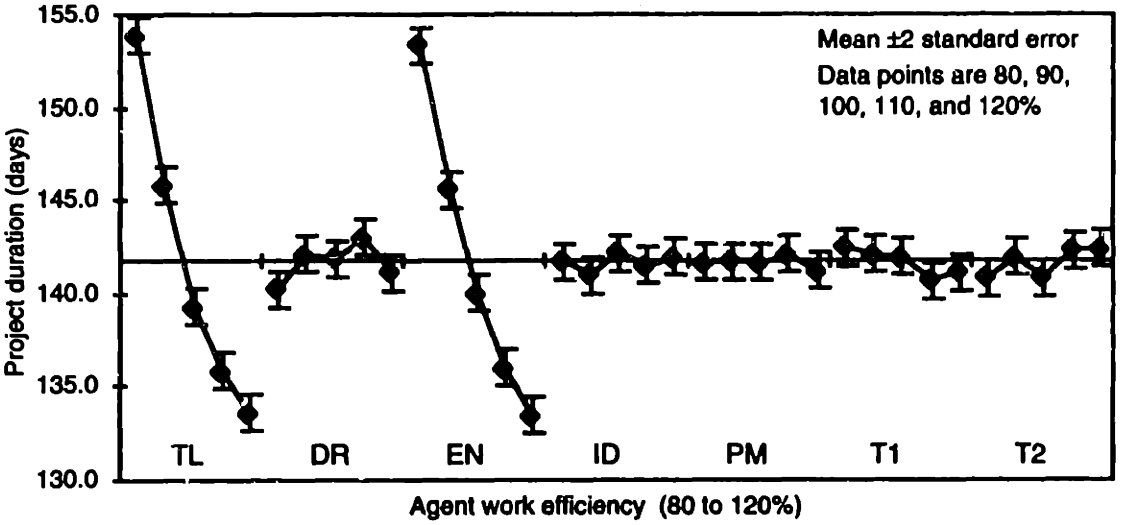


Figure 7.17: Sensitivity of project duration to agent working efficiency.

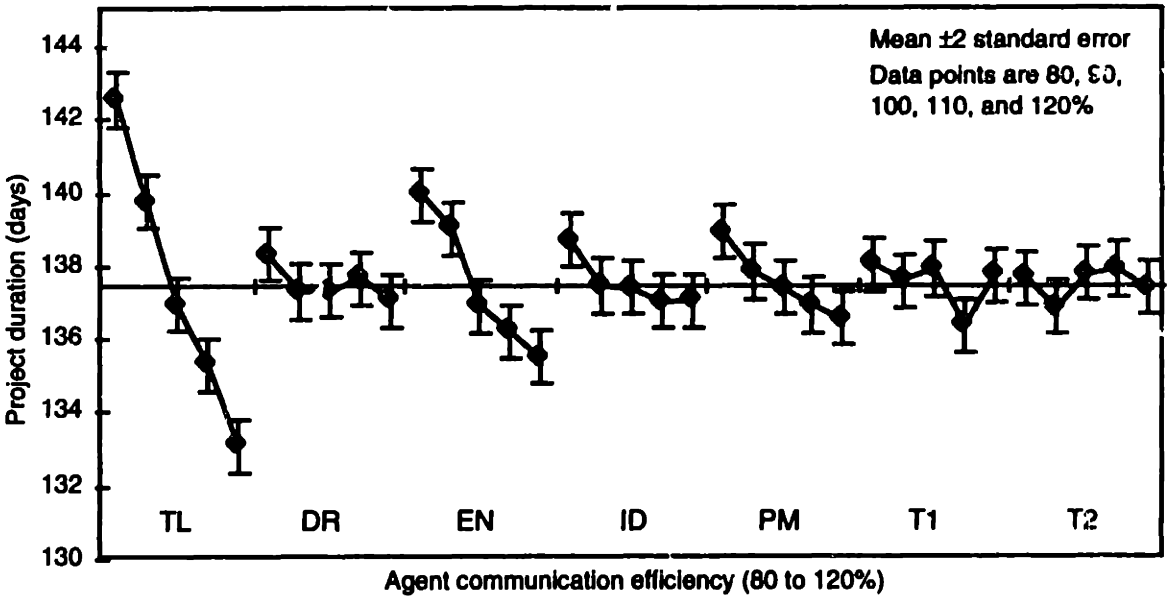


Figure 7.18: Sensitivity of project duration to agent communication efficiency.

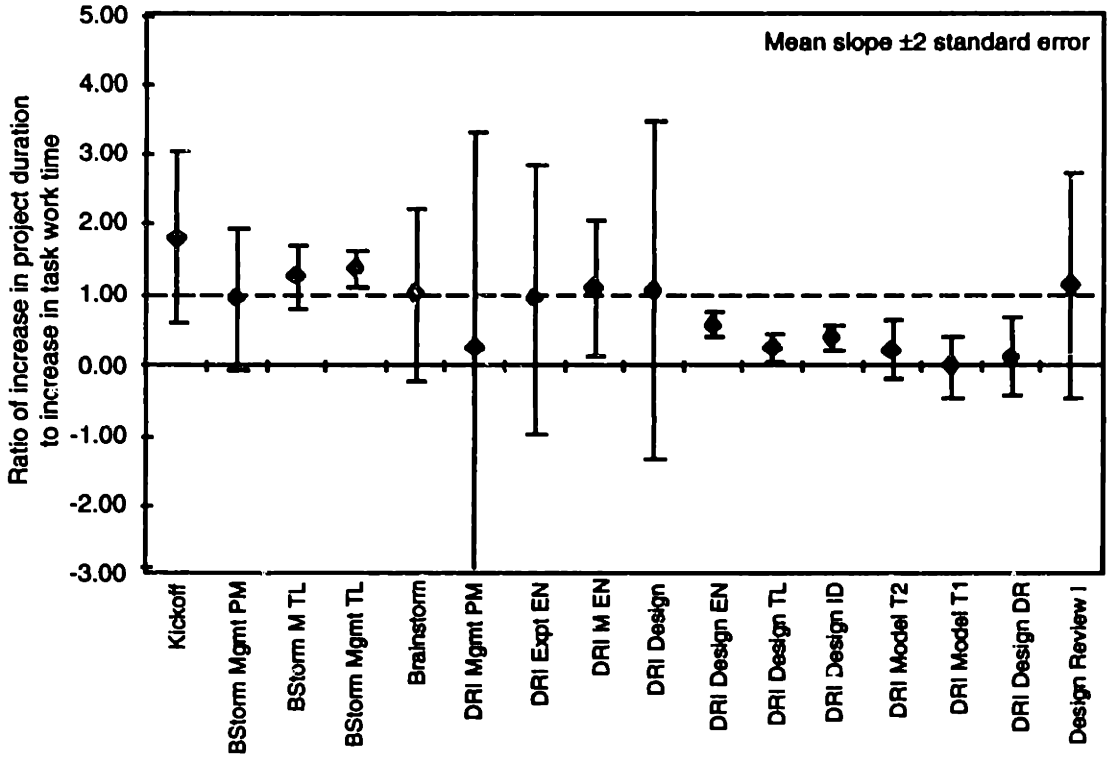


Figure 7.19: Sensitivity of project duration to task work time, part 1 of 5. Work hours were varied between three levels: 50%, 100%, and 150% of nominal.

The work sensitivity is the expected increase in project duration to a one hour increase in work time for an individual task. The work sensitivity can and does exceed 1.0; this occurs on group decision tasks where the extra hour of work often results in an extra meeting being scheduled.

Figure 7.24 presents the work sensitivities as the hours increase in project duration due to a 10% increase in task work time. The data are presented as a scatter plot that shows some correlation between the larger tasks and project duration.

To help interpret the large amount of sensitivity data presented, figures 7.25 through 7.29 present a diagram of the project model that has been shaded to accentuate large tasks and tasks that affect project duration. In these figures the size of an individual task is emphasized by increasing the thickness of the border around the task. The sensitivity of the project duration to the variations in task size (as given by figures 7.19 through 7.23) is shown by filling the tasks with a shade of gray that is darker with increasing work sensitivity.

Presenting work sensitivity figures serves to emphasize the tasks in the model that will have the greatest impact. Particularly important are *large* tasks that have a high sensitivity; a misestimate on the size of one of these tasks could result in a large change in the project duration.

7.5.3 Emergency Meeting

The engineering project was interrupted part way through the embodiment phase of the design by a series of miscommunications. The results were an emergency meeting with the client president and a quantity of lost work. As a part of this emergency meeting the program manager had a series of discussions with the client regarding billing the extra hours. The estimate at the time was that an extra 250 hours would be needed above and beyond the original quote to get the project back on track. The bulk of this lost work was in the industrial design and engineering categories.

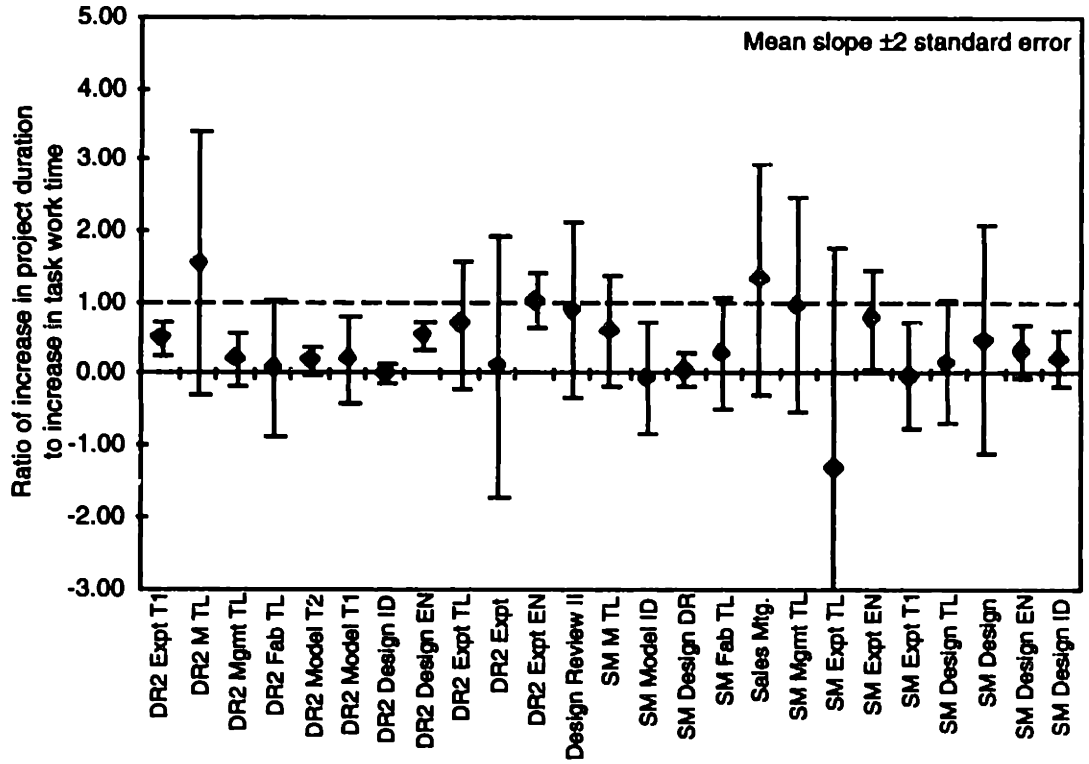


Figure 7.20: Sensitivity of project duration to task work time, part 2 of 5. Work hours were varied between three levels: 50%, 100%, and 150% of nominal.

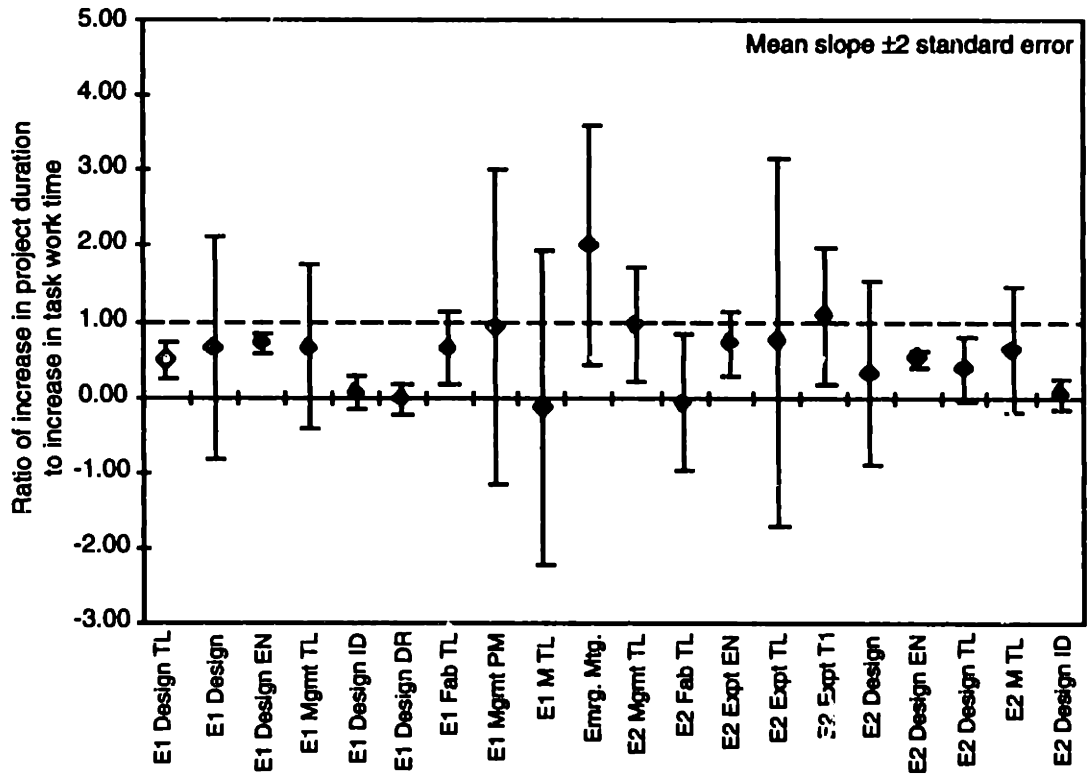


Figure 7.21: Sensitivity of project duration to task work time, part 3 of 5. Work hours were varied between three levels: 50%, 100%, and 150% of nominal.

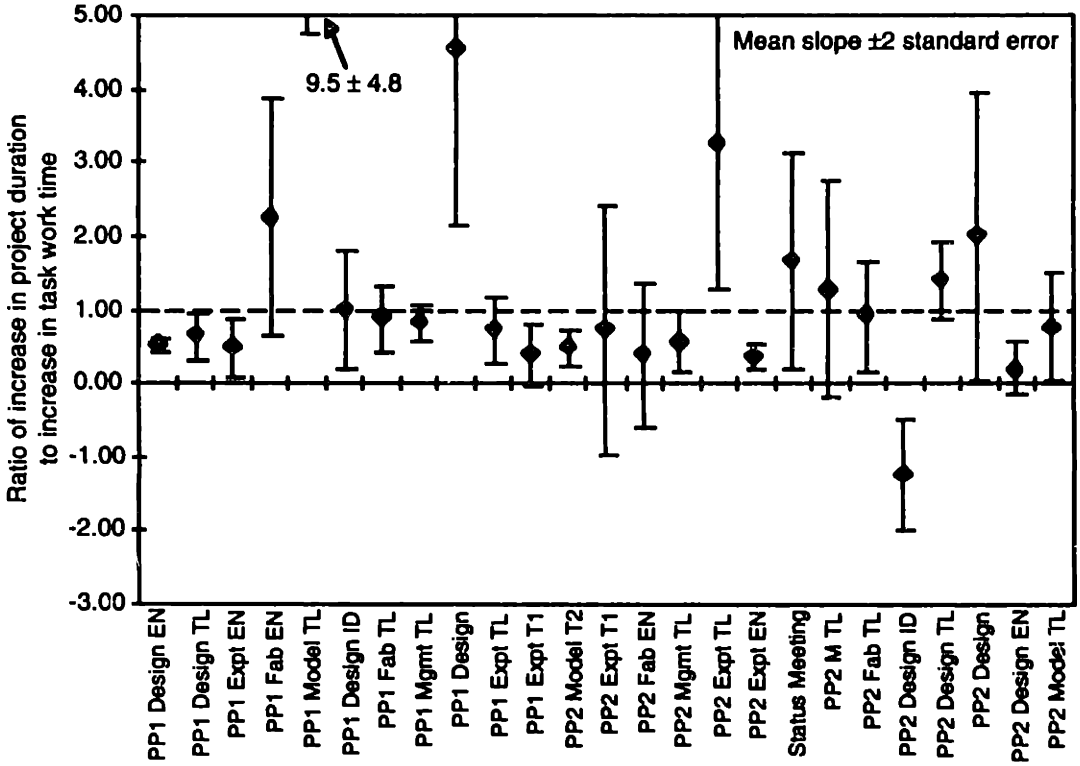


Figure 7.22: Sensitivity of project duration to task work time, part 4 of 5. Work hours were varied between three levels: 50%, 100%, and 150% of nominal.

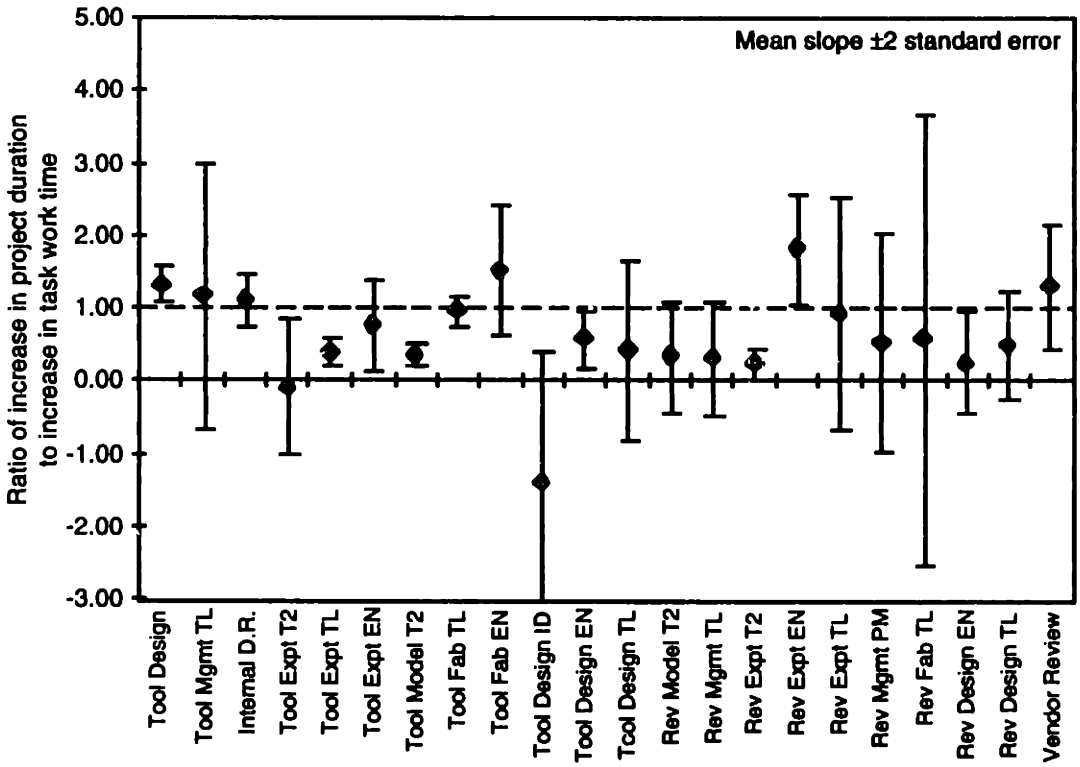


Figure 7.23: Sensitivity of project duration to task work time, part 5 of 5. Work hours were varied between three levels: 50%, 100%, and 150% of nominal.

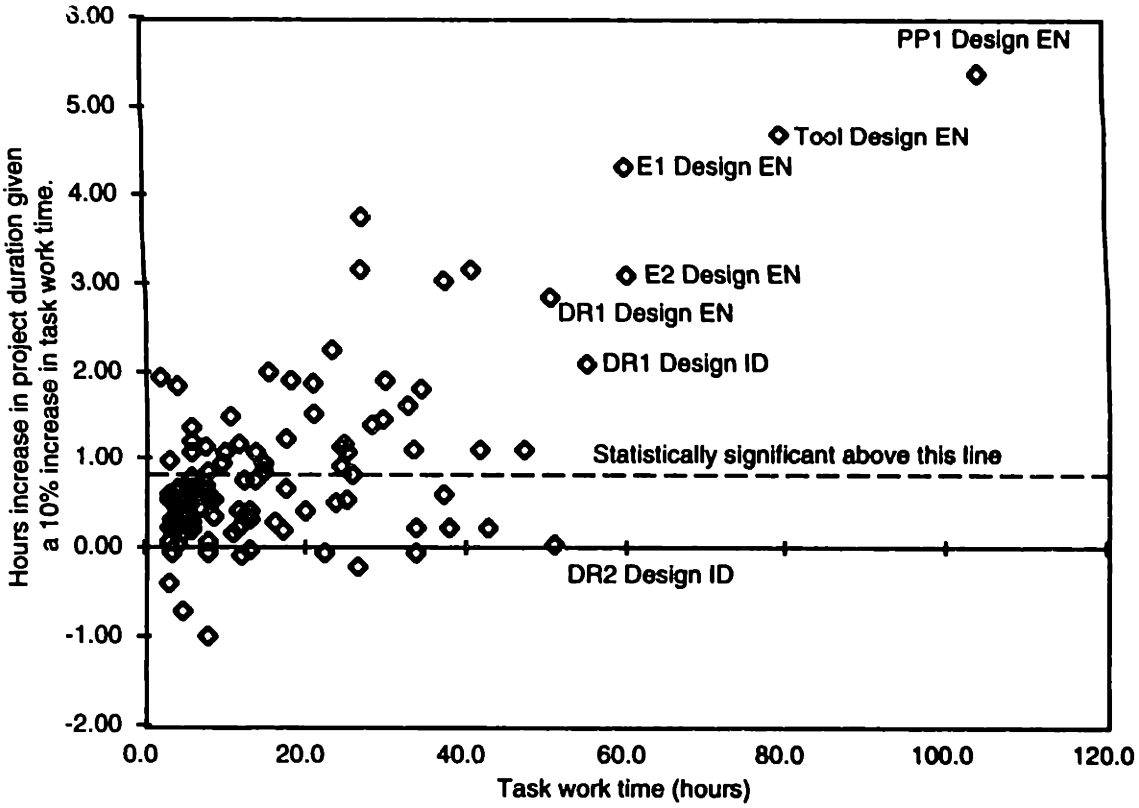


Figure 7.24: Increase in project duration due to a 10% increase in task work time.

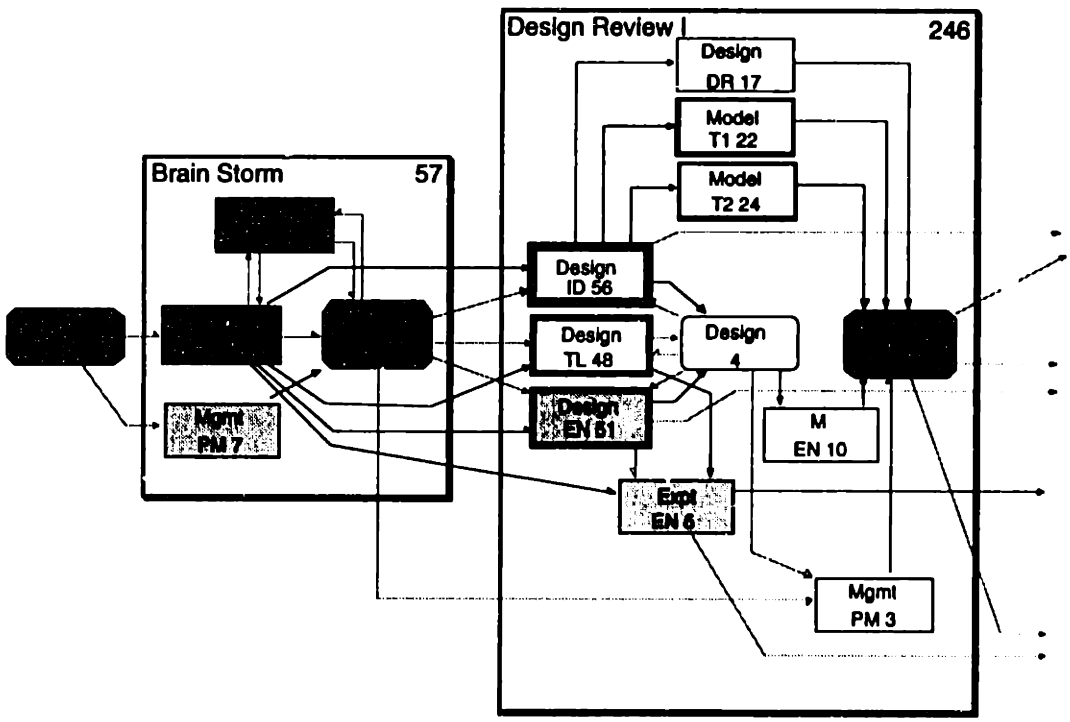


Figure 7.25: Illustration of the sensitivity of project duration to task work time, part 1 of 5. Grayness of task is proportional to sensitivity. Thickness of task border is proportional to task size.

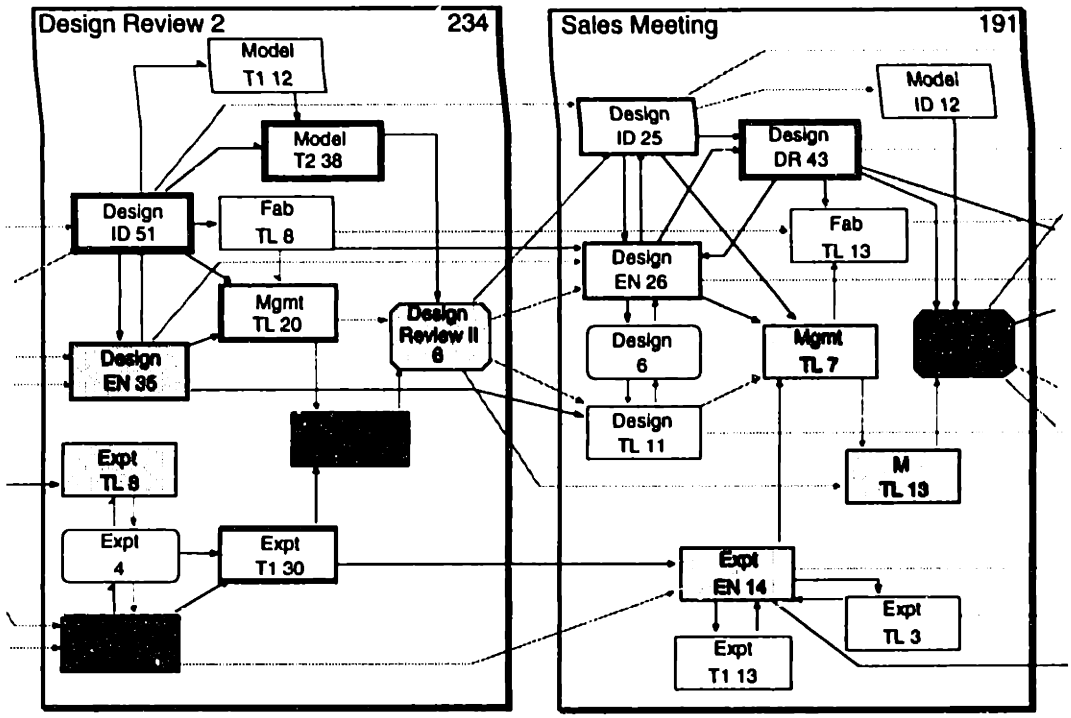


Figure 7.26: Illustration of the sensitivity of project duration to task work time, part 2 of 5. Grayness of task is proportional to sensitivity. Thickness of task border is proportional to task size.

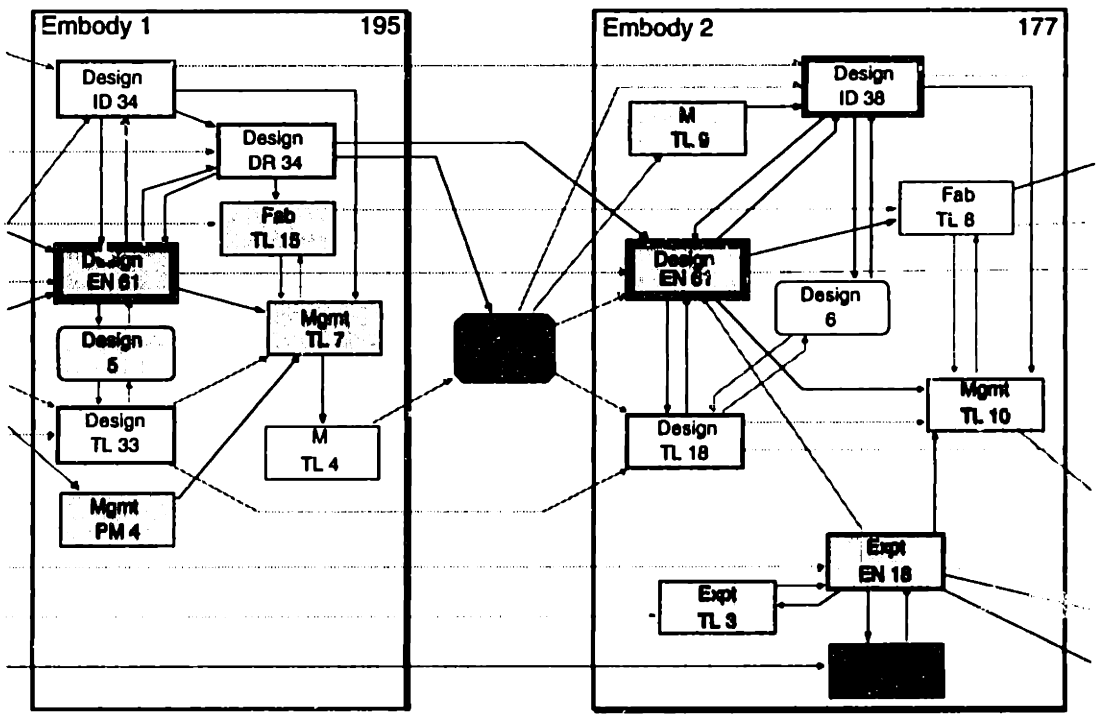


Figure 7.27: Illustration of the sensitivity of project duration to task work time, part 3 of 5. Grayness of task is proportional to sensitivity. Thickness of task border is proportional to task size.

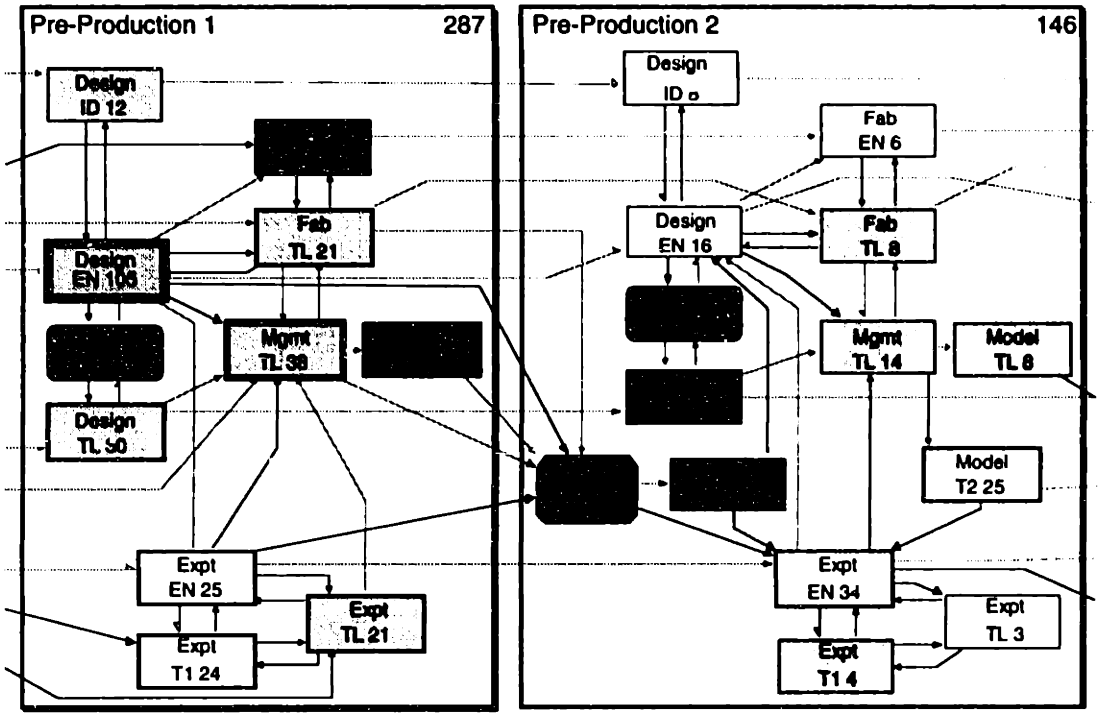


Figure 7.28: Illustration of the sensitivity of project duration to task work time, part 4 of 5. Grayness of task is proportional to sensitivity. Thickness of task border is proportional to task size.

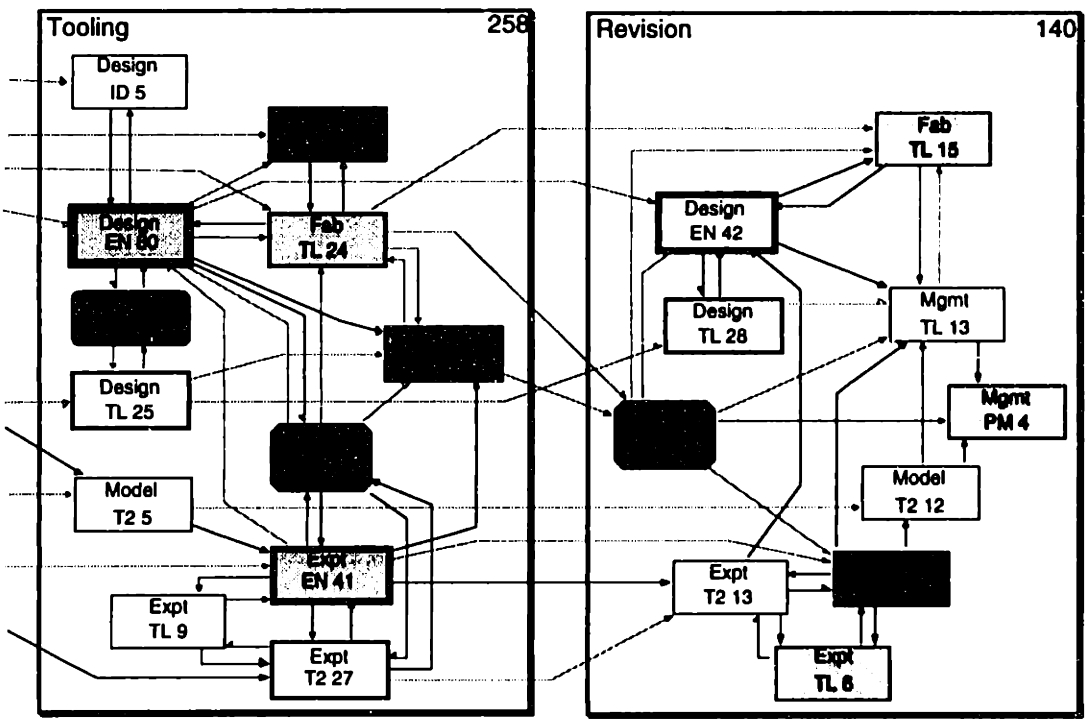


Figure 7.29: Illustration of the sensitivity of project duration to task work time, part 5 of 5. Grayness of task is proportional to sensitivity. Thickness of task border is proportional to task size.

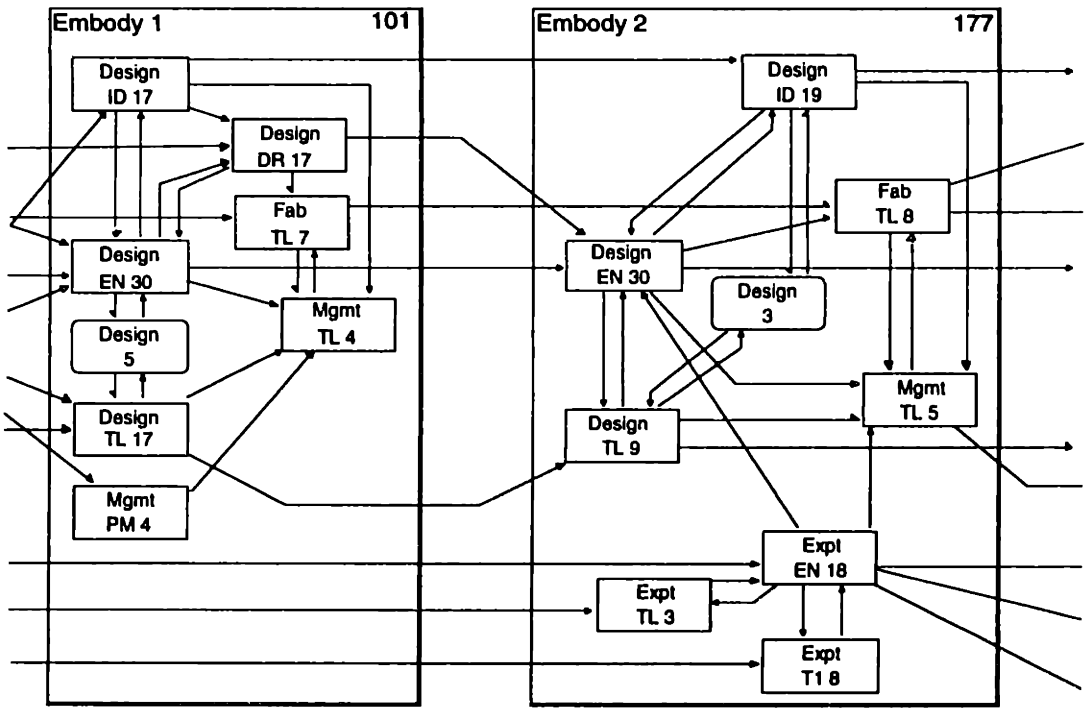


Figure 7.30: Alterations to project model to simulate not requiring an emergency meeting in the middle of the embodiment phase. Compare this figure to figure 7.4 on page 139.

It is possible to set up the *DiFS* simulation to reflect what might have happened had this series of miscommunications never occurred. A modified version of the project model was created that did not have an EMERGENCY MEETING task. In conjunction, the two management tasks (EMBODY 1:M TL and EMBODY 2:M TL) were removed as they were tasks that dealt with the changes introduced by the miscommunication. On the assumption that some of the work done before the emergency meeting was valid and that some of the work done after the emergency meeting were changes based on the emergency meeting, the sizes of each of the DESIGN tasks were cut by 50% (both work and communication time). The FABRICATION tasks in the embodiment phase represented discussing parts with vendors; as this work changed only a little because of the emergency meeting, only the work in the FABRICATION task in the "Embody 1" phase was cut by 50%. The EXPERIMENT tasks were not changed. Changes are shown in figure 7.30.

The result of removing the EMERGENCY MEETING task and cutting work in the DESIGN tasks was to reduce the total work in the embodiment phase from 372 working hours to 204 working hours. The simulated project duration dropped from 136.5 days to 127.2 days, and a matching decrease in billable hours from 3360 hours to 3070 hours (billable hours are hours logged by computer agents while the agent is not idle).

The effect of the simulation modifications was to reduce the simulation run time by almost two weeks and to reduce the billable hours by 290 hours. These results are relatively close to the official corporate estimates of 250 billable hours lost and approximately two weeks of time.

7.6 Summary

This chapter has presented a validation experiment performed in cooperation with a local design company. The purpose of the validation experiment was to demonstrate that (1) an actual engineering project could be represented in the *DiFS* simulation, (2) that the simulation could generate plausible results, and (3) that the simulation could be used to make predictions about

the engineering project.

Data gathered on the corporate project consisted of interviews with the individuals who worked on the project, the project documentation, and a database of timesheet entries. The timesheet data were categorized by phase, type of work, and class of work. The categorized data were then mapped into a *DiFS* project model and appropriate dependencies between the tasks were added.

Simulation results are comparable to corporate timesheet entries. After compensating for slow days in the corporate database and communication overestimates in the simulation model, the project durations match up within a few days of each other. Visual and calculated metrics between simulation and corporate time traces show a good correlation between the two.

A series of experiments were run to examine the results of the simulation. A sensitivity study on agent working efficiency and communicating efficiency demonstrates that only the two primary agents working on the simulation have a major impact on the project duration. A sensitivity study on task work time graphically illustrates the tasks in the simulation that have the strongest effect on project duration. Finally, because the corporate project had a communication problem in the embodiment phase, a "what if" experiment tests what would have happened had the miscommunication not occurred. The results of this experiment match well with what was predicted by the program manager at the time of the miscommunication.

The validation study of the corporate project demonstrates the sufficiency, plausibility, and predictive power of the *DiFS* simulation. The corporate project was successfully instantiated as a *DiFS* model, the results of running the model matched well with the corporate data, and the project model predictions agreed with corporate opinion. The validation study also served to point out a number of additional features and capabilities that should be added to the *DiFS* simulation; in particular, the scheduling of task completion dates.

Chapter 8

Project Management Metrics

This chapter addresses the question of how the *DiFS* simulation may be used by managers to understand the design process better and to improve their own design projects.

Four areas of management interest are discussed in this chapter: criticality, interdependency, personnel allocation, and the effect of the office environment. The criticality section addresses the question of which tasks or agents most strongly affect project performance. The interdependency section discusses how to isolate the loops in task dependencies and how to measure their strength. The personnel allocation section deals with how team size affects project performance. Finally, the office environment section shows how agent office location, communication methods, and behavior can affect a project outcome.

8.1 Criticality

8.1.1 Concept

The *critical path* of a network of tasks is the longest path or sequence of connected activities through the network. Tasks on the critical path are the limiting factor in how quickly the entire project can be completed. By definition, delaying a task on the critical path will delay the entire project; delaying a task off of the critical path will only delay the entire project if the task is delayed long enough so that it becomes part of the critical path.

The definition of critical path must be extended when discussing PERT networks. The size of a task in a PERT network is not a single fixed number. The size is normally given by three numbers: an optimistic estimate, a most probable estimate, and a pessimistic estimate. Because the size of the task varies, the concept of the critical path must be extended. The *criticality index* of a task is defined to be the percentage of time that a task is on the critical path of the network [83]. The criticality index is normally determined by running a Monte Carlo simulation of the PERT network.

The *DiFS* simulation differs from traditional PERT networks in that it allows tasks to be mutually dependent upon each other. Loops in the dependencies preclude the possibility of directly calculating whether or not a task lies on the critical path.

However, the concept of criticality can be experimented with in the *DiFS* simulation. The concept of criticality is that a critical task in a project network is one that directly affects the project duration. This is equivalent to saying that changing the size of the task would change the project duration. That is, if the size of a critical task increases, the project duration increases, and if the size of a critical task decreases, the project duration decreases. This can be tested in the simulation by running a series of trials that systematically vary the size of a task and measuring the project duration.

This concept of criticality is similar to the one expressed in networks of *Generalized Precedence Relations* or GPRs. Elmaghraby [24] discusses how GPRs allow the experimenter to

specify relationships between the start and end times of tasks. These relations are similar to the ones employed by *DiFS*, but not a logical subset because of extra relationships that can specify task duration relations. For example, in a GPR you can specify that a task is to be 40 hours long and that the difference between its start and finish time is to be no longer than 50 hours.

8.1.2 Approach

The criticality of a task is defined to be how strongly a variation in task size will affect the project duration. Consider the SIMULATOR DEVELOPMENT project, described in section 5.5 on page 107 (see figure 5.35 on page 110). At first glance it is unclear which tasks in this particular project will fall upon the critical path. Running the simulation (see figure 5.39 on page 112) suggests that Joe is the busiest agent and hence the tasks Joe is working on will probably be on the critical path, but other tasks may also be critical path. Note that the simulation definition of critical path is strongly a function of agent role assignments. A task that is critical under one set of assignments may not be critical under a different set.

To calculate the criticality of each task I ran a series of trials that systematically varied the size of each task and measured the project duration. The work time of each task in the project was varied between 4 levels: 75%, 100%, 125%, and 150% of nominal work time. There are 17 tasks within the project, and it can be assumed that variations in task size are linearly additive, so an orthogonal array experiment is suitable. The L'_{64} array varies 21 factors at four levels each over 64 trials, so it is an appropriate choice.[1] The L'_{64} array has 64 degrees of freedom; 1 is used for the grand mean and 51 for the 17 tasks at 3 DOF each. That leaves 12 DOF for error estimation.

Sample results from running the orthogonal array experiment are plotted in figure 8.1. A representative 4 of the 17 tasks are plotted. From this figure it is clear that variations in the sizes of the CODING and DEBUGGING tasks have a significant effect on the project duration, but that variations in the sizes of the SOFTWARE TESTING and SOFTWARE DISCUSSION tasks have no effect.

The next problem is to quantify how critical a task is. For small variations of task size, the relationship between task size and project duration is roughly linear. I define the *criticality* of a task to be equal to the slope of the line fitted to this curve. That is, if D is the project duration and W_A is the work time of TASK A, then the criticality C_A of TASK A is given by:

$$C_A = \frac{\partial D}{\partial W_A} \tag{8.1}$$

where C_A is evaluated at W_A equal to its nominal value. This measure C_A is the sensitivity of the project duration to a variation in task size. Another equivalent measure of criticality that can be used is given by:

$$C'_A = W_A \frac{\partial D}{\partial W_A} \tag{8.2}$$

This measure C'_A is the sensitivity of the project duration to a *percentage* variation in task size. It has units of time, whereas C_A is non-dimensional.

The first version of task criticality, sensitivity to variation in task size, is the the ratio of the increase in project duration to the increase in work time, where both are measured in hours. This measure shows how much longer the project becomes if just a single hour is added to an given task. The criticality and related error bars for the tasks in the sample project are shown in figure 8.2. Appendix C contains a discussion of how these slopes and error bars are calculated.

The second version of task criticality, sensitivity to a percentage variation in task size, calculates the change in project duration given a percentage error in estimating task size. This metric represents the concept that a 10% underestimate of the size of a 10 hour task is of less

[1] If you are copying this array out of the book by Phadke [61], be warned that the entries in column 11, trials 39 and 40 are reversed.

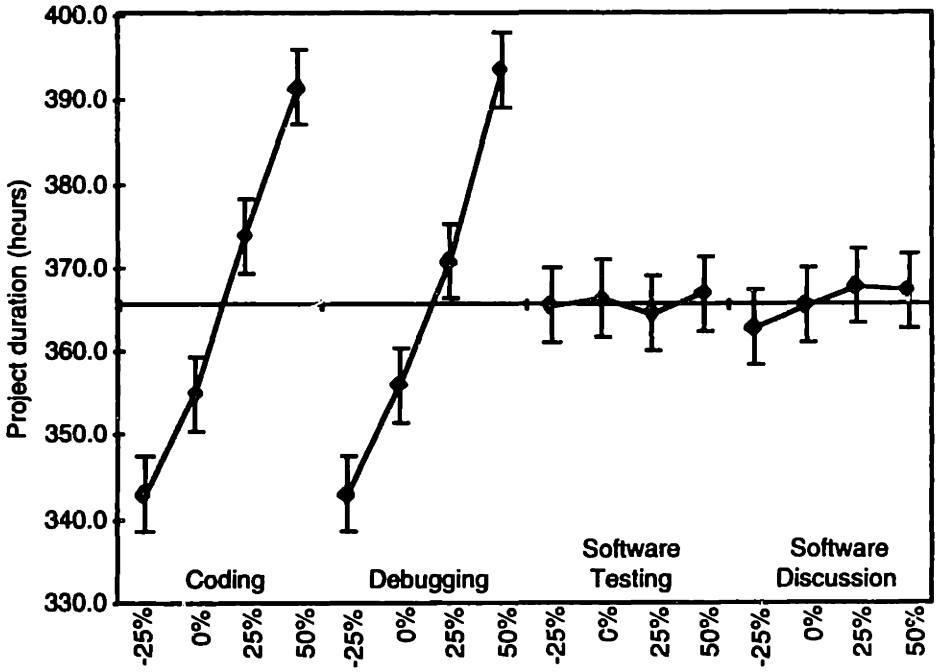


Figure 8.1: Sample of the relationship between project duration and size of a work task. Mean value plus or minus two standard errors are plotted for each level of work time by task.

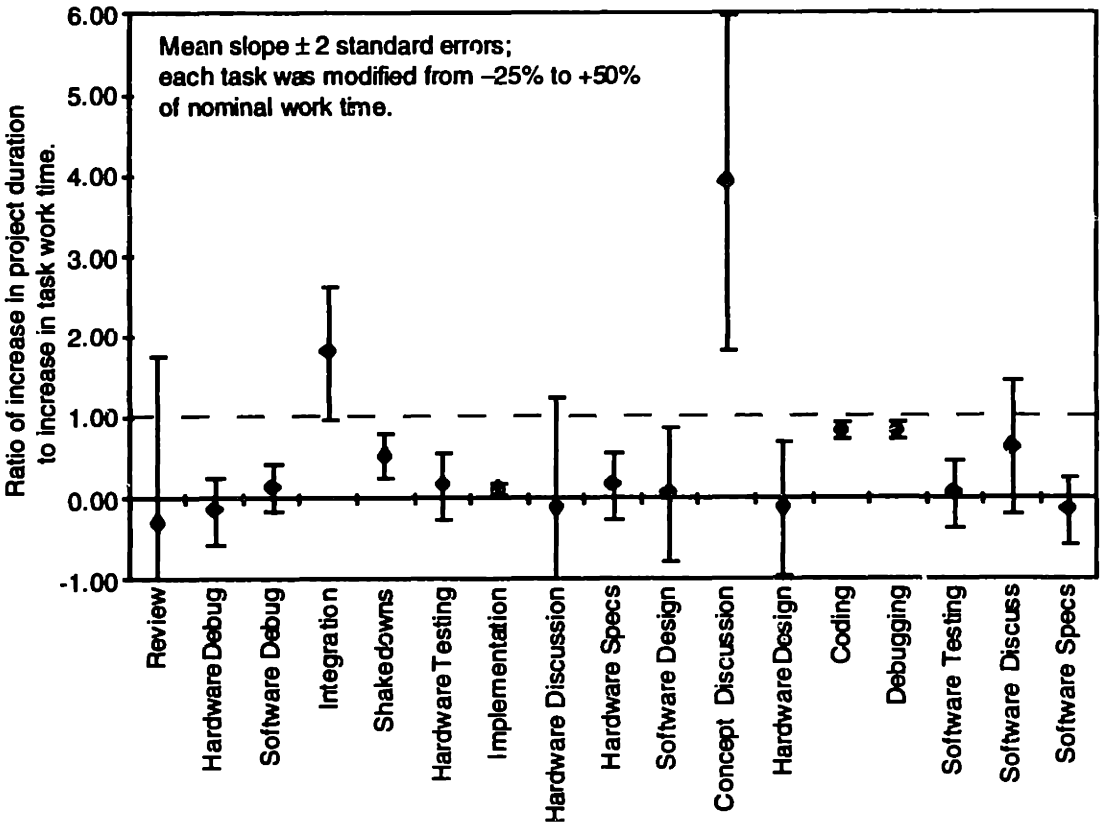


Figure 8.2: Sensitivity of project duration to variation in task work time. Work time was varied between four levels: 75%, 100%, 125%, and 150% of nominal.

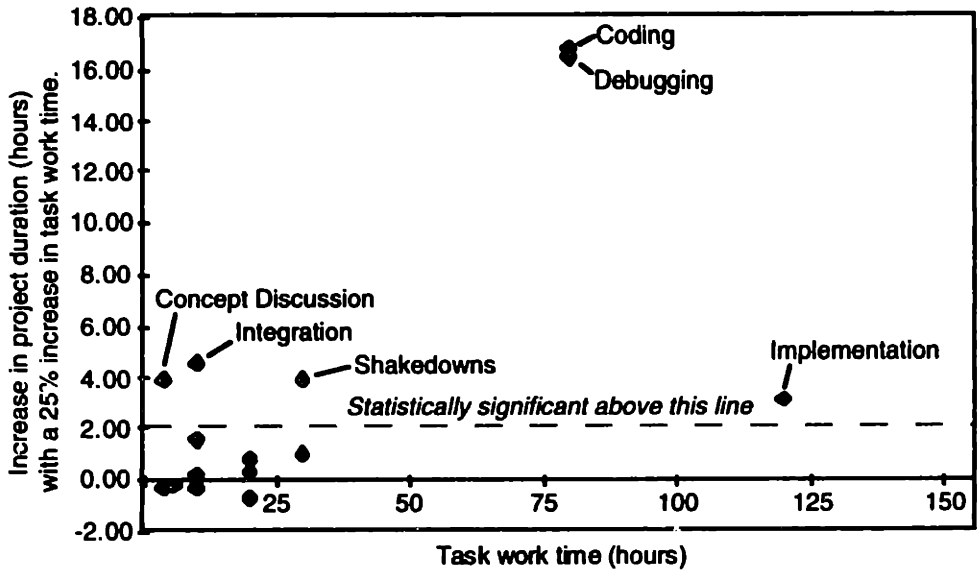


Figure 8.3: Sensitivity of project duration to percentage variation in task work time, plotted against task work size. Work time was varied between four levels: 75%, 100%, 125%, and 150% of nominal. Standard error is plus or minus 2 hours.

consequence than a 10% underestimate of the size of a 1000 hour task. Figure 8.3 is a scatter plot of the percentage task criticality measure versus the size of individual tasks.

Both figure 8.2 and figure 8.3 have their uses. The former can be used to identify tasks that, if delayed, will strongly affect the project duration. The latter can be used to identify tasks that strongly affect project duration if misestimated in size.

In the sample project, the INTEGRATION and CONCEPT DISCUSSION task have a very strong influence. In fact, adding an hour to the CONCEPT DISCUSSION results in approximately 4 extra hours of simulation time. This *super-criticality* is possible because CONCEPT DISCUSSION is a group discussion task in which three agents participate. Adding an hour of work normally would mean that an additional meeting would need to be scheduled, resulting in more than an hour's delay.

Consider figure 8.3. As influential as the CONCEPT DISCUSSION task is, it is still only a 4 hour task. When setting up the simulation, an 50% underestimate of the task size will add another 16 hours to the project duration, or a little under two days. However, the CODING and DEBUGGING tasks are considerably larger tasks; underestimating either of them by 50% will result in a 68 hour increase in project duration, or approximately a week and a half.

Figure 8.4 is a graphical illustration of the criticality of each task in the sample project. The shading in each task is proportional to the experimentally measured criticality of the task. The "critical path" of this project appears to pass primarily through the SOFTWARE development.

8.1.3 Slack

The *slack* of a task, as defined by Wiest [83], is the amount of time that a task could be delayed without affecting the project duration. Elmaghraby [24] uses the synonymous concept *float* to describe a number of ways of quantifying task slack. The one considered here is the *total float*, or the amount of time that a task can be delayed without affecting the project duration.

The current version of the *DiFS* simulation cannot instruct an agent to delay starting a task. However, it is possible to plot the project duration as a function of the size of a given task. The shape of this curve can be used to estimate the slack of a task. Note that this is an approximation to the total float because the agent responsible for the task will have extra work

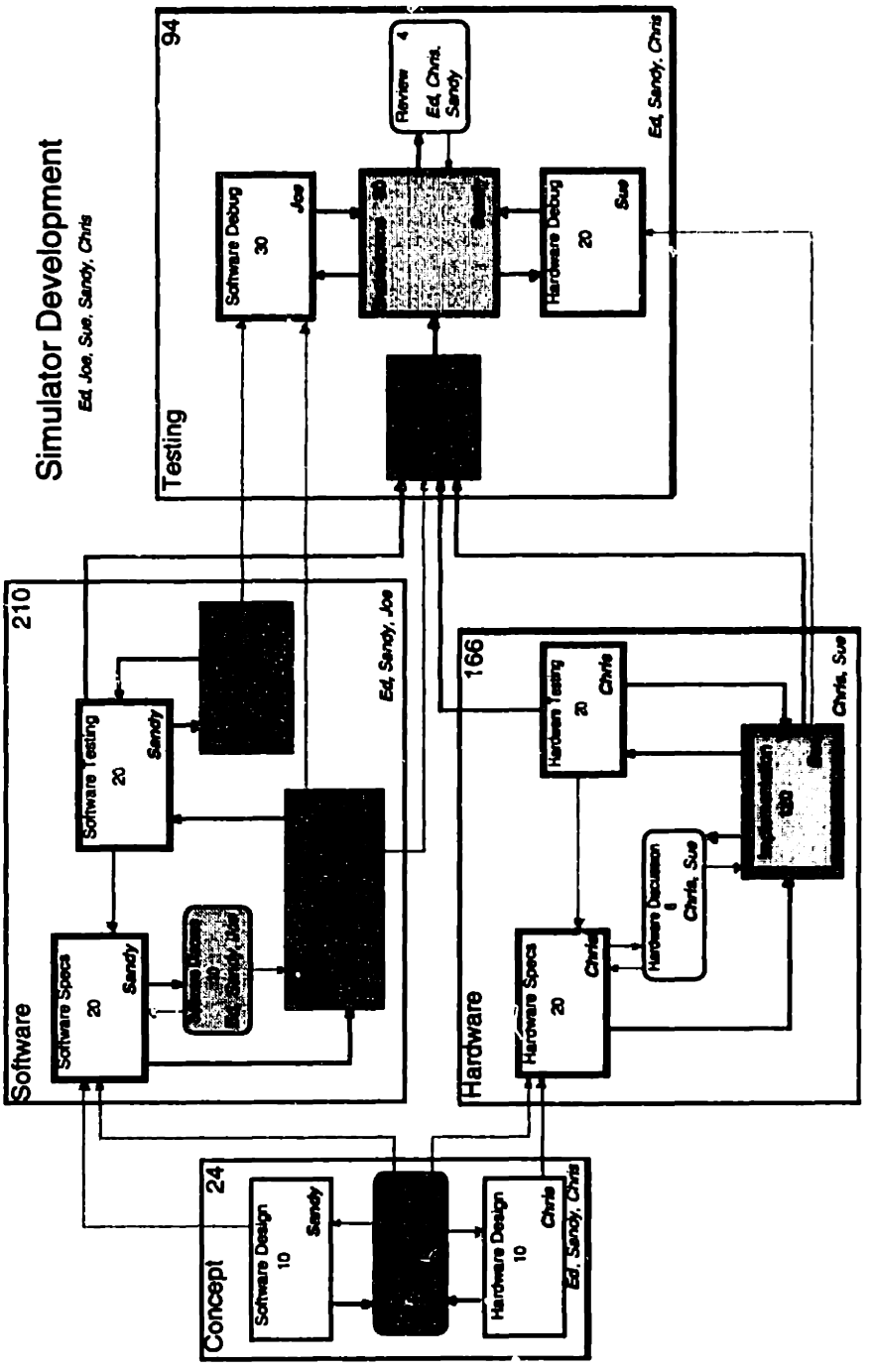


Figure 8.4: Graphical illustration of task criticality. The shading of each task is proportional to its criticality. The thickness of the border of each task is proportional to its work time.

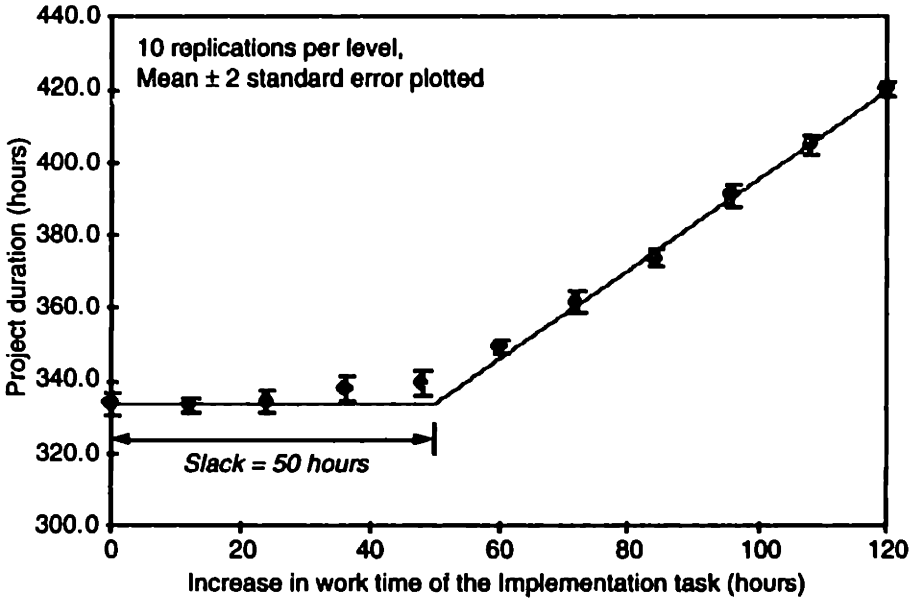


Figure 8.5: Slack time of IMPLEMENTATION task. The nominal work time in this task is 120 hours.

to do and hence not be free to work on other tasks.

Consider the SIMULATOR DEVELOPMENT project shown in figure 5.35 on page 110. The prior section demonstrated that the critical path of this project flows primarily through the SOFTWARE management task and associated subtasks. Figure 8.5 is a graph of the slack available in the IMPLEMENTATION task of the HARDWARE management task. This graph was created by systematically increasing the work time of the IMPLEMENTATION task from 120 hours to 240 hours and recording the project duration. Note that the initial increases in work time have no effect on the project duration. But after approximately 50 hours of work have been added, the project duration becomes linearly related to the work time in the IMPLEMENTATION task. After this point, the IMPLEMENTATION task is on the critical path of the project. Note that the criticality calculations for the IMPLEMENTATION task indicated it was marginally on the critical path; this is because the task size was varied from 90 to 180 hours and the IMPLEMENTATION task is critical path for a 180 hour task work time (a 60 hour increase in task work time).

Criticality calculations indicated that the CODING task was on the critical path. Figure 8.6 shows an attempt to determine how small the CODING task would have to be made in order to take it off of the critical path. As the figure shows, even when the CODING task is reduced to just 10% of its original size, it is still on the critical path. This happens because the critical path of the project runs through both the CODING and the DEBUGGING tasks; reducing one without changing the other still keeps both on the critical path.

8.1.4 Extensions

The sensitivities to variation in work time of a task are one way of locating that tasks that are on the critical path of a project. This concept may be generalized to other types of task and agent parameters if we define the criticality of a parameter to be how variations in that parameter affect the project duration.

There are many different types of parameters in a project that can be tested for criticality: variations in task work time and communication time, variations in agent working and communicating efficiency, and variations in individual dependency information requirements. Each of these parameters may be systematically varied to estimate how sensitive the project duration is to variations of the parameter. Critical parameters are ones that strongly affect the project duration.

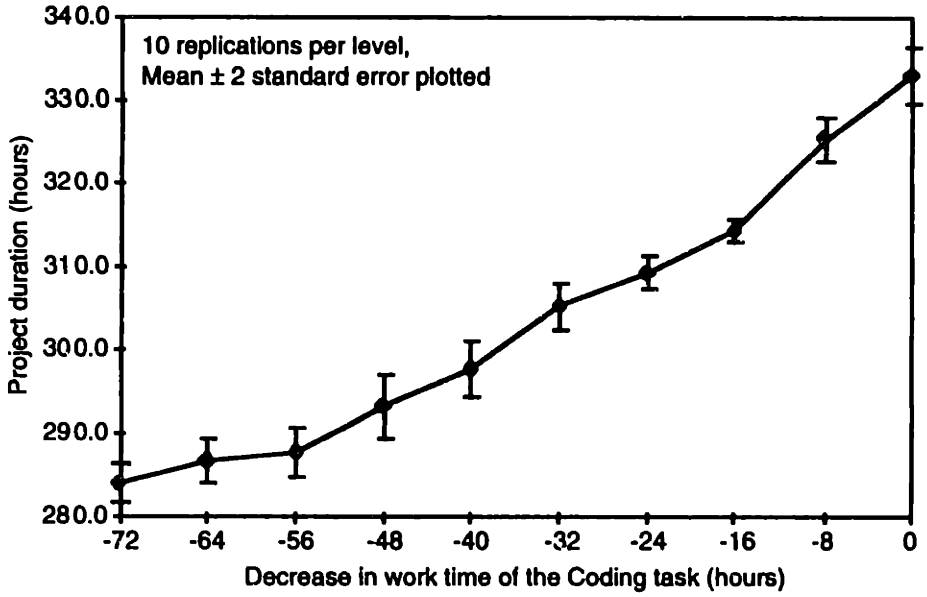


Figure 8.6: Slack time of CODING task. The nominal work time in this task is 80 hours. This is an example of task that has no slack; no matter how small it is, it is always on the critical path.

Figures 8.7 and 8.8 measure the sensitivity of the project duration to variations in task communication time. Tasks that are critical in communication time are tasks where a great deal of information needs to be transferred to another agent and the timely completion of this information transfer affects the project duration.

Figures 8.9 and 8.10 show the effect of varying the working and communication efficiency of an individual agent. Here the concept of criticality is applied to the idea that particular agents within the project may strongly influence the project duration. Figure 8.9 clearly shows that Joe's work efficiency strongly affects the project (Joe is the individual responsible for the CODING and DEBUGGING tasks). In an individual sense, Joe's work is on the critical path; if he takes a vacation or gets sick, the entire project will slip.

Figure 8.10 demonstrates that Edward's communication skills are the most influential among the five agents. Edward is the top level manager who oversees the entire project and leads a number of discussions about the software side of the project.

8.1.5 Criticality Summary

The criticality of a task in a PERT network is defined by the percentage of time it spends on the critical path; that is, the longest path through the network. Within the *DiFS* model, the criticality of a parameter is defined to be the sensitivity of the project duration to variations of that parameter.

The sensitivity of project duration to parametric variations has been explored for work and communication requirements for tasks and for work and communication efficiencies for individuals. The sensitivity of project duration to variations in task work time most closely matches the traditional definition of criticality. Tasks that fall on the traditional critical path are reflected in *DiFS* as having a high criticality; that is, a variation in the task work time strongly affects the project duration. Tasks that do not fall on the traditional critical path have a low criticality; variations in the sizes of these tasks does not affect the overall project duration. Sample slack calculations have been presented.

The slack of a non-critical task is defined to be the maximum increase of the task work time before the task becomes critical. A critical task may have a negative slack or no slack at all. A negative slack indicates how small the task must become before it moves off of the critical path.

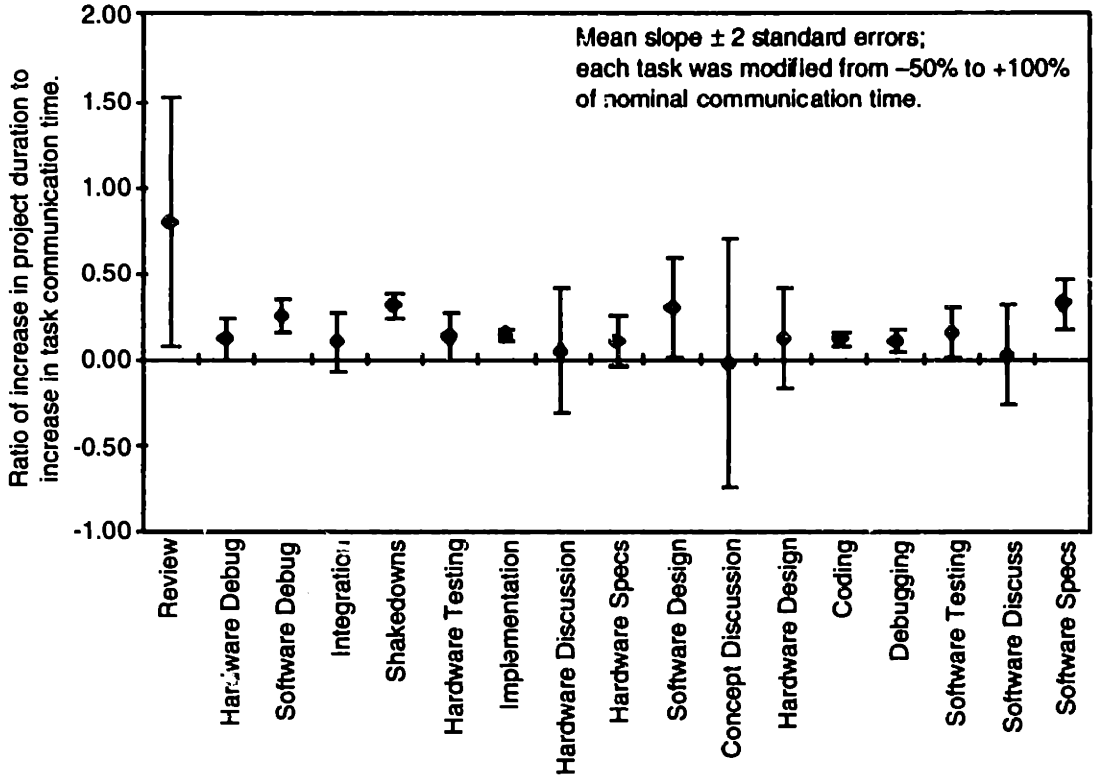


Figure 8.7: Sensitivity of project duration to variations in task communication time. Communication time was varied between four levels: 50%, 100%, 150%, and 200% of nominal.

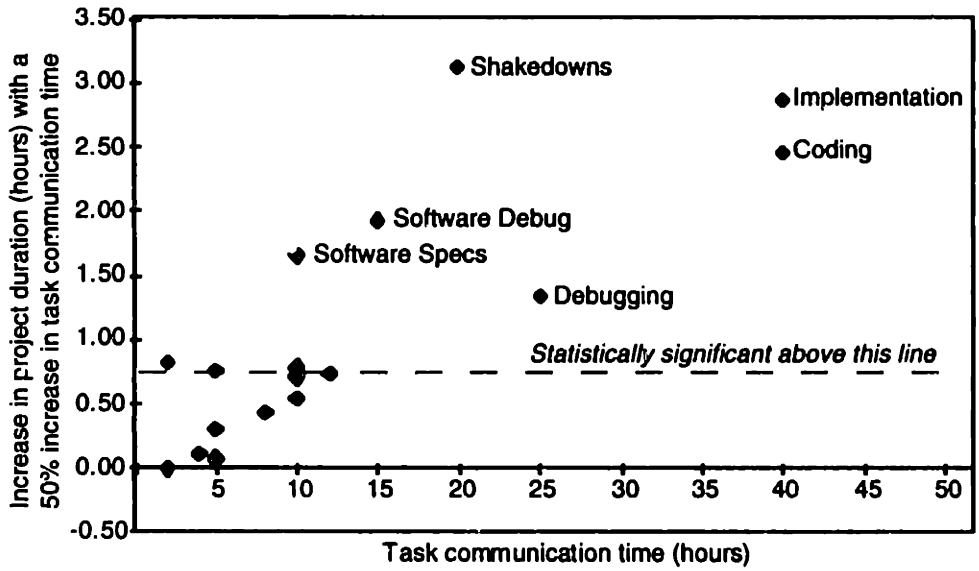


Figure 8.8: Sensitivity of project duration to percentage variation in task communication time, plotted against task communication time. Communication time was varied between four levels: 50%, 100%, 150%, and 200% of nominal.

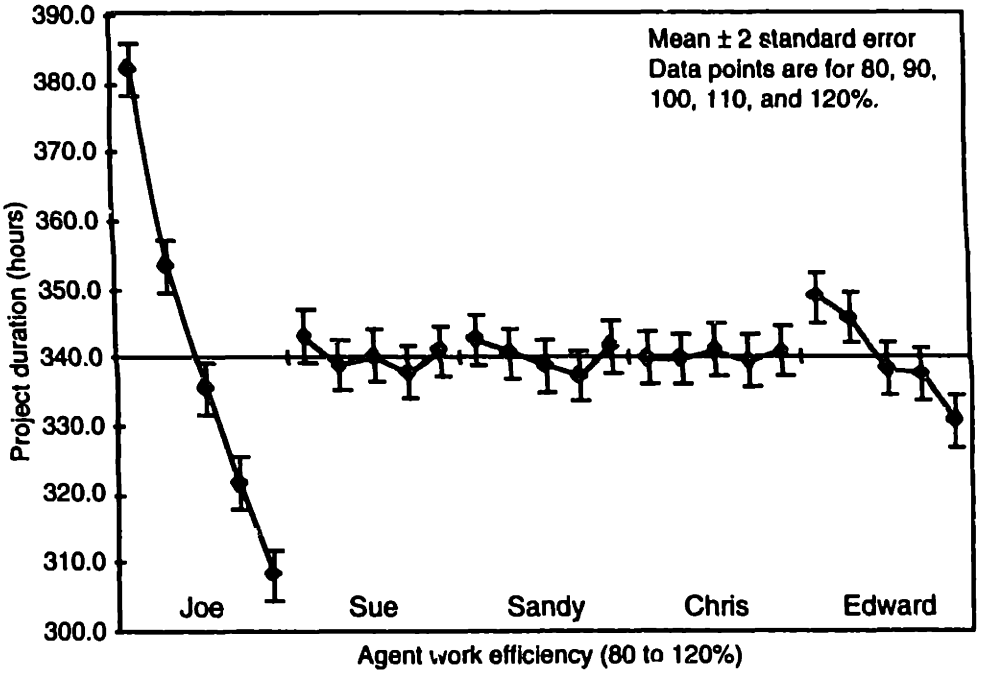


Figure 8.9: Sensitivity of project duration to agent work efficiency.

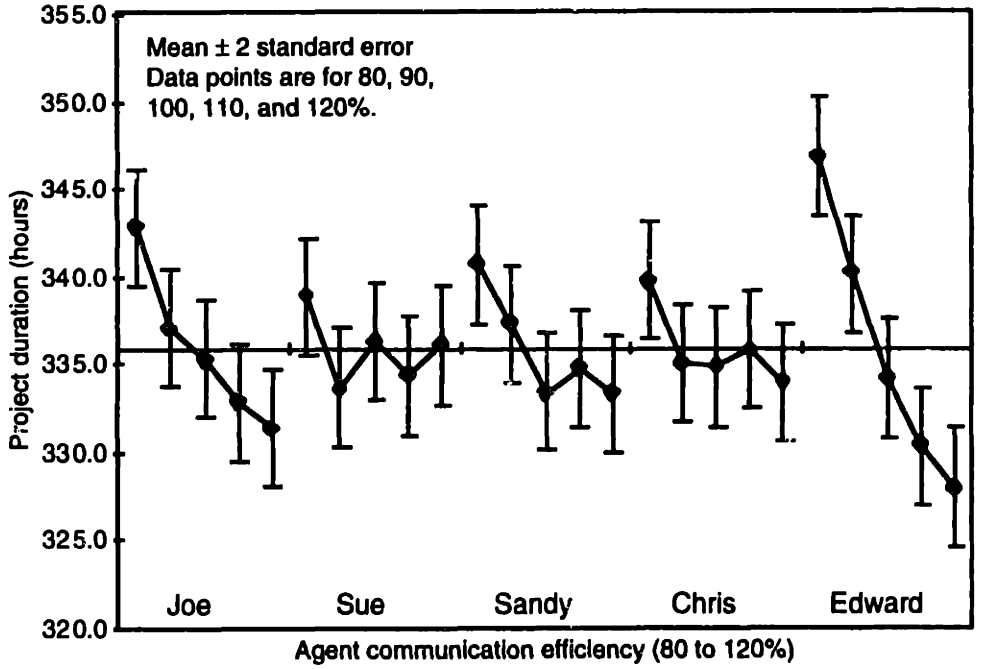


Figure 8.10: Sensitivity of project duration to agent communication efficiency.

Tasks that are permanently on the critical path have no slack.

From the managerial perspective, calculating criticality and slack metrics have a variety of uses. Metrics of task sensitivity indicate areas of the project that need careful scrutiny to ensure that they are accurately estimated and that they stay on track during the course of the project. Non-critical path tasks can be measured for slack to help determine how far off of the critical path they are. Metrics of agent sensitivity can be used to gauge how busy an individual agent is and whether or not a particular agent is the best choice for a job. By rearranging the assignments of agents, not only can the overall project duration be minimized, but also the sensitivity of the overall project duration can be made robust against individual working variations; particularly the kind that come when an individual leaves for a holiday or otherwise disrupts work on a project.

8.2 Interdependency

8.2.1 Concept

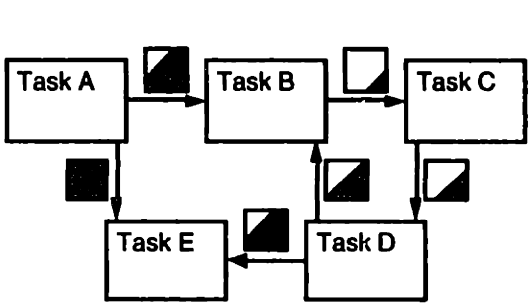
Many extensions of the PERT/CPM method of representing projects allow loops to be specified in the dependency relationships. These loops may involve just two tasks or the entire set of tasks in a project. A natural question to address is how to measure the magnitude of the *interconnectivity* of tasks. Information about loop interconnectivity may be used by managers to decide how to concentrate resources to minimize the effect of long iteration cycles.

Interesting research on interconnectivity has been done using the Design Structure Matrix (DSM). The DSM [74, 75] is a compact method of specifying the relationship between each pair of tasks in a project. The simplest form of DSM is the binary DSM. To construct a binary DSM, place the tasks to be accomplished in a design project into an ordered list. Construct a matrix where the rows and columns are labeled with the tasks in the ordered list. For any two tasks A and B in the project, if task B depends upon task A, place an "X" in the matrix in row A, column B. Similarly, if task A depends upon task B, place an "X" in row B, task A.

The binary DSM is a matrix representation of the ordering of dependencies between different tasks in the project. By rearranging the order of the tasks, the matrix may be placed in a lower triangular form, or close to it if there are loops in the dependencies. The problem with the binary DSM is that it provides no indication of how strongly tasks are coupled in the project, only that a loop exists in the dependencies. A straightforward extension to the binary DSM is to replace each "X" with a number [25, 26, 27]. There are a number of different possible meanings to assign to these numbers (and hence to use when estimating the numbers); they can represent relative importance of dependencies, coupling strength, communication times, parameter sensitivity, etc.

The numerical Design Structure Matrix provides useful clues as to the strength of the loops formed. Under some assumptions, it can be used to directly calculate the nature of the loops and their strengths. Smith [70, 71] models the DSM as an iterative process, where the entries in the matrix represent the amount of percentage of work in a task that needs to be redone after each iteration (the diagonals are set to zero). Starting with a unit vector representing the work remaining in each task (100%), the work remaining after each iteration is given by the product of the DSM and the work vector. The eigenvalues and eigenvectors of the numerical DSM then represent the speed of the convergence and the tasks that are a part of each converging set.

To evaluate interdependencies in the *DiFS* model, we use a matrix representation that graphically displays the strengths of the interdependencies between tasks. There are two versions of the matrix representation; the first represents the modeled informational dependencies between tasks and the second represents the experimentally determined influences of tasks on each other. Each has its use.



Dependent \ Driver	1	2	3	4	5
1. Task A	X				
2. Task B	●	X		●	
3. Task C		○	X		
4. Task D			●	X	
5. Task E	●			●	X

Figure 8.11: Sample design structure matrix showing the information dependency matrix.

Table 8.1: Numerical information dependency matrix of figure 8.11.

		Driver A	B	C	D	E
	A	X	0	0	0	0
	B	0.72	X	0	0.50	0
Dependent	C	0	0.12	X	0	0
	D	0	0	0.28	X	0
	E	1.00	0	0	0.72	X

8.2.2 Model Approach

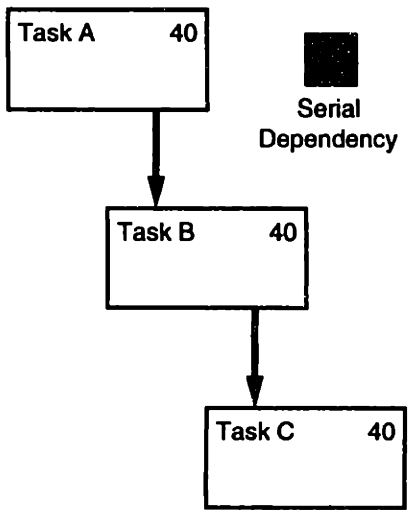
The information dependency matrix derived from the *DiFS* model is similar in form to the Design Structure Matrix. A graphical version of this matrix is generated automatically while creating and editing the model. A sample version of this matrix is shown in figure 8.11. In the information dependency matrix, the value of each cell is determined by the integral of the completion curve of the dependency. The completion curve is normalized on a scale from 0.0 to 1.0, so the value of each cell will range from 0.0 to 1.0. The integral of the completion curve was chosen because it is representative of the dependency. A value of 1.0 occurs only in a completely serial dependency. A value of just under 0.5 occurs if two tasks are tightly coupled to each other. A value of nearly 0.0 represents virtually no coupling or information requirement between the tasks. Table 8.1 shows the calculated numerical values for the information dependency matrix from figure 8.11.

The advantage of representing the informational dependencies in the matrix formulation is that it clearly displays to the experimenter the relationships between the tasks. A fast matrix partitioning algorithm guarantees that graphical displays of the information matrix are correctly sorted whenever a change is made to the dependencies; see section B.3 on page 255.

The disadvantage of the information dependency matrix is that it can only indicate loops in the project model; it cannot make accurate predictions about how these loops will affect project progress. Because many factors play a part in determining the loops and tasks that will most strongly affect the project, an experimental approach is needed to determine the influence of tasks on each other. This approach is described in the next section.

8.2.3 Experimental Approach

To experimentally measure the effect of loops in the simulation, we need to consider what a dependency loop represents. Conceptually it is a collection of tasks that jointly influence each other. Experimentally we can define the influence as follows: Given any two tasks A and B, the influence of TASK A upon TASK B can be described as the ratio of the change in *duration* of TASK B to a change in *size* of TASK A. The idea is that a task influences another task if



		Variation in A	B	C
Duration of	A	1.10	0	0
	B	0	1.10	0
	C	0	0	1.00

Figure 8.12: Influence matrix of three serial tasks.

changing the number of hours in the first task will somehow cause the second task to take longer to finish. Numerically we calculate:

$$I(A,B) = \frac{\partial D_B}{\partial W_A} \tag{8.3}$$

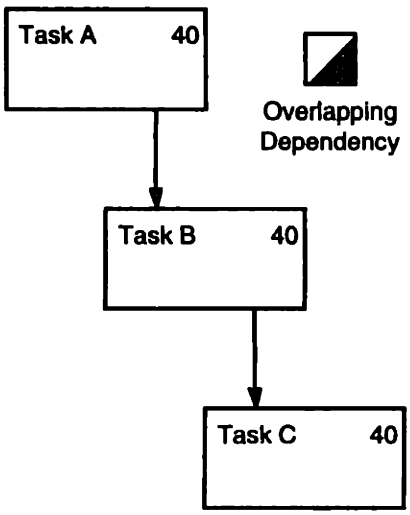
where $I(A,B)$ is the influence of TASK A on TASK B, D_B is the duration of TASK B, and W_A is the work time of TASK A.

Figures 8.12 through 8.17 are a series of sample illustrations of possible dependencies between three tasks and the experimentally determined influence matrix. For each task, a series of 64 trials was run, setting the size of the task to 100%, 150%, 200%, and 250% of nominal size (for both work and communication time). The duration of the task was calculated by subtracting the time of the first reported work in the task from the time of the last reported work in the task (ignoring the hours between 5:00 P.M. and 8:00 A.M.) The influence ratio was calculated by assuming a linear fit between work time and task duration; how to perform this calculation is described in appendix C. Standard error for the calculated numbers is on the order of 5% to 25% of the number itself. Each task has a unique agent assigned full time to it. The large number of trials needed to generate the matrix and the relatively large size changes in each task was necessary because of the difficulty in estimating mean task duration with statistical significance.

Figure 8.12 shows three tasks connected by perfectly serial dependencies; TASK B cannot start until TASK A has completed and TASK C cannot start until TASK B has completed. In this case the influence matrix is almost perfectly diagonal. Modifying the size of TASK A has had no effect on the durations of TASKS B and C. This is expected because although increasing the size of TASK A delays the start of TASK B, it doesn't actually change the amount of time that will have to be spend working on TASK B. Modifying the size of TASK A has a slightly greater than unity effect on the duration of TASK A; this is because there is always some lost time during the work day corresponding to answering phone calls or traveling in the office. Hence adding an hour of work to a task adds slightly more than an hour to the overall duration of the task.

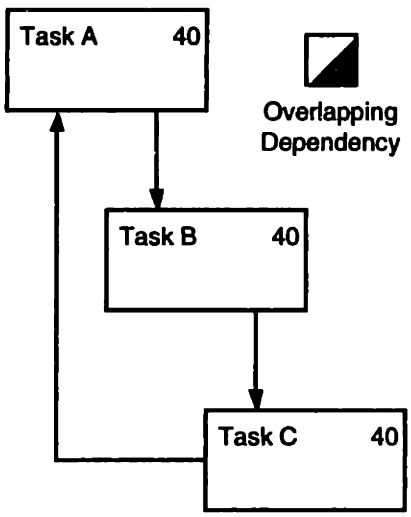
Figure 8.13 is similar to figure 8.12, but in this case the serial connections allow overlap between the tasks. In fact, all three tasks can be started at almost the same time. However, information must continuously flow from TASK A to TASK B and TASK B to TASK C in order to keep work progressing in each task. The overlapping dependencies create a lower-triangular influence matrix where each upstream task has a decaying influence on the downstream tasks.

Figure 8.14 shows what happens to the influence matrix if the three tasks are connected in



		Variation in A	B	C
Duration of	A	1.09	0	0
	B	0.86	1.05	0
	C	0.67	0.88	0.81

Figure 8.13: Influence matrix of three overlapping serial tasks.



		Variation in A	B	C
Duration of	A	0.85	0.69	0.77
	B	0.77	0.87	0.67
	C	0.67	0.78	0.86

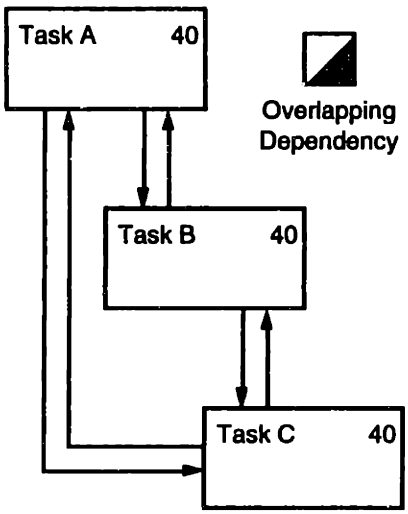
Figure 8.14: Influence matrix of three tasks connected in a loop.

a loop. Two interesting points should be noted: First, the strength of the entries in the matrix are related to how directly the two tasks are connected. TASK A is indirectly driven by TASK B; the value in that entry (0.69) is lower than the direct connection from TASK C (0.77). Second, the strength of a task on itself is less than one. This is possible because a portion of the time spent on a task is waiting for information; increasing the size of the task does not affect how much information is needed, so some of the waiting time may be used profitably where before it might have been wasted.

Figure 8.15 shows what happens when the three tasks are coupled to each other in a double loop, rather than the single loop of figure 8.14. As expected, the entries in the matrix are essentially symmetrical (within the tolerance of the measured error).

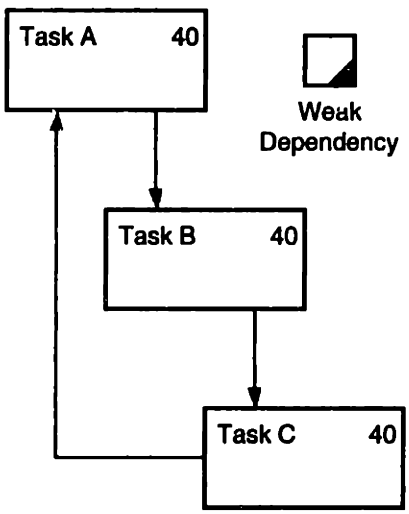
Figure 8.16 is similar to figure 8.14, but the dependencies between the tasks have been loosened. Specifically, each task is allowed to finish up to 50% of its total work before it requires any information at all from another task. Moreover, each task needs no more than 50% of the information generated in the other task (by completion curve). The result is that although the tasks are arranged in a loop, in fact each is only dependent upon the one upstream, not upon *all* of the upstream tasks. This is reflected by zeros in the influence matrix.

Figure 8.17 displays an effect that occurs when tasks are of different sizes. The only difference



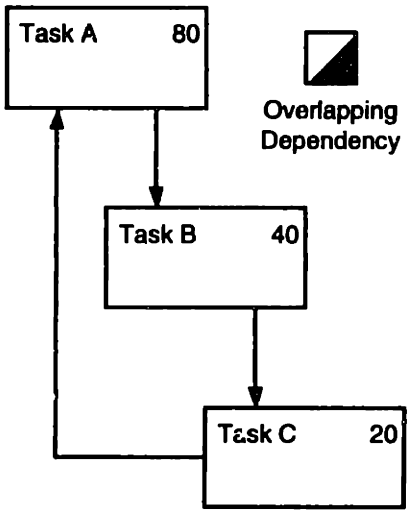
		Variation in A	B	C
Duration of	A	0.89	0.76	0.75
	B	0.78	0.89	0.75
	C	0.77	0.77	0.87

Figure 8.15: Influence matrix of three strongly interdependent tasks.



		Variation in A	B	C
Duration of	A	1.05	0	0.27
	B	0.28	1.05	0
	C	0	0.27	1.05

Figure 8.16: Influence matrix of three weakly connected tasks in a loop.



		Variation in A	B	C
Duration of	A	1.08	0.26	0.10
	B	0.98	0.49	0.10
	C	0.86	0.46	0.16

Figure 8.17: Influence matrix of three tasks of different sizes connected in a loop.

between this figure and figure 8.14 is that the sizes of the tasks are no longer the same. Because of the disparity is size, TASK A is now the limiting factor in the project; the other two tasks spend most of their time waiting for information from TASK A. This is reflected in the influence matrix. Variations in the size of TASK B or TASK C only lightly affect the other tasks, whereas variations in the size of TASK A strongly affect the duration of both TASK B and TASK C. In fact, this matrix is starting to resemble the one in figure 8.13 where there was no feedback loop. The conclusion from this experiment is that although a task may be dependent upon another task, if the information needed from the upstream task is readily available despite size variations, then the upstream task has little influence on the downstream task. In some sense that dependency, although it exists, is not important. Of course, varying the project or rearranging the personnel assignments may make it important.

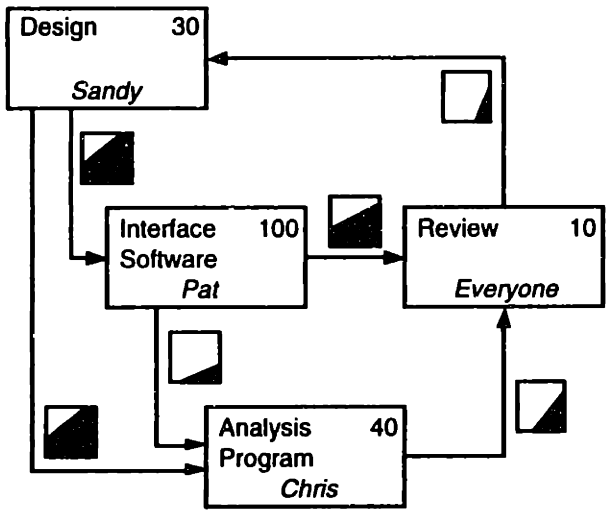
Figure 8.18 shows the calculated influence matrix for the human-factors project of chapter 2 (see figure 2.1 on page 29 and figure 2.9 on page 40). Several of the influence factors are considerably larger than unity. This comes about from two effects. First, the ANALYSIS PROGRAM task is only worked at a 50% rate; hence any increase in duration is effectively doubled. Second, the REVIEW task is a group decision task; all increases in duration come about from scheduling and holding more meetings. Meetings require a group of agents to coordinate their schedules and hence increase the duration of the project disproportionally.

Figure 8.18 illustrates the interconnected nature of a simple four-task project in a compact format. Examining the matrix immediately tells one that variations in the DESIGN task will not greatly affect the duration of the other tasks, whereas a variation in any of the other tasks will have strong ramifications.

Finally, table 8.2 presents a comparison of the information dependency matrix presented in figure 8.11 versus the experimentally determined influence matrix of the same project. The matrices match up well with just a few interesting differences. The influence of task A carries on into tasks C and D. The three tasks that form a loop, B, C, and D mutually influence each other. And interestingly, task A has no influence on task E despite the dependency. It turns out that dependency plays no part in the project; it is completely satisfied well before any work on task E has begun.

8.2.4 Interdependency Summary

This section has presented two separate matrix representations of the interdependencies between two tasks: the information dependency matrix and the influence matrix. The information dependency matrix is numerically calculated from the completion functions of the dependencies



		Variation in Design	Interface Software	Analysis Program	Review
Duration of	Design	0.89	0.97	0.96	0.79
	Interface Software	0.34	1.32	0.64	0.87
	Analysis Program	0.25	0.84	2.42	2.15
	Review	0.13	0.76	1.93	1.85

Figure 8.18: Interdependency of the human-factors project.

Table 8.2: Comparison of information dependency matrix and experimental influence matrix of the project shown in figure 8.11.

Information Dependency	A	B	C	D	E
Task A	X	0	0	0	0
Task B	0.72	X	0	0.50	0
Task C	0	0.12	X	0	0
Task D	0	0	0.28	X	0
Task E	1.00	0	0	0.72	X

Experimental Influence	A	B	C	D	E
Task A	1.20	0	0	0	0
Task B	0.76	0.99	0.40	0.96	0
Task C	0.57	0.41	0.95	0.24	0
Task D	0.19	0.03	0.61	1.15	0
Task E	0	0	0.10	0.67	1.00

between tasks. It represents the strengths of the dependencies between tasks, but does not predict how the influential each dependency will be during the course of the project. The information dependency matrix is closely related to standard applications of the Design Structure Matrix.

The influence matrix is an experimentally determined measure of how variations in task size affect the durations of the tasks in the simulation. Each column of the influence matrix is determined by systematically varying the size of a single task and estimating how that normalized variation affects the duration of each of the tasks in the project, including the varied task itself. The influence matrix has advantages over the information dependency matrix because it accounts for both the propagation of influences through intermediate tasks and the lack of influence of dependencies that exist but have no effect due to other, stronger dependencies.

To the manager, both types of matrices have advantages. The information dependency matrix is quick to calculate and interpret. This ordering helps the manager identify loops in the dependency relationship and how strongly tasks appear to be coupled. As discussed in the DSM literature, knowledge of these loops can be used to select ways of restructuring projects to avoid loops or minimize their impact. The experimentally determined influence matrix takes time to generate, but it has the advantage of directly showing how variations in one task affect other tasks. It is complementary to the concept of criticality of a task; the criticality of a task measures how size variations affect the project duration, whereas the influence matrix measures how size variations will individually affect other tasks. The combination of using both methods allows the manager to not only identify the tasks that will fall on the critical path, but also to evaluate the dependencies that most strongly affect the tasks on the critical path. Expediting information transfer on those dependencies will help prevent time or cost overruns in the project.

8.3 Task Division

8.3.1 Concept

The purpose of this section is to examine the effects of project topology and interdependency on optimal team size. During the course of constructing a large project model, the experimenter is constantly faced with decisions of how to model small pieces of the project. Choices must be made on the basic topological arrangement of tasks, the strengths of the dependencies between those tasks, and the required amount of communication. Often these decisions must be made based solely on anecdotal evidence or engineering judgment. For example, if you are attempting to model the effect of assigning two engineers to a portion of a project that has traditionally had a single engineer assigned to it, then you must decide how that portion of the project is to be divided into tasks that could be allocated between the engineers.

Dividing a project into more tasks and adding more agents will increase the total billing cost of a project, but will not necessarily result in a reduction of project duration. The additional tasks will be related to each other via additional dependencies. Based upon agent role assignments and communication times, these extra dependencies may or may not result in additional required communication. Given sufficiently interdependent tasks and high communication requirements, it is possible that a subdivided project may take longer than the undivided project.

8.3.2 Experiments

I ran a series of experiments to illustrate the effects of project subdivision and team size. Each trial consists of one to four agents working together on a small design project consisting of four interrelated tasks (each of 80 work hours), and an optional fifth group decision task. Each project model is a potential method of subdividing a design task into pieces to be accomplished by a team rather than an individual. With a single assigned agent, the project takes approximately 320 hours to finish. Additional agents may or may not improve upon this total.

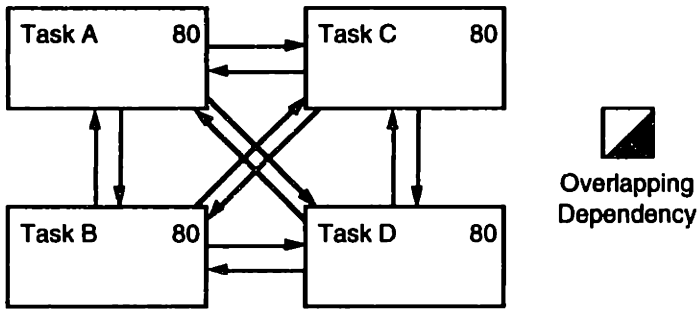


Figure 8.19: Strongly interdependent tasks.

Methodology

Three factors are varied in these experiments: project topology, personnel allocation, and communication time. Four project topologies were considered. Personnel allocation consisted of 1, 2, and 4 agents assigned to the project. In the two agent case, topological variants were considered where relevant. The communication time was varied as a percentage of the work time in each task. All dependencies are set to total scope and thoroughness, so at a communication percentage of 100%, the required communication time through a strong dependency would be exactly equal to the work time of the task.

Two types of results have been recorded: project duration and project billing time. The duration is the length of time that the project runs for. The billing time is the sum of all of the non-idle time logged by the agents assigned to the simulation. Assuming that each agent is billed at the same rate, it represents an estimate of the total cost of the project.

Each project topology variant will be considered in turn.

Strongly Interdependent Tasks

Figure 8.19 is the first sample project. It represents the case where a project can be logically divided into smaller pieces of work that require information exchange in order to coordinate their activities. Note that the dependencies are overlapping; each agent assigned to a task or tasks within this project can only make limited progress before he/she has to gather information from another agent.

Figure 8.20 shows the project duration for this project. Because of the large number of dependencies between the tasks, as the number of agents increase, the effect of the required communication increases quickly. In the four agent case, each agent will have to not only complete his/her own task, but will also have to gather information about three other tasks and discuss his/her own; hence we expect project duration to be roughly $W(1 + 6\lambda)$, where λ is the ratio of communication time to work time. In the two agent case, agents have to gather information about two other tasks and explain their own, and hence the project duration will be roughly $W(2 + 4\lambda)$, whereas for a single agent the project duration will be $4W$.

The rough estimates for project duration suggest that all three types of agent assignments would be equally good when $\lambda = 1/2$. However, figure 8.19 suggests that the unmodeled costs of coordinating the extra agents makes the four agent case unworkable at a communication time of just 35%.

Figure 8.21 shows the project billing time for this project. Not surprisingly, as the communication time required increases, the cost of adding more agents increases rapidly. The added cost of the multiple agents is strongly a function of the communication time.

Group Discussion Interconnection

Figure 8.22 shows a different possible division of a sample project. Here it is assumed that the project can be divided into four relatively independent sections with a common group discussion

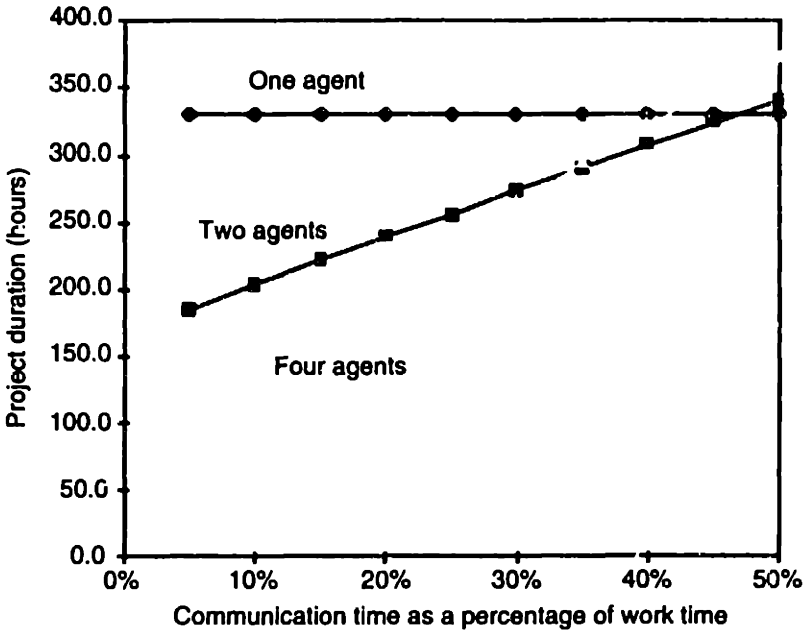


Figure 8.20: Project duration vs. required communication for strongly interdependent tasks of figure 8.19.

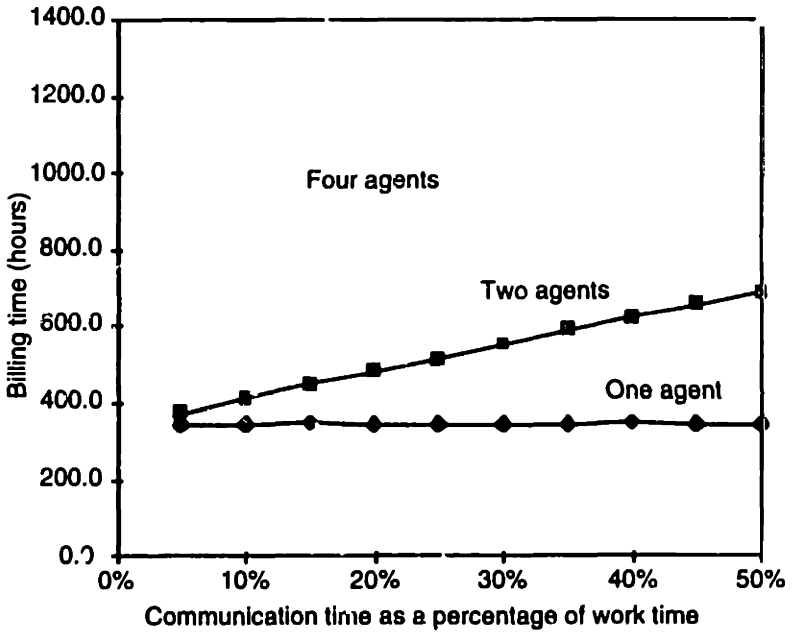


Figure 8.21: Total billed hours for strongly interdependent tasks of figure 8.19.

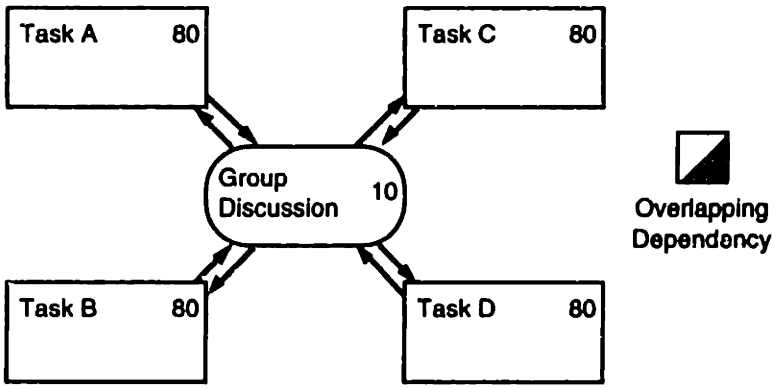


Figure 8.22: Tasks interconnected via a group discussion.

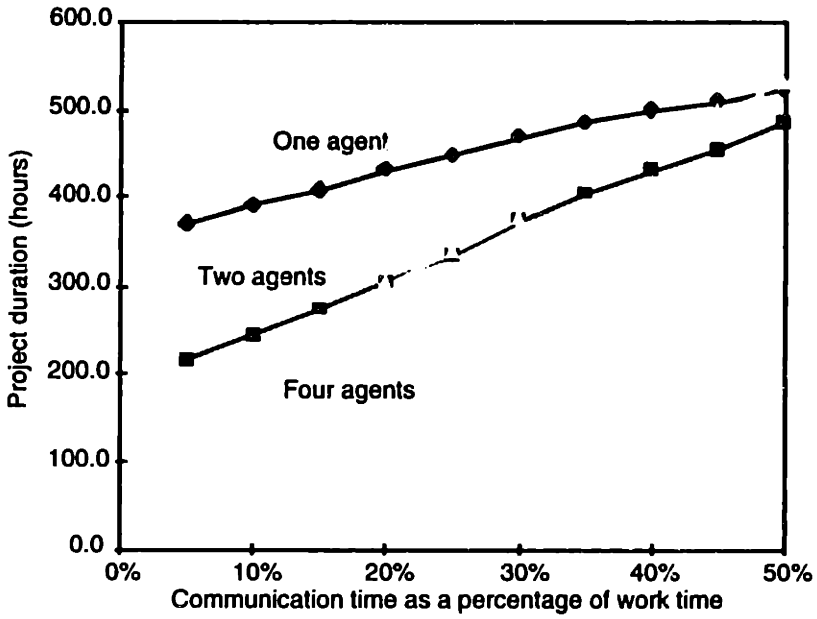


Figure 8.23: Project duration versus required communication for tasks interconnected via a group discussion in figure 8.22.

task to resolve differences. The information sharing between the tasks is assumed to only be that required for the group discussion. One of the agents assigned to the set of tasks is placed in charge of the group discussion. Just to make things more realistic, a manager is assigned to be a participant on the group discussion panel; this means that even in the single agent case, there is a manager who insists of showing up and discussing the problems encountered.

Figure 8.23 shows the project duration for the group discussion project as a function of team size and required communication time. Note that even the single agent assignment shows an increase in project duration as a function of communication time. This is because a manager was assigned to participate in the group discussion.

The project duration comparison between agent assignments on the interconnected discussion task project demonstrates that even though a smaller amount of information sharing is required per agent, the extra requirements of scheduling and holding the group discussion task mitigate that advantage. The billing time shown in figure 8.24 reflects that cost. Four agents is always more costly than two, and yet at a communication requirement of approximately 25% they become less effective.

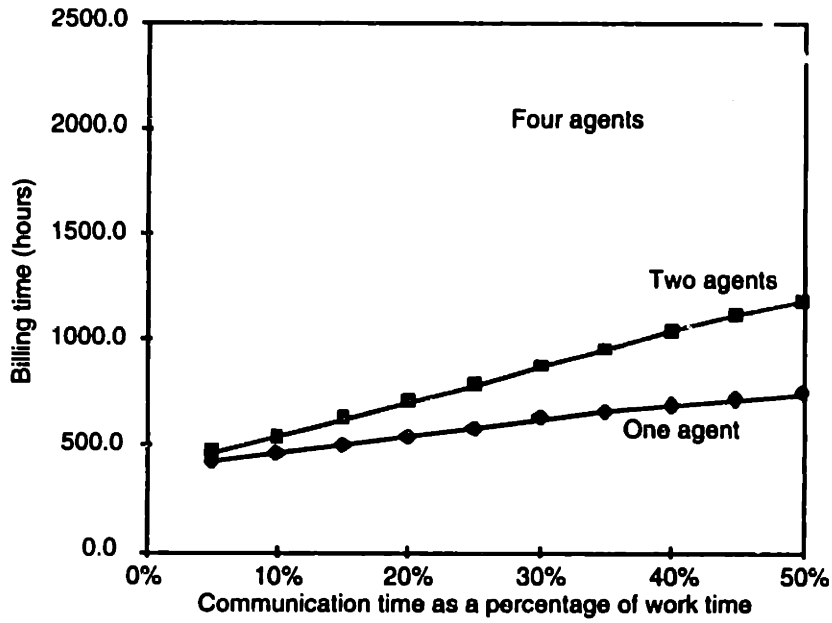


Figure 8.24: Total billed hours for tasks interconnected via a group discussion in figure 8.22.

Waterfall

A common model of the design process is the "waterfall" model of design; a serially connected set of tasks to be accomplished that include small local feedback from the downstream tasks to their immediate predecessor. Figure 8.25 shows a project decomposition using such a task breakdown. The dependencies are strongly serial progressing down the waterfall, and the dependencies going up represent small quantities of feedback.

Figure 8.26 shows project duration results for the waterfall project model with one, two, or four assigned agents. Note that there are three logically distinct cases of two assigned agents; the two shown here consist of an alternating case where the assigned agent changes with each task in the series and an ordered case where one agent takes the first two tasks in the series.

The waterfall project model demonstrates that the difference between alternating and ordered agents is a function of the required communication time. Below about 20% communication time, the alternating agents have the advantage in project duration; here the two agents can effectively overlap their efforts. Above this point and up to about a 55% communication time the ordered agents have the advantage; the savings in communication time outweighs the disadvantage in not allowing the tasks to overlap. Note that in both cases the billing cost of alternating agents is always higher than the ordered agents (see figure 8.27).

Feedback Loop

Figure 8.28 shows a simple serial connection of tasks with one long feedback loop representing modifications that must be made after the bulk of the work has been completed. This type of project model is appropriate for projects that have a well-understood series of stages and some potential feedback to correct earlier errors.

Figure 8.29 shows project durations for different agent assignments to the feedback loop project. As in the waterfall project model, there are three logical possible assignments of two agents. Two of those are represented here; an alternating assignment and an ordered assignment where one agent takes the first two tasks. Also as in the waterfall project model, alternating the two agents provides a marginal project duration advantage over not alternating them for low communication times. At larger communication times the cost of the extra information transfer required outweighs the advantages of allowing the tasks to overlap. Figure 8.30 shows that

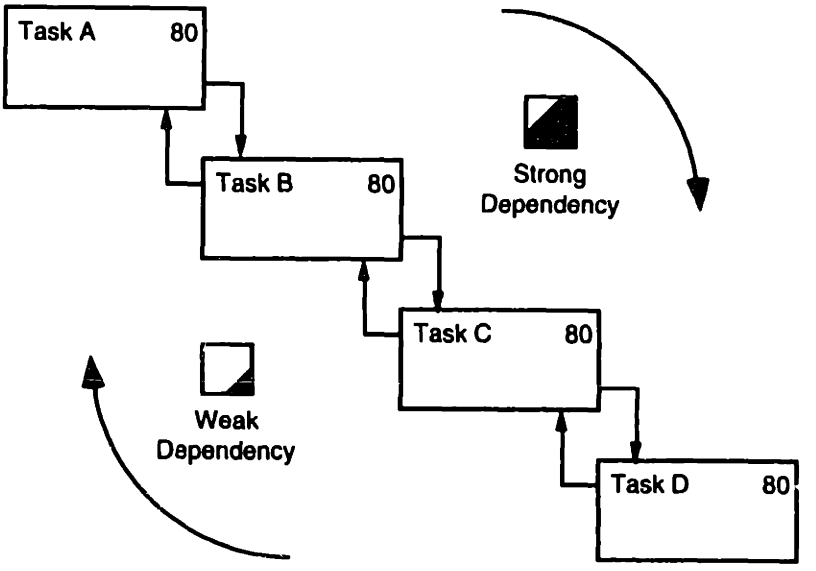


Figure 8.25: Waterfall tasks.

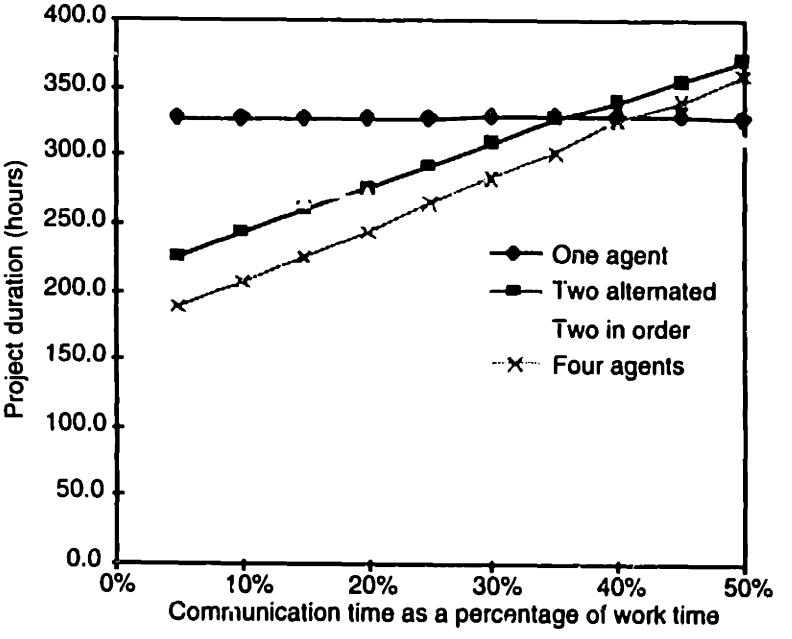


Figure 8.26: Project duration versus required communication for waterfall tasks of figure 8.25. For the two alternating agents case, one agent was assigned to A and C, and the other to B and D. For the two ordered agents case, one agent was assigned to A and B, and the other to C and D.

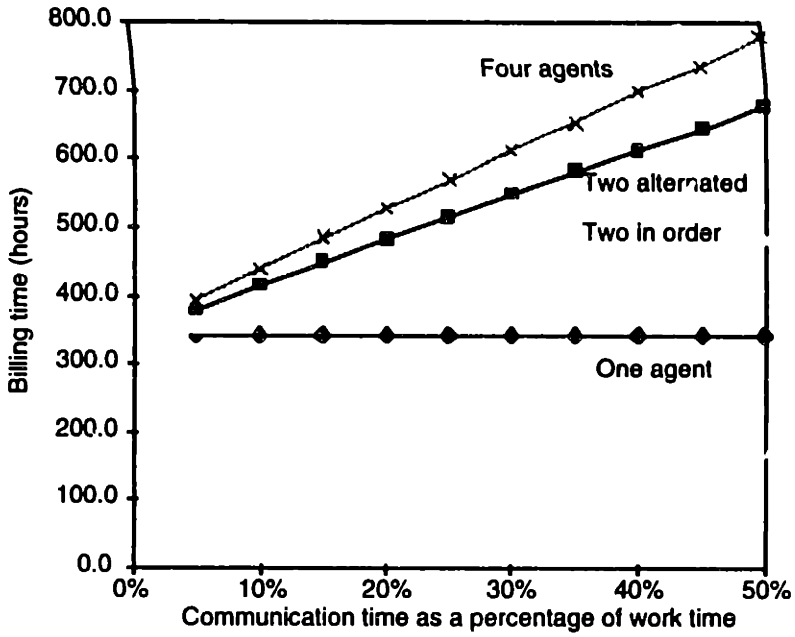


Figure 8.27: Total billed hours for waterfall tasks of figure 8.25. For the two alternating agents case, one agent was assigned to A and C, and the other to B and D. For the two ordered agents case, one agent was assigned to A and B, and the other to C and D.

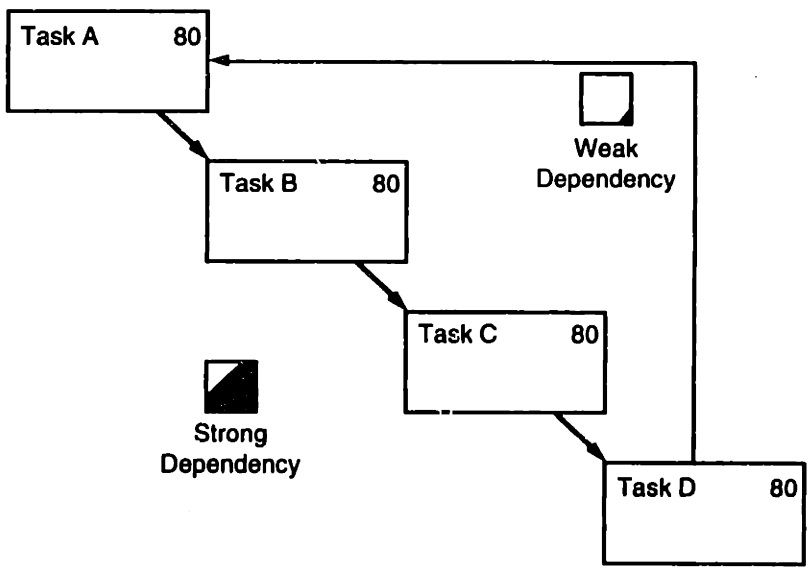


Figure 8.28: Feedback loop tasks.

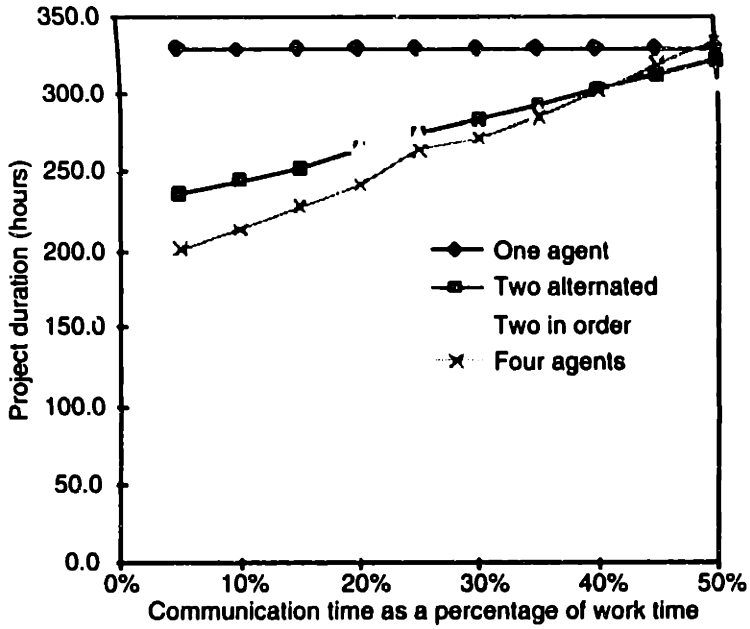


Figure 8.29: Project duration versus required communication for feedback loop tasks of figure 8.28. For the two alternating agents case, one agent was assigned to A and C, and the other to B and D. For the two ordered agents case, one agent was assigned to A and B, and the other to C and D.

billing costs are directly proportional to communication time and that alternating two agents is always more expensive than ordering them.

8.3.3 Task Division Summary

This section has presented a number of project models that represent possible ways to decompose a small design project or task into subtasks. This decomposition allows a team of engineers to be assigned to the project. However, the quantity of communication required between the team members strongly influences how efficient the team will be. In all cases, adding extra engineers increases the billing cost of the project, but may decrease the project duration.

For all possible task decompositions, at low required communication times it is advantageous to subdivide the task across many agents. As the communication requirements increase, the advantage of overlapping activities is mitigated by the cost of the extra communication. This cost is particularly strong when team members must hold group discussion sessions to resolve task differences; in this case the extra cost of scheduling and holding meetings quickly removes all advantages of dividing the work over many agents.

From a managerial perspective, there are a number of factors to be considered. First, when modeling a project and deciding how many individuals can be usefully applied to it, the examples in this section demonstrate that the topology of the subdivided task is quite important. Care needs to be taken in selecting the best possible task subdivision for the project at hand in order to generate relevant results. Conversely, if different methods of dividing a project into tasks are available, then a careful application of simulation may determine the division with the best properties.

A second managerial issue is that the communication time required in a small set of tasks drives how efficiently a group can perform those tasks. Throughout all of these examples, the cost of putting more than one agent on a project increased dramatically with the quantity of communication required. Minimizing these communication requirements by careful task divisions and by ensuring an efficient exchange between agents is important as the required amount of communication increases.

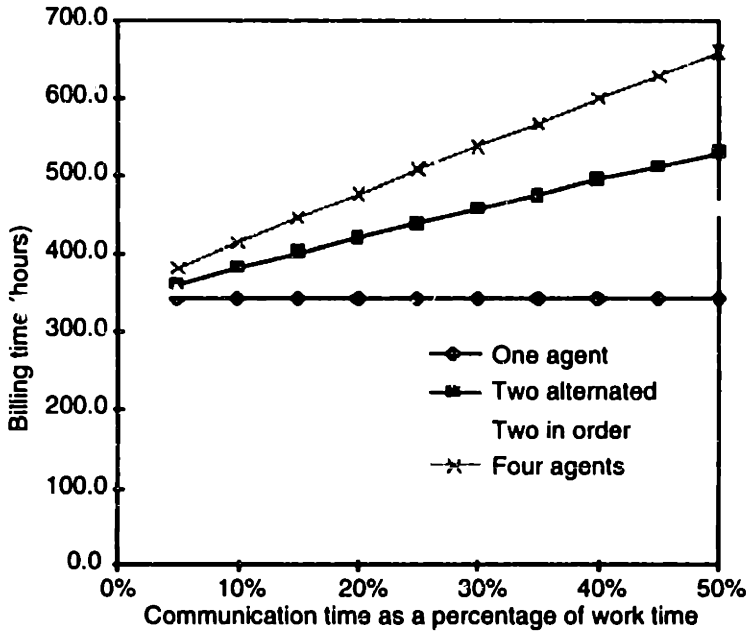


Figure 8.30: Total billed hours for feedback loop tasks of figure 8.28. For the two alternating agents case, one agent was assigned to A and C, and the other to B and D. For the two ordered agents case, one agent was assigned to A and B, and the other to C and D.

The third managerial factor is that the relationship between project duration and billing time can be exploited in CPM tradeoffs. The Critical Path Method [83] traditionally trades off the cost of reducing a task duration with the monetary advantage of finishing a task early. For a given cost model, a manager can use the simulation and task subdivisions to examine the advantages of adding extra agents to projects.

8.4 Office Environment

8.4.1 Concept

The *DiFS* simulation provides great control over the way that the office environment, projects, and personalities are modeled. Section 8.3 dealt with how team size relates to project topology and required communication times. This section contains a discussion of how communication channels and agent behavior interact with project topology.

Allen's book [5] on information transfer in research organizations contains a discussion of how the interactions between individuals decrease as the distance between them increases. It is expected that a team of engineers who must work together will work less effectively as the distance between the team members increases. However, it is also possible that in tight quarters, the effectiveness of the group will drop. Shaw's book [67] on the psychology of group dynamics discusses the potential advantages and disadvantages of physical proximity.

Team performance should improve when team members are co-located and exchange information readily. The purpose of this section is to demonstrate how the *DiFS* simulation may be used to examine how group performance varies depending on the relative locations of the team members, their behavior, and the required information transfers.

8.4.2 Experiments

I ran a series of experiments to illustrate the effects of team co-location and behavior. Each trial consists of four agents working together on a small design project consisting of four interrelated

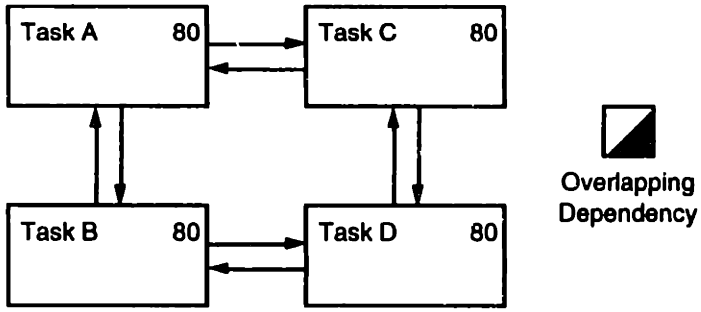


Figure 8.31: Strongly coupled tasks.

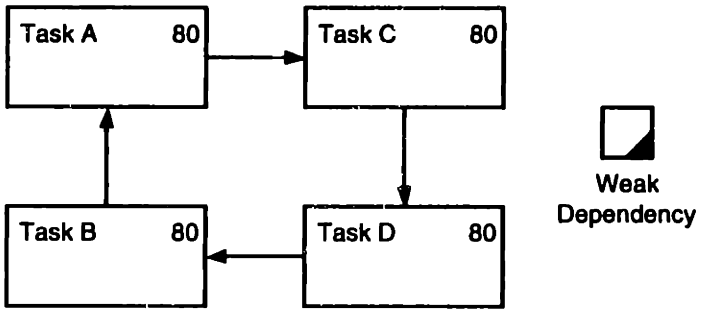


Figure 8.32: Weakly coupled tasks.

tasks (each of 80 work hours), and an optional fifth group decision task. If a single agent was assigned to all of the tasks in a single project, it would take that agent approximately 320 working hours to finish the tasks; having four agents assigned to the tasks may or may not improve upon this total.

Methodology

Four factors are varied throughout the experiment: project type, office arrangement (including communication channels), agent behavior, and amount of required communication.

Three topologically distinct project types are considered: strong dependencies, weak dependencies, and coupling through a group decision task (figures 8.31, 8.32, and 8.33). The different project topologies dramatically affect the amount of coordination between agents. This increase in coordination will increase the project duration, but it should also be influenced by the location of the agents and the communication methods.

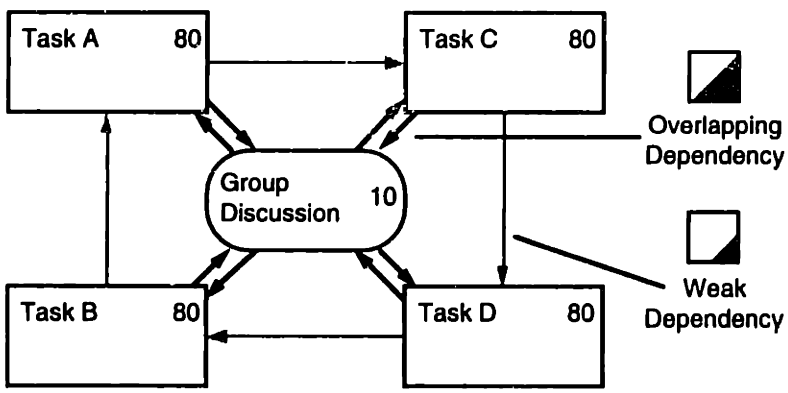


Figure 8.33: Tasks coupled through a group decision task.

Table 8.3: Personality factor settings

	Social	Antisocial
Request spacing	2 hours	4 hours
Request wait time	16 hours	24 hours
Gossip limit	10 minutes	5 minutes
Talking bonus	2.25	2.00
Talking loss	0.10 (per min)	0.2 (per min)
Contact delay	30 minutes	60 minutes
Transmit delay	15 minutes	30 minutes
Update status	1 day	2 days
Start talking bonus	1.8	1.0
Request meeting proximity	1.5	2.0
Answer meeting proximity	1.5	2.0

The office arrangement is varied between three logical extremes: co-located agents, agents in adjacent offices, and agents that telecommute. In the co-located case, all four agents are assigned to the same office. I think of this case as having agents in open, adjacent cubicles: each can hear whatever one of the others is saying. In the office case, all four agents are assigned to offices that are just far enough apart that it takes a minute or two to travel between them. For the telecommuting case, I replaced all methods of communication with videoconferencing equipment. The videoconferencing equipment is 5% less effective than a face-to-face conversation (in terms of speed), but allows multiple agents to participate in a discussion. Because the agents have no other communication channels available, they end up spending all of their time in their offices.

The three office arrangements provide three types of interactions between the agents. In the cubicle arrangement, agents are free to instantly start or interrupt a conversation at any time. In the office arrangement, agents tend to travel from office to office, and hence they frequently are not located in their own offices (and are therefore unavailable for phone calls). In the telecommuting arrangement, agents communicate almost as effectively as the office arrangement, but they may be unavailable for discussion because their videophone is tied up. Hence we would expect that information would flow most freely in the cubicle arrangement and less freely in the office or telecommuting arrangement.

The agent behavior was varied between two types of behavior, affectionately referred to as "social" and "antisocial". The social behavior is just the default settings of the *DiFS* simulation. The antisocial behavior was created by modifying 11 of the personality factors of agent behavior that deal with communication, as shown in table 8.3. Communication factors were increased for the antisocial agent. For example, agents wait longer between requesting information (*Request spacing* = 4 hours), tolerate less chit-chat in a conversation (*Gossip limit* = 5 minutes), and wait longer between voluntarily providing task status information to other agents (*Update status* = 2 days). The remaining behavior parameters deal with how desirable agents view conversation (as opposed to work). Refer to appendix B for a description of the algorithms.

The important fact to note about the antisocial agent behavior is that it represents a variation on the standard behavior that does not engage in conversation quite as readily or as frequently; hence the name antisocial. It is a *relatively* antisocial behavior compared to the normal behavior. Because the normal social behavior was derived by optimizing the behavior factors, we expect that the antisocial behavior will not perform as effectively.

The final factor varied during the course of the experiments is the communication time of the tasks that make up the project. For each experiment the communication time of a given task is set to a percentage of the task's work time. Each dependency between tasks has been set to total scope and thoroughness. Hence for a strong dependency, the agents will have to communicate all of the information generated in one task to the other. However, there are no strong dependencies in any of the three projects used in this experiment; in practice the

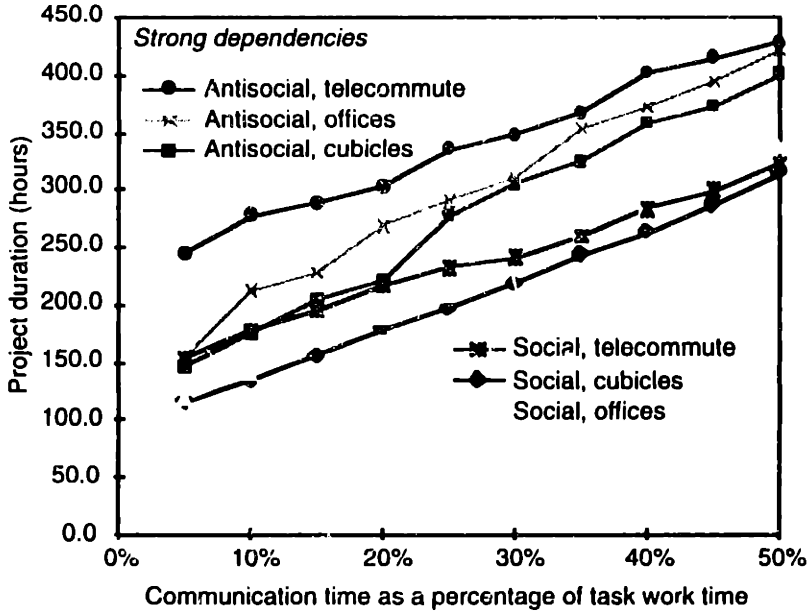


Figure 8.34: Project duration versus communication time by agent behavior and location in the strong dependencies project of figure 8.31.

amount of information communicated for a given task will be approximately 80% of the varied percentage of the task work time.

Trials

A total of 900 trials were run (3 project topologies, 2 personalities, 3 office layouts, 10 levels of communication time, and 5 repetitions per level). We will consider the results by project topology, by agent behavior, and by office layout.

Trial results by project topology are shown in figures 8.34, 8.35, and 8.36. Several effects can be noted: social behaviors perform better in project topologies that require information coordination, and agent location is always a factor. The project that requires the greatest voluntary coordination between agents (the strong dependency project) shows the greatest spread in performance between social and antisocial agents. In all cases the antisocial agents are at a significant disadvantage. This distinction is non-existent in the weak dependency project and has little effect in the project with the group decision task (and hence enforced meetings). Agent location also plays a factor; telecommuting agents are at a disadvantage. Interestingly, the magnitude of this disadvantage decreases as the amount of communication required increases.

Trial results by agent behavior are shown in figures 8.37 and 8.38. For these plots the effects of agent location have been collapsed by project to emphasize the effect of the project topology (they do not vary significantly on the scale of the plots). Two effects are interesting to note: first, that the group decision project takes significantly longer than the strong dependency project; more so than the extra amount of communication and work required entails. The coordination of agents to attend regular meetings slows the project progress. The second interesting effect is that antisocial agents do much worse on the strong dependency project than social agents. These illustrations highlight the result that antisocial agents perform acceptably on projects with either weak interactions or coordinating mechanisms like a group decision task, but they do not do well when required to exchange information amongst themselves.

Experimental results by office locations are shown in figures 8.39, 8.40, and 8.41. These graphs show an interesting marginal effect; as the distance between agents increases, the effect of social vs. antisocial behavior also increases. Particularly, as the decision project is moved

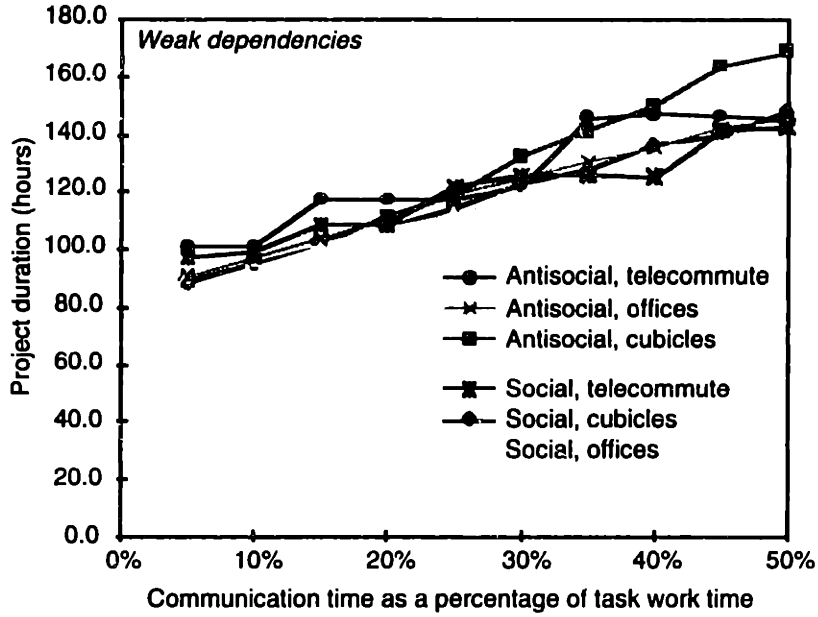


Figure 8.35: Project duration versus communication time by agent behavior and location in the weak dependencies project of figure 8.32.

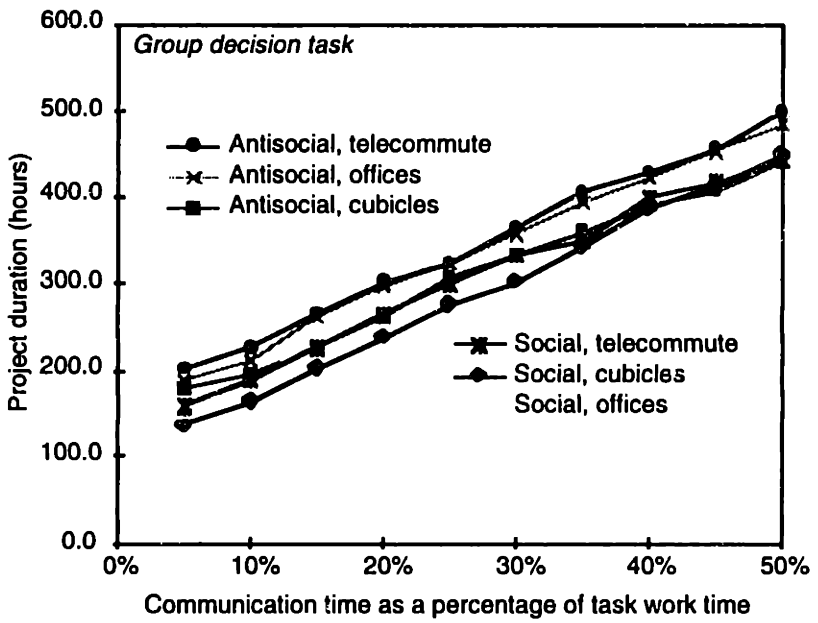


Figure 8.36: Project duration versus communication time by agent behavior and location in the group-decision project of figure 8.33.

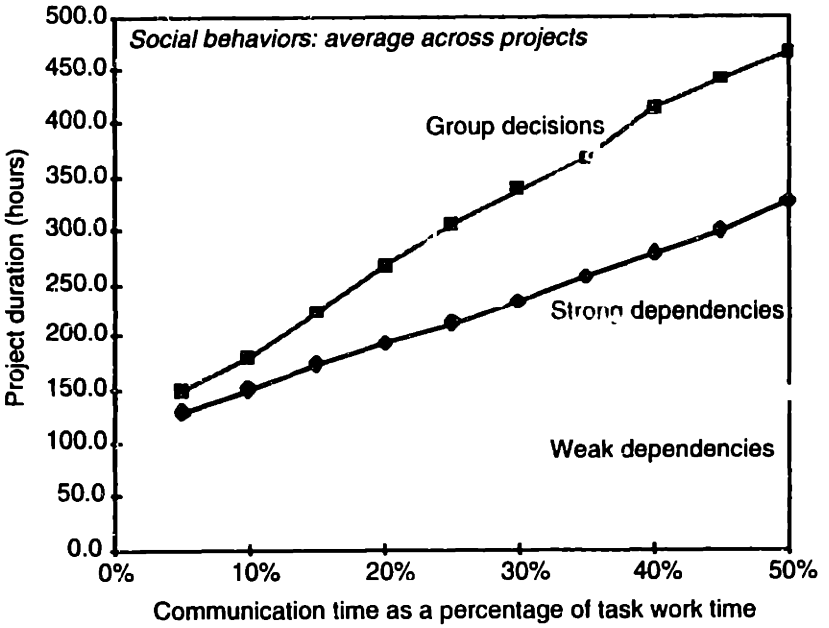


Figure 8.37: Project duration versus communication time by project type using social agents.

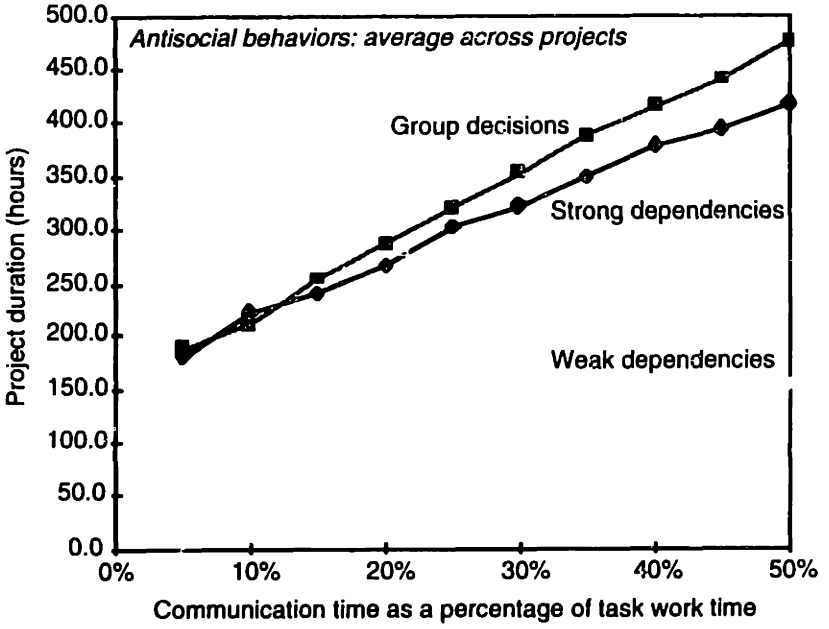


Figure 8.38: Project duration versus communication time by project type using antisocial agents.

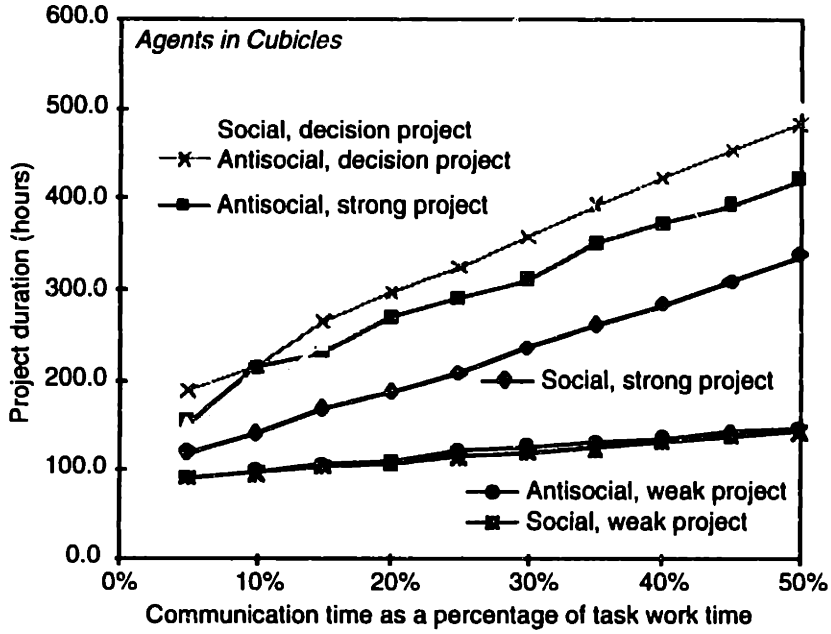


Figure 8.39: Project duration versus communication time by agent behavior and project type using agents located in adjacent cubicles.

from a cubicle arrangement to a telecommuting arrangement, the spread between social and antisocial behavior performance widens.

The strongest second-order effect displayed by the combination of personal behavior, agent location, and task type is that antisocial agents assigned to the strong dependency project perform poorly. This poor performance is a result of their unwillingness to exchange information. Hence it is possible that enforcing more agent interaction may improve their project performance.

Figure 8.42 shows the result of requiring the telecommuting antisocial agents to hold a one-hour daily team meeting. The enforced morning meeting significantly lowers the project duration, although it does not lower it to the level achieved by the social agents. Enforcing a team meeting on agents who are already more closely located (in offices or cubicles) did not have this effect. The likely cause of the improvement in the telecommuting agents is that normally telecommuting agents are limited to conversing with exactly one other agent at a time (they have no way of breaking into an established conversation). But the scheduled team meeting involved all four agents and hence some information of common interest could be exchanged at lower cost than sending individually.

8.4.3 Office Environment Summary

This section has presented a series of experiments on the effects of agent location, agent behavior, and project topology. General results are that project interconnectivity drives duration, antisocial agents perform less effectively than social agents, and team co-location with equivalent communication methods marginally affects team performance.

An interesting second-order effect is that antisocial agents perform particularly ineffectually in strongly coupled projects that do not have an enforced information exchange mechanism such as a group decision task. The strength of this effect can be mitigated by mandating daily team meetings for telecommuting agents. The daily meetings allow the agents to share information with all other agents simultaneously; this significantly reduces the overall project duration.

From the managerial perspective, this section has merely hinted at the possibilities of the *DiFS* simulation. The simulation clearly displays the effects of co-locating agents or altering

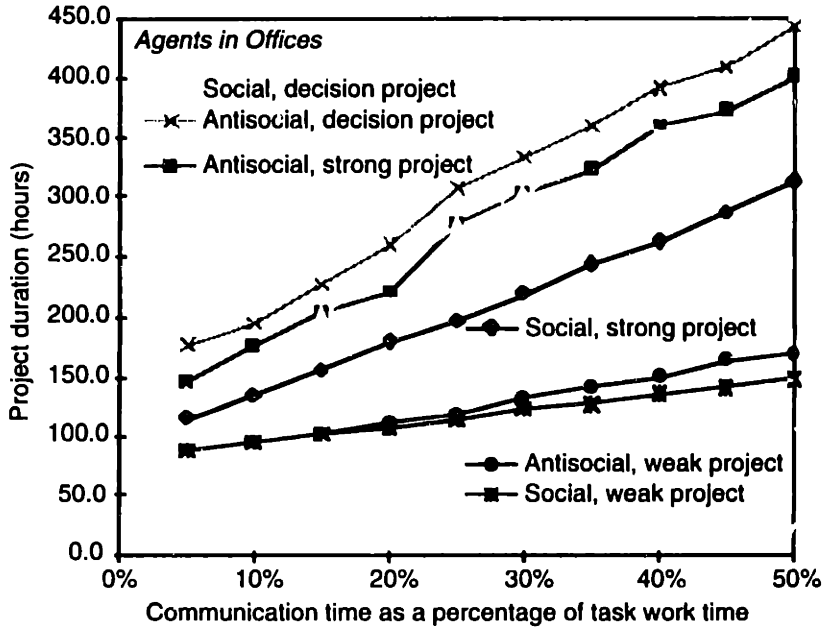


Figure 8.40: Project duration versus communication time by agent behavior and project type using agents located in nearby offices.

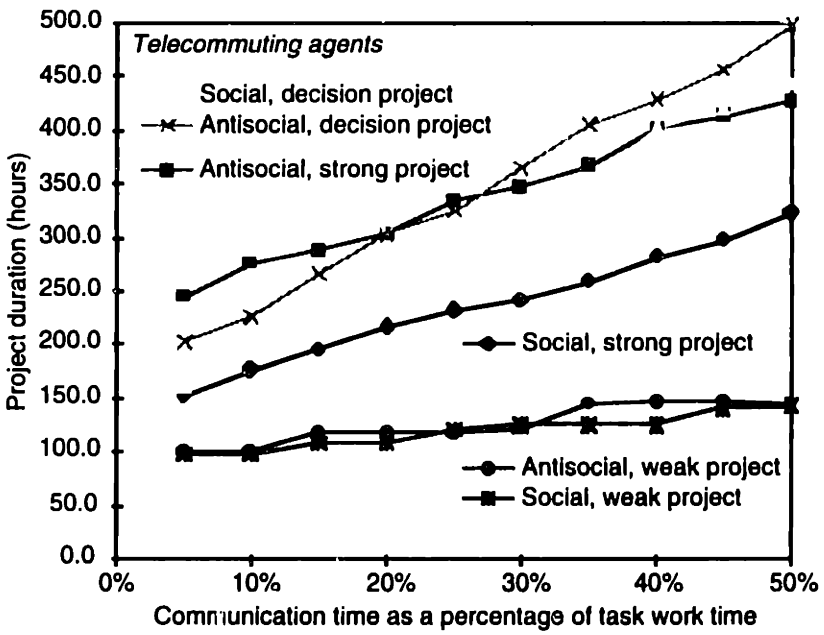


Figure 8.41: Project duration versus communication time by agent behavior and project type using agents that telecommute.

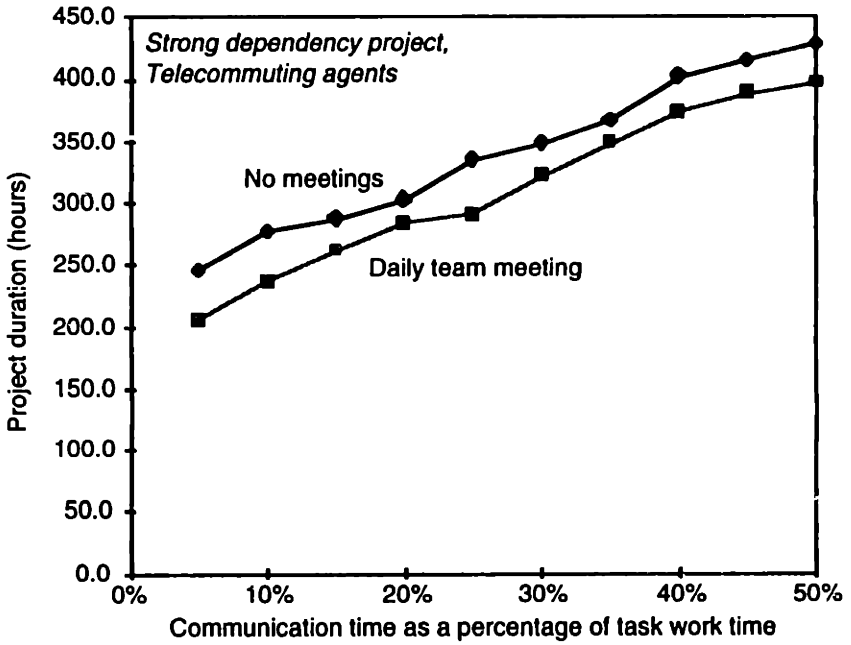


Figure 8.42: Effect of scheduling daily meetings for telecommuting agents on the strong dependency project.

their personal behaviors. For larger projects, this could be used by a manager to examine potential effects of office space allocation and new communication technology. However, the *DiFS* simulation is still somewhat limited in the nature of interpersonal interactions and the advantages of co-location of team members; simulation enhancements in this area would improve its potential utility.

8.5 Summary

This chapter has presented four different methods of analyzing an engineering project using the *DiFS* simulation. The purpose of this chapter was to demonstrate how the analysis of a project can identify potential trouble spots and suggest ways of improving project performance.

The first section dealt with *criticality*; the concept that the performance on a particular task or the efficiency of a particular agent may directly influence the project performance. The criticality of a task or agent was defined as the sensitivity of the project duration to variations in task or agent parameters. Four types of criticality were discussed: sensitivity to errors in work-time estimates, sensitivity to errors in communication-time estimates, sensitivity to agent working efficiency, and sensitivity to agent communication efficiency. The utility of the criticality analysis to a manager is that tasks or agents on the critical path can be identified, allowing their effects to be mitigated by careful management or reassignment.

The second section dealt with *interdependency*; the concept that groups of tasks mutually influence each other through loops in the dependency relationships. Loops may be located in two ways: visual inspection or an experimental sensitivity analysis of how tasks affect each other. The experimental measurement has two advantages: first, it ignores dependencies that exist but are always satisfied due to arrangements of project topology or agent assignment. Second, the effects of dependency propagation through other dependencies are recorded. The utility of interdependency analysis to a manager is that the strong loops within the project model can be identified, allowing the manager to rearrange the project or team assignments to break long dependency loops.

The third section dealt with *task division*; or the relationship between team size and project

performance. Using small projects as examples, a variety of team sizes and project topologies were explored with varying communication requirements. As communication requirements increase the benefits of dividing tasks into many subtasks decrease. The effectiveness of task subdivision is also a strong function of the topology of the task relationships; tightly connected tasks with or without group discussions do not benefit from task subdivision. For management purposes, this simple exploration of the effects of project topology, communication times, and personnel assignments serves as an example of how these factors can be critical in determining team performance. This information can be used to help decide how to break down large projects into smaller pieces and how to assign appropriate team sizes. Future applications of this methodology should allow a manager to automatically assign project teams.

The fourth section dealt with the effect of the *office environment*; or how agent personality, office location, and communication technology affect project performance. The purpose of this section was to demonstrate how an experiment could be conducted that varied agent behavior between normal social behavior and antisocial behavior, and varied office assignments from cubicle-based, office-based, or telecommuting agents. This sample experiment demonstrated the importance of good information coordination between agents and how proximity strongly affects project performance. In an extreme case of antisocial telecommuting agents, it demonstrated that enforcing regular team meetings could be used to improve project performance. For management purposes, the methodology of this section could be used to experiment with office assignments for personnel and new technologies for acquiring information.

Chapter 9

Conclusions

9.1 Summary

This thesis addressed the problem of developing methods to better understand the group design process. The approach selected was to model the design process as a collection of interdependent tasks and then simulate the behavior of engineers and managers working on the tasks.

The design process has been modeled as an augmented precedence network similar to PERT or CPM techniques. Individual tasks represent work to be done by individuals or by teams of engineers in a group discussion. Dependencies between the individual tasks represent a need for some of the information generated by the upstream task. Parameters within a dependency specify what portion of that information is needed and when it is needed by. By carefully specifying the nature and strengths of the dependencies between tasks, it is possible to model simple forms of design iteration and rework. The information-flow model of a design project is a description of the work that needs to be done in the project and in what order it needs to be done.

The *DiFS* computer simulation of engineers working on the design project was created in order to make predictions based on the information-flow model. Engineers assigned to the design project are simulated by computer agents. The computer agents work within a simulated office environment. They send messages to each other through experimenter-defined synchronous and asynchronous communication channels. They schedule and hold meetings, work on their assigned tasks, and attempt to finish the design project as quickly as possible.

The model of agent behavior used in the simulation is based upon a two-stage model of human reasoning. High-level reasoning processes within the agent convert knowledge of tasks, contacts, the local environment, planned activities, and recent events into a current objective. Agent scripts translate the objectives into specific actions to be taken. The high-level reasoning process selects the current objective based on a limited rationality approach that evaluates the perceived utility of the possible objectives available to the agent and selects the most promising one.

Validation studies of the model and simulation were conducted. The preliminary studies (1) demonstrated that the simulation is stable and (2) were used to improve agent performance. The final validation study was conducted in conjunction with a local design company. An information-flow model of one of their design projects was constructed based on discussions with the project team, generated documentation, and timesheet data. Simulation results from running the project model matched well with the corporate experience in the actual project.

The final portion of the thesis dealt with how the design simulation could be used to assist managers of design teams. Experimental methods were developed to examine the sensitivity of design projects to variations in task size and individual performance. These sensitivity studies isolate the tasks and people who fall on a project's critical path. Further methodologies were developed to identify important interdependencies in the design project, aid in determining

optimal team size, and examine the effects of team co-location and communication methods.

9.2 Future Work

The information-flow model and *DiFS* simulation has shown great promise as a method of exploring the engineering design process. This section discusses a collection of potential simulation and model enhancements. It is divided into two parts: simulation extensions that would be relatively easy to incorporate and model extensions that would require significant amounts of work.

9.2.1 Simulation Extensions

Many simple extensions to the *DiFS* simulation have been proposed. Some of those have found their way into the current software; many were placed on the queue of good ideas to be added at a later date. The following is a list of the most useful sorted roughly in order of their relative importance.

1. Assign suggested start and stop dates to each task. Agents within the simulation would attempt to meet these suggested dates by concentrating their efforts on tasks that appear to be falling behind schedule. This modification would almost certainly result in agents that work past 5:00 P.M. on particularly late tasks. Hence the project duration would no longer necessarily be a useful measure of project performance. Measures of agent tiredness and how easily each deadline was met would be substituted.

2. Create vacation schedules for agents. A few managers have expressed an interest in being able to predict what would happen to an engineering project when personnel take vacations in the middle of the project. This modification would require agents to behave intelligently before another agent went on vacation to gather information and plan activities around the missing agent.

3. Specify personal relationships between agents. These personal relationships may be statically assigned by the experimenter or dynamically develop during the course of the simulation (to simulate effects such as the development of teams). The personal relationships would affect how frequently agents communicated with each other and how effective that communication would be.

4. Specify the skills required to work on a particular task. Each agent in the simulation would have a list of skills. Each task would specify how important each skill is and how it affects the task duration. By specifying required skills, the simulation could be used in a resource scheduling program that would attempt to optimally allocate team members to tasks.

5. Add a lunch room and other locations within the simulation that agents could travel to. Agent behaviors would be modified so that they would be inclined to travel around the office more than they currently do. This would affect the availability of the agents. A lunch room would serve the additional use of bringing agents together for the opportunity of casual conversation and information exchange.

6. Create a type of work task that corresponds to dealing with outside vendors. A problem with the current simulation is that there is no good way to represent interactions with outside vendors or work on any type of task that inherently possesses a time lag. A simple way to implement a vendor task is to specify the maximum amount of work that can be done on a task in a single day.

7. Add a few message types corresponding to exhortations from other agents. Currently an agent's priorities are affected by how many outstanding requests for information have been received from other agents. Messages directly corresponding to requests for getting projects done by certain dates are a more direct way of incorporating outside agent influence.

8. Make the meeting scheduling mechanism more flexible. Allow meetings to be called on shorter notice either by letting agents schedule meetings outside of the global meeting schedule, or by allowing agents in close proximity to schedule meetings on shorter notice than a single day.

9. Make small, frequent information transfers more costly than large, infrequent transfers. For example, the total required communication time for a task may be larger if information is transferred while the task is progressing rather than all information being transferred at the end. This could be implemented by making recently generated information expensive to exchange.

All of these model enhancements represent simple modifications to the *DiFS* simulation and would require extensions to the agent behavior model.

9.2.2 Model Extensions

This section discusses potential extensions to the information-flow model. These extensions change or augment fundamental assumptions made about the design process model and hence would require significant work to incorporate in the simulation software.

Agent Roles

The current list of roles that agents can be assigned to on a task is limited. There are many interesting roles that could be added to improve the validity of the model and simplify the modeling process. The following is a list of potential additions:

Reviewer: A person who reviews the progress in a task and criticizes it. Before a task could be considered complete, an agent would need to get the approval from each reviewer assigned to it. This could be done by a joint meeting of all reviewers or by individual contacts.

Consultant: A person who must be regularly talked with during the course of the task for advice on how to proceed. Effectively this could be implemented by requiring one consultation every X hours of task work.

Manager: A person with higher authority on the task. This person would play the roles of reviewer and consultant with the additional role of exception handling (discussed later in this section).

Part-time participant: A person who can participate in group discussions, but whose participation is not required. Participation could be required for at least a certain percentage of the work done by the group, or task work efficiency could vary based on how many participants are present.

Liaison: A person who connects two teams or groups via a common task by acting as the contact person. That is, individuals outside of the immediate group would contact this person for information about a task, rather than directly contacting the person responsible for the task.

Adding any of these roles would require substantial changes in the simulation, agent behavior, and extra parameters to be set by the experimenter. For example, adding the "consultant" role requires specifying not only how many hours must be spent in consultation, but also requires that

agents keep track of how many hours have been spent. Additionally, how does the simulation decide that two agents have been in consultation? A reasonable approach is to assume that they must spend time in a common conversation and exchange some form of message that specifies they have been consulting on a particular task (analogous to the way group decision work is accomplished).

Management Exception Handling

The current simulation does not implement all types of standard management behavior. Managers in the *DiFS* simulation do not give advice, influence the timing of tasks within their management task, or handle exceptions. The only behaviors given to managers are to (1) hold periodic team meetings and (2) keep track of the progress of their subtasks.

Allowing managers to give advice or influence the timing of their subtasks is equivalent to extending the managerial role (see preceding section), and adding a few types of exhortation messages. Adding managerial exception handling as suggested by Galbraith [30] requires that a mechanism for creating the exceptions and for resolving them.

Exceptions may be created either within a task or between tasks. An exception within a task may spontaneously arise (stochastic ir cidence) or occur at programmed intervals. An exception from a dependency between two tas'.s could be created either spontaneously, at programmed intervals while the dependency is active, or with some probability every time that information is transferred between the tasks. Once created, an exception would halt progress in the task or tasks that created it.

Resolving the exception would require the exception be brought to the attention of the responsible manager (a new type of message), the manager be sufficiently informed to the state of the tasks (some information transfer), and the manager to pass back an exception resolution (another new type of message). Particularly tough exceptions might be passed on to the next level of management.

Incorporating managerial exception handling into the model would allow the simulation to test the effectiveness of the management hierarchy as well as the teams assigned to the design process. Readers interested in this potential improvement would be well advised to sample the Virtual Design Team literature [16, 17, 47, 48].

Task Quality and Rework

The current simulation does not allow the experimenter to directly model task quality tradeoffs and project rework. These effects are strongly related; the quality of a product is related to the amount of time spent on it. Rework in a project is usually done because the original quality was not sufficiently high.

To include quality and rework into the simulation, it is sufficient to simply tag all generated information as possessing a certain level of quality. The longer an agent spends working on a task, the higher the quality of the resulting information. Agents would be allowed to "rush" tasks at the price of generating lower quality information. This lower quality information would be passed on through the dependencies to agents working on downstream tasks. Maximum quality of the work on the downstream task would be limited by the quality of the information it was based upon. Agents could also assume information that did not yet exist and perform work on a task beyond that allowed by the dependencies, all at the cost of a lower quality product.

Rework comes into the picture when an agent has the time to go back to a task and improve its quality. Improvements to the information of a task would then be passed on to other tasks who could then improve their own quality level and thus propagate further changes.

Depending upon the interest of the experimenter, a variety of termination criteria for the design project could be employed. For example, the project could terminate when all tasks have reached a sufficiently high level of quality. Or it could terminate when a designated amount of

time has passed; the project performance would then be a function of the quality of the resulting tasks.

This method of introducing quality and rework into the design process would require extremely extensive changes to the ways that agents store and transfer knowledge. It would also require great improvements in agent behavior. Agents would be called upon to decide not only what to work on, but to what level of quality.

Decision Making and Teamwork

The effectiveness of individuals working together in teams and making joint decisions is critical to the success of a design project. The current *DiFS* simulation does not address the interpersonal relationships between agents and their efficiency of coordinating their activities, reaching consensus on actions to be performed, and committing to carrying out those actions in a timely fashion.

Incorporating these effects into the model involves creating a representation for individual motivation and how individuals can influence each other. Teamwork is not simply convincing everyone to work hard and exchange information freely; agents already do that to a limited extent. To properly represent teamwork, agent knowledge needs to be extended so that an agent has a model of what other agents are planning to do. Using this model, an agent would be able to predict how the project will progress and when information will be available.

Experimenting with teamwork requires extending the personal behavior model to include relationships with other agents, motivation to work on assigned tasks, a model of what other agents are planning to do, and methods of communicating this information with other agents. For maximum utility, these extensions are best combined with the task quality and rework extensions suggested above. These simulation enhancements have great potential in allowing experiments with the social structure of the design team. However, an accurate and efficient implementation would not be trivial.

The Underlying Model of the Design

Ultimately all improvements in the information-flow model and *DiFS* simulation are limited by a simple fact: the model does not care about *what* is being created, only that there are a bunch of tasks to be accomplished. The dependencies between tasks specify how much information is to be transferred, but they do not know what that information means.

I believe that eventually the correct way to extend the information-flow model is to divide it into two parts: an underlying part that somehow specifies what is being designed, and an overlaying layer that specifies the sequence of tasks to be performed to do the design. The tasks would still generate information, but that information would be directly connected to the specification of the thing being designed.

This addition of a layer that specifies *what* is being worked on would potentially solve many of the current limitations of the design process. Rework and quality of the design would be straightforward to implement. Information would be tagged with what part of the design it refers to; hence overlapping sources of information could be correctly modeled. Specifying the nature and number of dependencies between tasks could potentially become simpler (or at least removed to a different layer of the modeling process).

Appendix A

DiFS User's Manual

This appendix is the User's Manual for the *DiFS* computer program, revision 5.5. *DiFS* is an acronym for "Design Information-Flow Simulation." Although *DiFS* was written as a part of the author's thesis, it still should be considered an ongoing piece of work. For the latest version, you should contact the author or Professor Warren P. Seering. *DiFS* is a Macintosh program. It is written in C++ using the Metrowerks CodeWarrior environment and PowerPlant. It runs best on PowerPC Macintoshes, but is available for 68000 machines as well (running System 7 or later). Versions of the program with the extension "PPC" are compiled for use on PowerPC machines only. Versions of the program with the extension "68K" are compiled for use on 68000 machines, but will run in emulation mode on a PowerPC machine. If the program is called simply "*DiFS*," then it is a fat binary and will run native on either type of machine. *DiFS* is not currently available for Intel-based computers and there are no immediate plans for porting it.

A.1 Introduction

The intent of this section of the user's manual is to assist a person in the use of the *DiFS* program. It is not intended as a general explanation of the entire Ph.D. thesis. The preliminary sections of the document provide a brief overview of the ideas and purpose of the thesis. For a more detailed description, read the thesis.

A.1.1 Purpose of the Simulation

Imagine that you are a manager in charge of a group of people who have an engineering design project that needs to be accomplished. How long will it take? Who should work on what first? Would adding or removing one person help or hinder your progress?

Managers attempt to resolve these problems by modeling the design process. One approach is to write down all the tasks that need to be accomplished, what skills are required to accomplish those tasks, and what dependencies exist between the tasks. By comparing the requirements with the available resources, you can calculate roughly how long the design process will take.

The difficulty with the traditional approach is that it works best for projects that divide into nice little chunks of work; this is a common occurrence on a construction site, but an uncommon occurrence in a design team. Moreover, the traditional approach has difficulty in answering questions like: Would adding another person speed up the project? What if some of the people work in one building and the others work elsewhere? Answering such questions depends strongly on the experience of the modeler.

The purpose of this thesis is to come up with a "better" method of modeling the design process. *DiFS* is the latest incarnation of a series of programs that attempt to model the design process.

A.1.2 Approach

The *DiFS* program takes a two-part approach to modeling the design process. The first part is a method of modeling the tasks to be accomplished in a manner that lends itself to computation. This modeling method is called the "information-flow model". The second part is a dynamic simulation of the office environment. Rather than try to calculate how long it will take to accomplish a design project, we simulate the actions of the people and watch what they do.

Information-Flow Model

Traditionally managers model the design process by writing down the tasks that need to be accomplished and the temporal dependencies between these tasks. These tasks may be large, involving dozens of people. But engineers on a day to day basis work on tasks they are individually responsible for. Traditionally dependencies between tasks are modeled as temporally serial; that is, the first task must finish before the second can begin. However, design tasks often overlap extensively and may be dependent on each other simultaneously.

The information-flow model solution to the modeling problem is as follows: First, we define tasks to be small chunks of work with a definite purpose and a single responsible person. These tasks are collected together into management tasks. Second, dependencies between tasks represent a need for information. Each dependency specifies when the information is needed and how much information is required.

The individual **work task** in the design simulation is a small task that is the responsibility of a single person. Parameters within the task specify how long it will take the responsible person to accomplish the task (work-time) and how long it would take the responsible person to communicate all of the information generated in the task to another person (comm-time). The work-time is the amount of time it would take to finish the task if the engineer already knew everything he or she needed from all driving dependencies. The comm-time is the amount of time it would take for the engineer to fully explain the completed task to another individual so that that individual completely understood the task and all of its aspects. The comm-time is a measure of the information content of the task.

Not all work is done by an engineer working in isolation. It is common for an engineer to work in a group making decisions. In this case, the task is referred to as a **group decision task**. Group decision tasks are modeled the same as the individual work task; the difference is that one person is in charge of the team (the responsible engineer) and the remaining people participate on the team.

Finally, it is common for a manager to group together tasks and assign someone the responsibility of overseeing their progress. This case is referred to as a **management task.**

Tasks are not done within a vacuum; they almost always depend on results from other tasks. In a design project, this **dependency** is a need for information. The comm-time of a task measures the total amount of information generated by that task. A task that depends on another task generally does not need all of the information generated, nor does it necessarily need that information before any work can be done on it (that is, two tasks may overlap). These factors are modeled by the **scope**, **thoroughness**, and **completion function** of a dependency between two work tasks. Scope, and thoroughness act to trim the amount of information required; the completion function determines when that information is required.

Scope is the breadth of knowledge required. Thoroughness is the depth of knowledge required. If task A is the artistic design of a chair and task B is a survey of potential users, the dependency of B on A might be modeled as a wide scope (we need to know what the whole chair looks like) but a narrow thoroughness (we don't care about the exact dimensions). If task C is the manufacture of the chair legs, the dependency of C on A has a narrow scope (we only care about the legs), but a high thoroughness (we need to know the precise dimensions).

The completion function is modeled as a function relating progress in the driving task to allowable progress in the dependent task. The stronger the completion function, the more serial the dependency. Refer to page 210 to see a sample completion function.

Dynamic Simulation of Individual Behavior

Given a precise description of the tasks that make up a design model, the dependencies between those tasks, and the individuals responsible for those tasks, it is in theory possible to calculate the optimal behavior of the individuals so that the tasks will be accomplished in minimum time. This computation would involve specifying the precise times and tasks the individuals work on, as well as when they communicate with each other and what they say. Given current technology, I believe that this computation is intractable. Moreover, it is of doubtful value because humans do not always behave in a fashion guaranteed to give optimal performance. Hence, the second part of the thesis modeling approach is to simulate the behavior of individuals in an office environment.

Simulations have a number of advantages over direct computation (besides being tractable). For example, a simulation can easily take into account the nature of communications between individuals, the assignment of individuals, and the arrangement of an office. The disadvantage of a simulation is that you must create people who behave rationally (or as some would argue, with bounded rationality). The advantage is that you can easily modify the conditions of the simulation or the behaviors of the individuals to see how they affect the overall progress of the design project.

To create a tractable simulation of the world, this program incorporates a number of fundamental decisions and compromises:

- Each engineer in the project is modeled as a computer agent in the simulation. Each agent has a well-defined list of what tasks they are assigned to in the simulation. Every agent in the simulation knows what tasks every other agent in the simulation has been assigned to.

- The office is modeled as a simple collection of rooms. Each agent is assigned an office and given a list of equipment that they use to communicate with other agents.

- Agents may only transfer information via well-defined communication channels by using the equipment they possess. Different methods of transferring information have time requirements that may be set by the person setting up the simulation.

- There is no telepathy. Agents do not automatically know what other agents have done. It takes time to transfer information from one agent to another.

- The simulation runs on a day-to-day basis. Agents come in to work in the morning and go home in the afternoon. Agents may be assigned a percentage of work time; in this case they go home earlier in the day.

- Meetings are scheduled on a global calendar. To schedule a meeting, an agent specifies a list of agents who will attend and the duration of the meeting. The global calendar selects an appropriate time and place for the meeting that will not conflict with other scheduled meetings. The price of the convenience (it generally isn't done this way in the real world) is that meetings must be scheduled at least a day in advance.

- Work always progresses forward. There is no rework. No mistakes are made. All dependencies must be satisfied for work to continue. If you wish to model iterations, you must explicitly include them in your model of the design process.

- Design "Process" is implicitly modeled. In the real world, engineers must follow the corporate standard operating procedure (SOP) while working. In the simulation, SOP is not directly modeled; it is implicitly modeled in the personal behavior of the agents and the requirements for management-task meetings. In future research we may implement some of our approaches to modeling SOP.

- Outside influences and individuals are not modeled, but may be approximated. For example, if you have a task of gathering information from vendors, you can model that as (hypothetically) a 10 hour work task. Realistically the telephone should be tied up at least part of the time and the individual working on the task should not be able to do all 10 hours in one continuous stretch. Currently there is no good way to compensate for this problem.

- There is no global plan. Each agent in the simulation chooses what he/she will do at all times based on a local planning approach. It is possible that in the future we will add suggested start and stop dates for each task.

This approach to simulation has been implemented as the *DiFS* program. The author's thesis should be referred for a detailed discussion of the model, simulation, experiments, and implications.

A.2 Program Description

DiFS is an acronym for "Design Information-Flow Simulation." It is a traditional Macintosh program. You may open more than one data file at a time (I've had three different simulations run simultaneously; more than that and you should probably increase the partition size of the program). Each data file has a single **Main Window** associated with it (see figure A.1). The Main window is used to switch between editing modes and open various types of windows. Closing the Main window closes your data file and all associated windows.

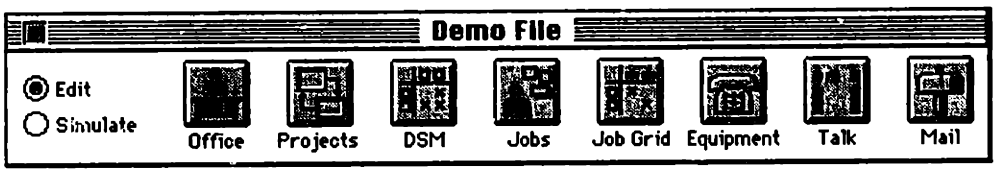


Figure A.1: Edit mode of the Main window.

To edit and run a simulation, *DiFS* contains 16 windows and 8 dialog boxes. The *DiFS* interface is divided into two logical modes: editing and simulation. Switch between these modes by clicking on the radio buttons at the left edge of the Main window (see figure A.1). Each mode has an associated set of icon buttons that are used to open a variety of windows. The functions of the two modes are:

- Edit Mode: Edit the office environment, the tasks, the roles that agents perform on the tasks, and the equipment and communication channels available to the agents.

- Simulate Mode: Runs the simulation. Displays task progress, what agents are doing, and what they are thinking.

I suggest that you open one of the sample files that comes with the program and try your hand at making changes as you read this document.

A.2.1 Editing Tasks and People

Switch to **edit mode** by clicking on the "Edit" radio button in the Main window. Within this mode you modify the office environment, the people within the office, the tasks to be accomplished, and the assignment of individuals to tasks. All of these types of editing are accomplished in dedicated editing windows, which are opened by clicking on the appropriate icon in the Main window. The following sections describe each of the types of editing windows and how to use them.

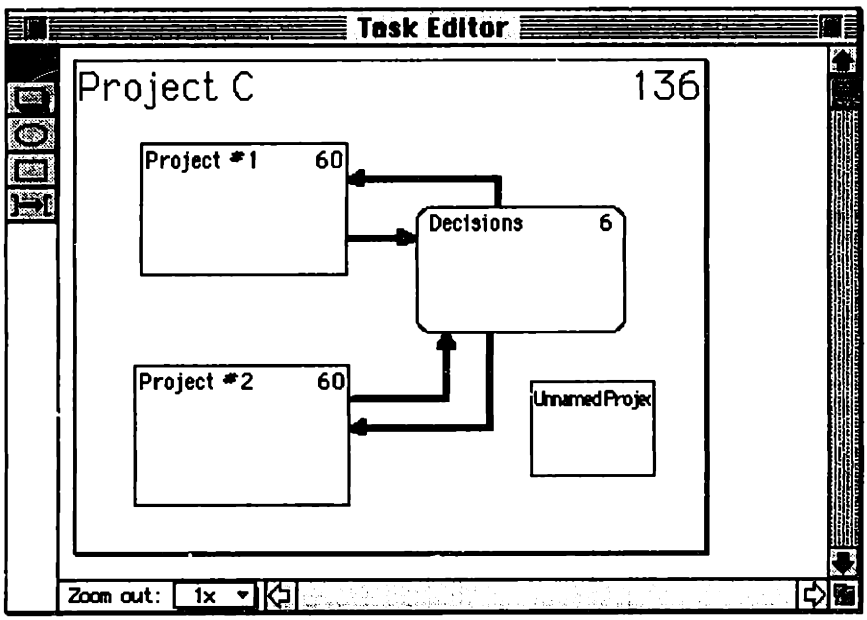


Figure A.2: Task Editor window.

Task Editor


Projects

The Task Editor window (figure A.2) is used to create and modify the various tasks in the simulation and the dependencies between work tasks. Open the Task Editor window by clicking on the Projects icon in the Main window.

Tasks come in two categories: work and management. Work tasks represent well-defined chunks of work to be accomplished. Management tasks represent a group of work and management tasks. Management tasks are also referred to as **meta tasks**. They are the responsibility of a manager who oversees all of the tasks within the management task, attends meetings, and keeps track of progress.

The number displayed in the upper right of each task is the number of hours of work required to finish the task (work time). The work time of a management task is the sum of the work time of its contained tasks; this number is displayed solely for convenience and has no effect on the simulation.

The tools in the toolbar on the left side of the Task Editor window (figure A.2) are used to select tasks and dependencies and to create new tasks and dependencies. Click on a tool to use it. Note that the creation tools automatically switch to the arrow tool after a task or dependency has been created. Double-clicking on a creation tool locks it; that is, it will not automatically switch to the arrow tool after the item has been created.

The "Zoom Out" menu on the bottom-left of the Task Editor window allows you to view large projects on a small screen. It affects printing, so be sure to select "1x" before printing.

Creating Tasks and Dependencies: The task tools are used to create tasks. Click and drag within the desired parent management task to create a task. The simple task tool creates an individual work task. The rounded edge task tool creates a group decision task. The task tool with a drop shadow creates a management task.

The right pointing arrow tool is used to create dependencies. Click on the task you wish to "drive" the dependencies and drag to the task you wish the be "driven" by the dependency. Note that you can only connect work tasks via dependencies; management tasks can not drive or depend upon a dependency. Also note that there may only be one dependency connecting a driving task with a driven task (although you may have a separate dependency connecting them in the other direction).

Moving Things Around: The arrow tool is used to select, move, and resize tasks and dependencies. Click on a task to select it: selection handles will appear. If you select a task other than the top-level management task, then the parent task of the task you have selected will be drawn with a gray border. This is an easy way to check to which parent a child task belongs.

Clicking and dragging on a selection handle resizes a task. When resizing, tasks are constrained to remain within the bounds of their parent management task. Management tasks are constrained to remain larger than their contained tasks. If you wish to resize a management task to be smaller and you are running up against a subtask, move or resize the subtask first.

Click and drag on a task to move it. A moving task is not constrained to remain within the bounds of its parent task; it is constrained to remain within the bounds of the top-level task. The top-level task is free to be dragged anywhere in the work surface. Always be careful when dragging; it is quite easy to pull a task out of its parent or to accidentally cover up another task (look for the gray border to see which task is the current parent task).

Tasks may be cut, copied, pasted, and deleted. A cut or copied task is placed on the clipboard. Pasting from the clipboard attempts to put the task into the *currently* selected management task. A pasted task is not resized, so it is important that the management task it is being pasted into is large enough to accept it. When in doubt, you can always enlarge the top-level task, paste into that, and then drag the pasted task to its desired location.

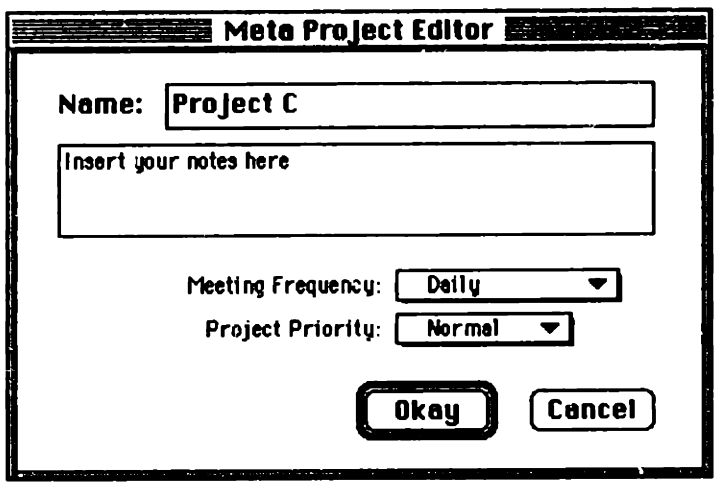


Figure A.3: Management task dialog box.

Editing Management Tasks: At all times there will be a top-level management task. You cannot delete this top-level task. All tasks that you add to the system are contained within the top-level task. The top-level task is always a management task. Select a management task and choose "Inspect" from the Items menu (or double-click on it) to edit its parameters (see figure A.3).

The management task dialog box has a few special features. **Meeting Frequency** is how often the person responsible for a management task will call a general meeting of agents involved in the management task (i.e., agents marked either as responsible or assisting). The purpose of this meeting is to share information, not to make decisions. **Task Priority** is used by the individual responsible for the task to help decide how important this task is.

Editing Work Tasks: Select a work task and choose "Inspect" from the Items menu (or double-click on it) to edit its parameters (see figure A.4).

The type of the work task may be Individual Project, Group Decision, or Milestone Meeting. Individual projects are work tasks to be done by an individual. Group decision tasks are done

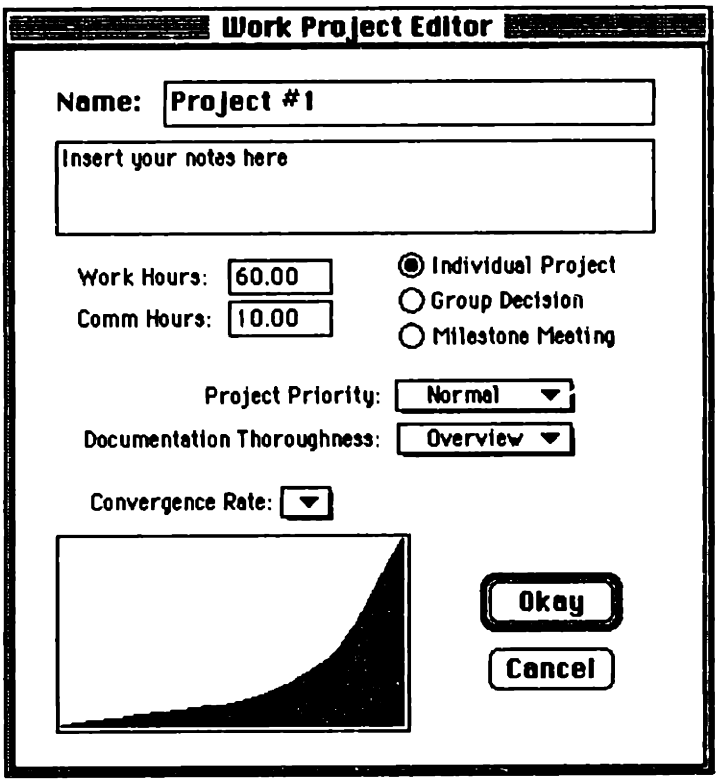


Figure A.4: Work task dialog box.

by teams of agents working in a common discussion. Both of these types of tasks have been described before. The milestone meeting task is a special form of group decision task; the agent responsible for this task will attempt to finish this task in a single day by scheduling one long meeting. This type of task can be used to model project reviews and important meetings.

Work tasks have two time measurements: hours of work and hours of communication. **Work Hours** is how many hours of concentrated work must be done on this work task in order to finish it. This number does not take into account any communication or information gathering, meetings with the boss, or other such work-related activities. In the case of individual tasks, this is the amount of "alone" work to be done by the person responsible. In the case of group tasks, this is the amount of "decision" time in a general meeting.

Comm Hours is a measure of the information content of the work task. This is the number of hours it would take for one agent to completely explain the results of a task to another agent. All time requirements for dependencies are subsets of this time.

Task **priority** is an experimenter-defined importance attached to a task. The computer agents use the task priority when they evaluate the utility of working on a task.

Documentation thoroughness is a measure of how much of the information of a task is written down as a part of working on the task. That is, if the agent responsible for a task suddenly disappeared, the documentation thoroughness measures how much of the information generated would still be available. Because this information is readily available, it may be easily sent from agent to agent. This written information is the formal information generated while working.

The **Information convergence curve** at the bottom of the dialog box may be edited by clicking in the box and dragging the inflection point of the curve. You may also select from several sample preset curves via the popup menu. The convergence curve represents the amount of information generated by working on the task (ordinate or "y-axis"; units are hours of communication) as a function of the amount of time spent working (abscissa or "x-axis";

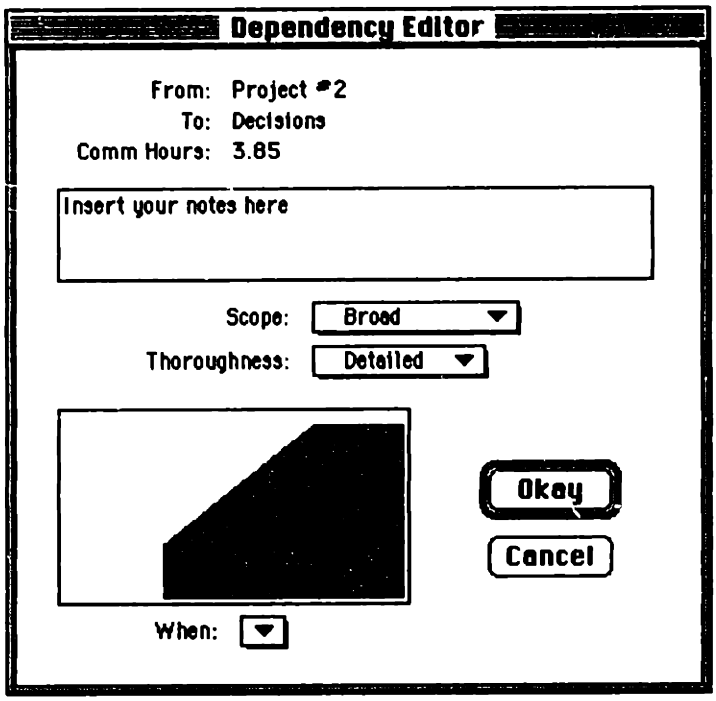


Figure A.5: Dependency dialog box.

units are hours of work). The convergence curve is constrained to monotonically increase from (0, 0) to (100%, 100%).

Editing Dependencies: Click anywhere on a dependency to select it. Selected dependencies show up in gray. Control points may be added to a dependency to route it around tasks. Choosing "Add Point" from the Items menu will add a control point to the currently selected dependency. To delete a control point, select it and then choose "Remove Point" from the Items menu. A maximum of eight (8) points are allowed for routing dependencies. Click and drag on a control point to move it.

Select a dependency and then choose "Inspect" from the Items menu (or double-click on it) to edit its parameters (see figure A.5). The Dependency dialog box displays the driving task, the dependent task, and the calculated required communication time for this dependency. The calculated communication time is the total quantity of information required to be transferred from the agent responsible for the driving task to the agent responsible for the dependent task. It is a function of the scope, thoroughness, and completion function of the dependency as well as the total communication time of the driving task.

The scope and thoroughness popup menus are used to select appropriate levels of scope and thoroughness for the dependency. The completion function relates how complete the dependent task (ordinate or "y-axis") can be as a function of the completeness of the driving task (abscissa or "x-axis"). The popup menu adjacent to the graph provides a number of common cases. You may also click and drag on the graph to rearrange it.

Project Grid Window

There is a separate editing window that can be used for displaying dependency relationships between work-tasks; this is a Design Structure Matrix (DSM) representation of the project. Open the Project Grid window by clicking on the DSM icon in the Main window.

The Project Grid window displays all of the dependencies between work tasks in a matrix

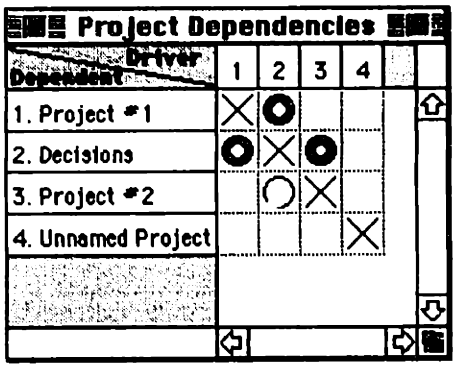


Figure A.6: Project Grid window.

(see figure A.6). Each dependency appears as a circle in the grid. The thickness of the line used to draw the circle is proportional to the integral of the completion curve of the dependency; that is, stronger dependencies show up as darker. The work tasks in this window are automatically sorted in the standard DSM reduced form (i.e., as close as possible to lower-triangular form).

Clicking in any square in the grid pops up a menu that allows you to add, edit, or delete dependencies between two tasks. Note that all editing changes will show up in this window and in the Task Editor window.

Verification: It is quite possible to create a set of work tasks and dependencies that cannot be accomplished. For example, if you create two work tasks and make them each strongly dependent on the other, then it will be impossible for people to successfully complete these two tasks. This case is called "over-constrained".

The Task Editor window (figure A.2.1) provides an option for checking to insure that your project is not over-constrained. With the Task Editor window in front, select "Verify" from the Items menu. The computer will begin moving time forward in 10 minute steps, calculating the maximum possible work in each task at each step. If the project is over-constrained, it will stop time and report the condition. By watching the visual progress, it is straightforward to locate the offending dependencies. Select "Clear Progress" from the Items menu to remove the progress bars.

If the project is under-constrained, the simulation will inform you of the total amount of time that it took to complete the project (in hours). Note this time calculation is almost the minimal time possible (an infinitesimal step size, instead of 10 minutes, would be the minimal time calculation). The verify process assumes (1) no communication is required, (2) there are an infinite number of agents available to work on the tasks, and (3) meetings never interfere with work. Normally the simulated amount of time required to complete the design task is considerably longer than the verified amount of time required to complete the design task (often by a factor of 4 or more).

Boilerplates: A common complaint about the information-flow model is that it requires the person using the simulation to create a large number of small tasks and specify the dependencies between them. This is a tedious process. It is my belief that it is possible to identify commonly recurring types of task relationships within a company. Once identified, these corporate "motifs" can be placed in a library so that managers would build their models out of the library rather than individually specifying each task and dependency.

This thesis is not in the business of creating such a library. However, to demonstrate one way that this library may be implemented, I have created a collection of **boilerplates**. A boilerplate is a template for creating common types of management task.

To use a boilerplate, first open the Task Editor window. Then select a boilerplate from the "Boilerplate" submenu of the Items menu. Fill in the dialog box parameters and select "Okay"

to insert the boilerplate into your task window.

Figure A.7 is the dialog box associated with the Decision Group boilerplate. This boilerplate represents the act of working on a collection of tasks that are coupled through joint decisions. Figure A.8 shows the result of inserting this boilerplate into the Main window.

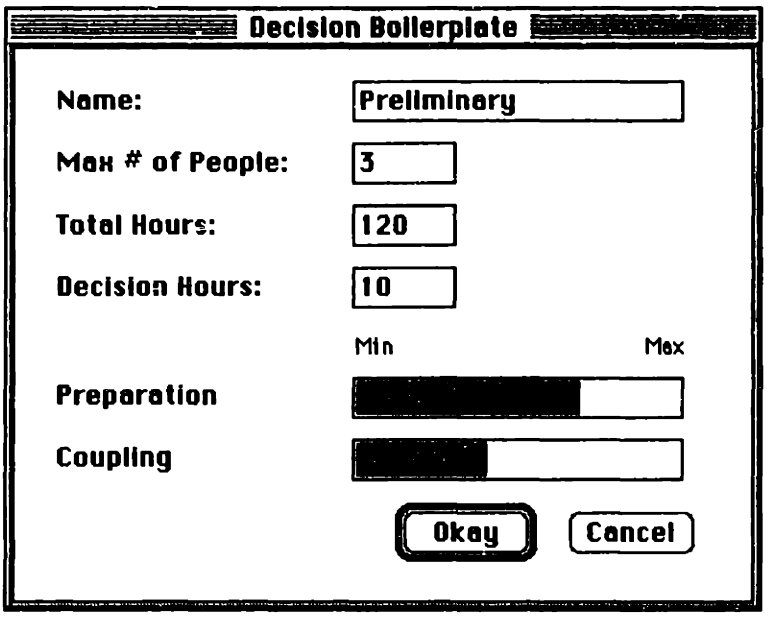


Figure A.7: Decision group boilerplate dialog.

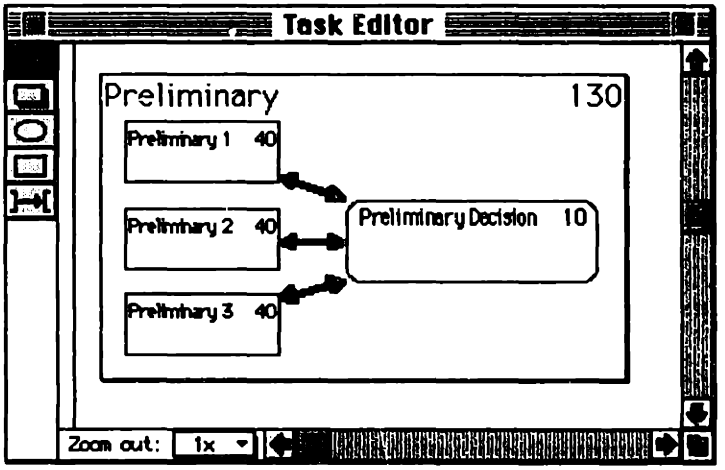


Figure A.8: Decision group in the Task Editor window.

Office Editor


Office

The Office Editor window is used to design an office layout and create agents (see figure A.9). Open the Office Editor window by clicking on the Office icon in the Main window. Like the Task Editor window, the Office Editor window contains a toolbar on the left side and a "Zoom Out" menu on the bottom-left (refer to page 207 for a description of the operator of the toolbar and zoom menu).

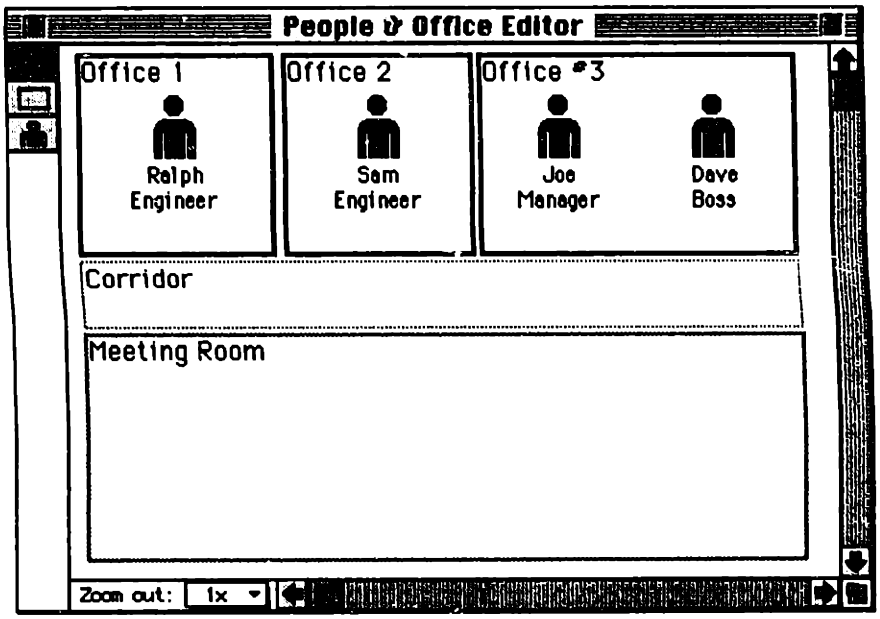


Figure A.9: Office Editor window.

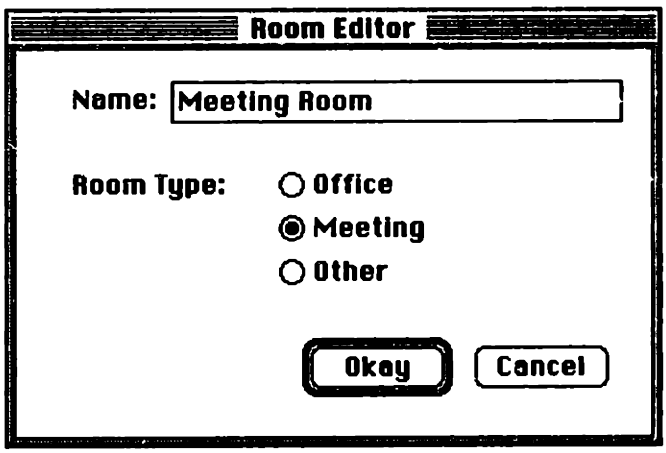


Figure A.10: Room Editor dialog box.

Working with Rooms: The room creation tool is the square tool in the tool palette. Create a room by choosing the room creation tool and clicking and dragging in the window. Note that rooms may freely overlap and obscure one another. Choose the arrow tool and click on a room to select it. Selected rooms may be resized by clicking and dragging on their selection borders, or may be moved by clicking and dragging on the room itself. Select a room and choose "Inspect" from the Items menu to edit its parameters (or double-click on it; see figure A.10).

Rooms may be of one of three possible types: office, meeting, and other. Offices are places where agents work. Agents may only be assigned to an office, and the office that they are assigned to is considered to be the location of their desk. Meeting rooms are publicly available rooms where meetings may be scheduled and held. It helps to make meeting rooms large on the screen so that you can easily see all of the agents currently in the meeting room. If there are no meeting rooms in your layout, agents will hold meetings in their office (or they may teleconference). "Other" rooms are never used by the simulation; they just exist to help you create a visually appealing office layout. I often use them for corridors, closets, and the like.

The "Meeting" and "Other" radio buttons in the Room Editor dialog box (figure A.10) will

be disabled if there is an agent currently installed in the room. You must remove the agent to change the room to a type other than office.

Working with Agents: To create an agent, select the agent tool from the tool palette and click in any office. The office an agent is located in is the office that his/her desk is located in (and where he/she will work). You may freely select and move agents from office to office (using the arrow tool).

Double click on an agent or select "Inspect" from the Items menu to edit an agent's parameters (see figure A.11). **Time Available** is the percentage of the normal working day that this agent is available to work. You may enter a number from 1 to 100. For simulation purposes, this number is used to calculate when an agent goes home and how many days per week he or she works. The number of hours worked per day is 9 hours multiplied by the percentage of time available. If this number comes out to less than approximately 2 hours per day, the individual will only come in on the first day of the week (i.e., Monday) and will skip sufficient weeks so that his/her working day is at least two hours long. For example, assume Sue works 6% of the time. This would be only 0.54 hours if she came in daily, but 2.7 hours if she came in weekly. Hence Sue will show up for 2.7 hours once a week (days 0, 5, 10, ...) If Sue worked only 3% of the time, she would come in for 2.7 hours every two weeks (days 0, 10, 20, ...).

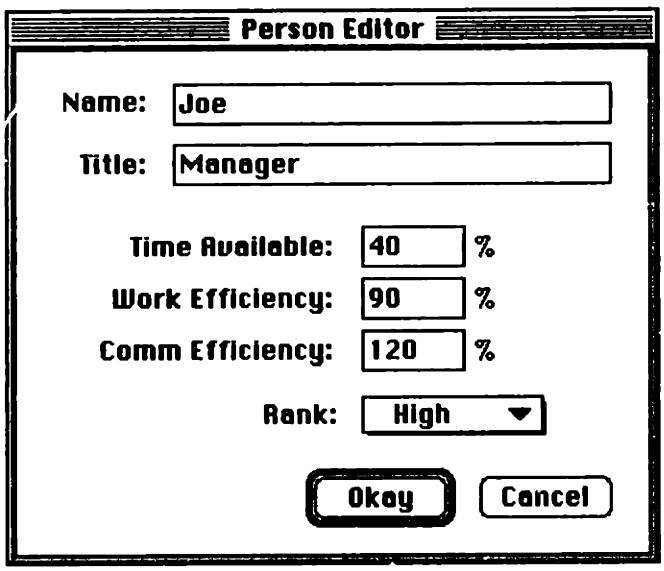


Figure A.11: Agent Editor dialog box.

Work efficiency is how fast the individual completes things. A work efficiency of 100% is the nominal rate. A work efficiency of 200% would mean that the individual will accomplish solo work in half the nominal time. Conversely, a work efficiency of 50% means that the individual will take twice as long as nominal to finish the task.

Comm efficiency is how fast the individual communicates. Just like work efficiency, a value of 200% means it takes half the time and a value of 50% means it takes twice the time. Note that in a group decision task, these two factors both apply. That is, if the leader of a group decision task has a work efficiency of 200% and a comm efficiency of 200%, then the group work will go four times as fast as nominal.

Rank is the agents's stature in the organization. It is used to help decide who should get to talk first in various situations. In the future we may also use it to rate how important requests are considered to be and so forth.

Job Assignments

There are two separate ways of assigning agents to roles on tasks: the Job Grid window and the Job Assignment window. Both perform exactly the same function, but display the data in different ways.

Let me quickly summarize the currently available roles for agents on tasks. There are two types of roles: responsible and assisting (or participating). The function of these roles varies depending on the type of task. In general, agents who are responsible for a task call meetings regarding that task and do the work. Agents who assist on a task attend the meetings. For a solo work task, the responsible agent does the work. Assisting agents currently serve no purpose (but may in the future). On a group decision task, the responsible agent calls regular meetings and leads the discussions. All participants must show up to the meeting for work to be accomplished. On a management task, the responsible agent calls general meetings for which all of the assisting agent show up for.

The responsible agent on a management task also keeps track of the progress of all of the tasks contained within the management task. The responsible agent is the preferred contact source of information on a task (in the future we may add the *liaison* role). The only exception to this rule is for agents who are responsible for high-level management tasks. They prefer to receive status information on contained tasks by contacting their immediate subordinates for information rather than directly contacting the responsible agent.

Job Grid Window


Job Grid

The Job Grid window displays all of the tasks in the simulation versus all of the agents in the simulation (see figure A.12). Open the Job Grid window by clicking on the Job Grid icon in the Main window.

Job Grid Assignments

Person / Project	Ralph Engineer	Sam Engineer	Joe Manager	Dave Boss
Project C			Assist	Responsible
Decisions	Assist	Assist	Responsible	
Project #1		Responsible		
Project #2	Responsible			
Unnamed Project				Responsible

Figure A.12: Job Grid window.

Each box in the Job Grid window shows an agent's role on a task (note that an agent may have only one role on a given task). Clicking in a box brings up a (short) menu of the possible roles for that agent on that task. Use the menu to add, change, or remove a agent's role on a task. Note that only one agent at a time may be responsible for a task. If the "responsible" field of the menu cannot be selected, that means that another agent is already responsible for that task. To make a new agent responsible, you must first remove the old agent's role.

Job Assignment Window


Jobs

The Job Assignment window is a different way of assigning roles (see figure A.13). It performs the same function as the Job Grid window. Open the Job Assignment window by clicking on the Jobs icon in the Main window. The Job Assignment window is split

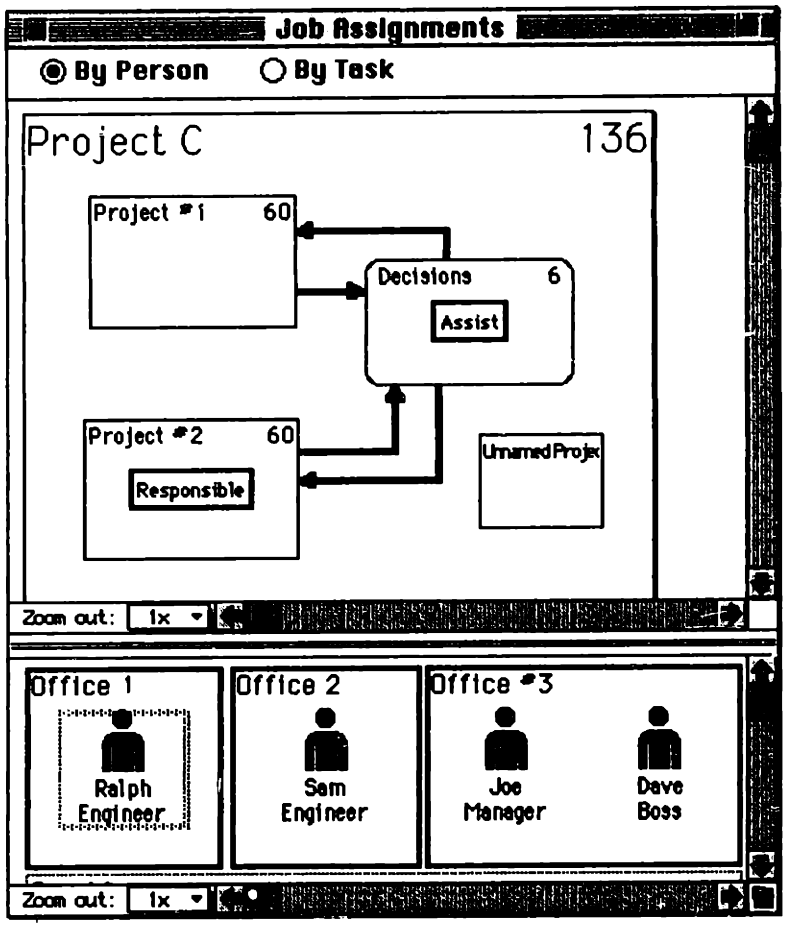


Figure A.13: Job Assignment window.

into two panes; a view of the agents in the simulation and a view of the tasks in the simulation. Click and drag on the horizontal line dividing the two panes to resize them.

The radio buttons at the top of the Job Assignment window control how the window behaves. If you select "By Person" and click on an individual in the simulation, then that agent will be highlighted and all of the tasks they have roles on will display labels showing the agent's role on that task. Click on a task to display a menu for adding, changing, or deleting an agent's role on that task. Click on a different agent to display that agent's roles on tasks.

Selecting "By Project" and clicking on a task will highlight the selected task and display all of the agents with roles on that task. Click on an agent to display a menu to add, change, or delete the agent's role on the selected task.

With the Job Assignment window open, you can quickly check to insure that there is at least one agent with the "responsible" role on each task in the simulation. Select "Verify" from the Items menu, and the program will check for you.

A.2.2 Exchanging Information

Agents exchange information with each other by experimenter defined communication channels. Agents interact with a communication channel by using office equipment. For example, consider three agents participating in a teleconference. The communication channel they are jointly participating in is the teleconference itself. The equipment each agent uses to participate in the teleconference is his/her telephone. While in teleconference, each agent's telephone is occupied and (assuming no "call waiting") cannot receive other telephone calls.

Equipment types are defined by the experimenter to model all types of equipment that agents may use to communicate with each other. In fact, agents may only communicate with each other through equipment; there is no concept of intrinsic communication abilities like talking or handing an agent a message. All communication must be done through modeled equipment.

Communication channels are defined by the experimenter to model all the ways that two or more agents may exchange information. There are two classes of communication channels: synchronous and asynchronous. Synchronous communication occurs when two or more agents are interacting in a conversation. Synchronous communication is generically referred to as "Communication". Asynchronous communication occurs when one agent sends another a message (e.g., faxes them a document). Asynchronous communication is referred to as "Transmission". Note that in a transmission an agent does not know when the intended recipient will receive the message.

Equipment Window


Equipment

The Equipment window is used to add and remove types of equipment from the simulation (see figure A.14). Open the Equipment window by clicking on the Equipment icon in the Main Window.

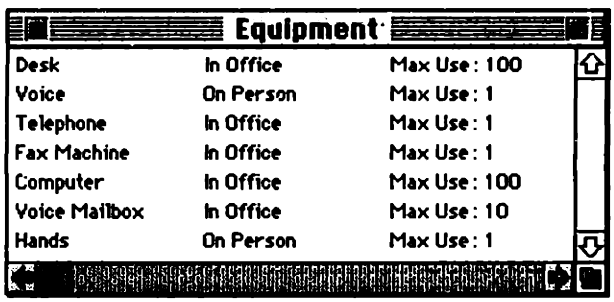


Figure A.14: Equipment window.

Equipment is used in conjunction with a communication channel or transmission channel to send a message from one agent to another. Sample equipment includes a telephone, an answering machine, a fax machine, a person's voice, a person's hands, a mailbox, a pager, a computer (used for E-mail or electronic "talk"), video-conferencing equipment, etc. A new data file starts with a few sample pieces of equipment. Note that all agents in the simulation have exactly the same equipment available to them at all times (a restriction that might be changed in future versions).

The Equipment window shows a list of the equipment possessed by each agent in the simulation, as well as whether each piece of equipment is portable and its maximum usage. Select the equipment you wish to edit by clicking on it. Double-click on the equipment to edit it or select "Inspect" from the Items menu. Add a new piece of equipment by selecting "Add" from the Items menu, and delete a selected piece of equipment by selecting "Remove" from the Items menu (or "Delete" from the Edit menu).

The equipment dialog box allows you to name the equipment and set its parameters (see figure A.15). **Maximum Simultaneous Usage** refers to how many transmissions and communications can utilize this piece of equipment at the same time. For example, a Fax machine has a usage of one (1) because it is incapable of being used by anyone else when it is sending or receiving a fax. A computer terminal has a usage greater than one because it is capable of receiving E-mail from more than one source simultaneously.

Agents either carry equipment or leave it on their desk. **Carried by Person** refers to whether or not the equipment is always in the same room as the agent it is assigned to. If it

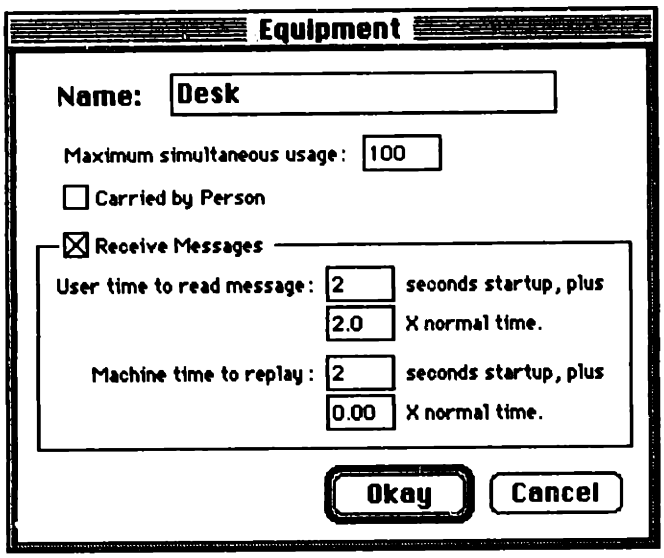


Figure A.15: Equipment dialog box.

Table A.1: Sample equipment time parameters.

		Answering Machine	Desk (Incoming Mail)
User	(constant)	5 seconds	2 seconds
	(multiplier)	1.1 ×	2.0 ×
Machine	(constant)	8 seconds	2 seconds
	(multiplier)	1.1 ×	0.0 ×

is not carried (or portable), the equipment is in the owner's assigned office. An agent can only access non-portable equipment when in his/her assigned office.

Receive Messages refers to the equipment's ability to receive transmissions. Receiving a transmission requires the equipment to store an incoming message for an indefinite period and then play it back for the recipient. Messages are stored by a FIFO basis (First In, First Out). Examples of equipment types that receive messages include: answering machines, mailboxes (or desks), computers (e-mail), fax machines, a person's hands (used to directly hand information from one agent to another), and pagers. Examples of equipment types that do not receive messages include: ordinary telephones (not portable), voice (a person's ability to talk face to face), and cellular telephones (portable).

An equipment type that can receive messages has four fields used for calculating how long it will take the agent to extract a message out of the equipment (these fields are ignored for equipment that does not receive transmissions). These fields are divided into two categories: how long it takes the agent to listen to and understand the stored message and how long it takes the machine to replay the stored message. Each pair of fields consists of a constant amount of time and a multiplier. Each message sent out has a nominal amount of time associated with it. The nominal amount of time of a message is how long it would take an agent to "send" the message to another agent in a face to face communication. The amount of time it takes the agent to read the stored message is equal to the constant plus the product of the multiplier and the nominal message time. The amount of time it takes the machine to replay the message is calculated in the same way.

Table A.1 shows some sample time parameters for two types of equipment. Given a 30 second (nominal time) message, table A.2 shows the calculated times it will take for an agent to retrieve the message from each type of equipment and the amount of time it will take the equipment to replay the message. Note that the answering machine takes a certain amount of startup time

Table A.2: Time to retrieve a 30 second (nominal time) message.

	Answering Machine	Desk (Incoming Mail)
User time	38 seconds	62 seconds
Machine time	41 seconds	2 seconds

(associated with the rewinding of the tape, etc.), and that the machine time required is longer than the user time required. Hence the answering machine is occupied the entire time that the agent is replaying the message and can't accept any incoming messages. On the other hand, the desk (which stores incoming paper mail) is not occupied the entire time the agent is reading the message. The agent spends a few seconds of digging through papers, and then the desk is free to be used by someone else.

Communications Window


Talk

The Communications window displays the various communication channels available in the simulation (see figure A.16). Open the Communications window by clicking on the Talk icon in the Main window.

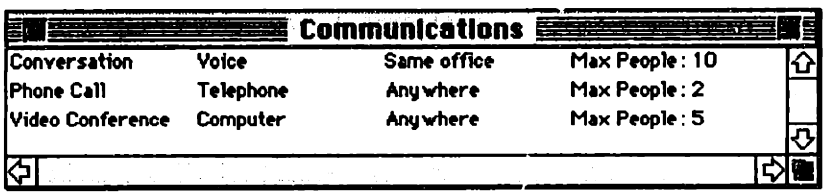


Figure A.16: Communications window.

Communication channels represent synchronous methods of talking between two agents. Communication channels allow an agent to connect to other agents using a particular type of equipment. All meetings are conducted over communication channels (and hence you'd better have at least one, or no meetings will ever occur). Examples of communication channels include: face to face discussion, telephone calls, teleconferences, computer "talk", and videoconferencing.

The Communication window displays a list of the communication channels in the simulation, as well as the equipment utilized, office restrictions, and the maximum number of agents that can be talking on this type of communication channel. Edit a communication channel by clicking on it and choosing "Inspect" from the Items menu (or by double clicking on it). Add a new communication channel by choosing "Add" from the Items menu, and delete a selected channel by choosing "Remove" from the Items menu (or "Delete" from the Edit menu).

The Communications dialog box allows you to name the channel and set its parameters (see figure A.17). **Equipment Type** is a popup menu of the available types of equipment in the simulation. Note that all agents on a communication channel have to use the same type of equipment. **Same Office?** represents a restriction on where the communication channel can be used. If checked, it means that each agent in the conversation must be in the same room in order to participate in the communication. In general this implies that the equipment used for the communication should be portable, because non-portable equipment is always left in an agents's office (it is possible to have same room, non-portable communications if two agents share an office). **Maximum # of people** is the limit on the number of agents who can be a part of a single communication channel of this type. For example, a telephone call might be limited to two agents, whereas a general discussion might be limited to 20.

The **Conversation startup time** parameters refer to how long it will take for the agent beginning the communication to make initial preparations. At the end of this amount of time, each agent who is being asked to join the communication will receive an interrupt (i.e., their telephones will ring). If the equipment they will use for the communication is already engaged,

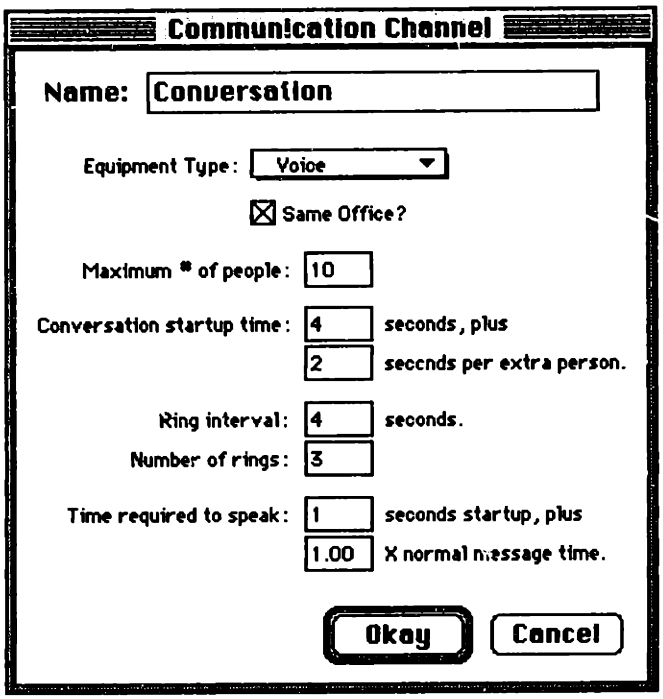


Figure A.17: Communications dialog box.

the agent requesting the communication gets a busy signal. Each agent will continue to be contacted every **Ring interval** seconds until either they have answered the communication or until contact has been attempted **Number of rings** times. The communication channel is declared open once all of the agents being contacted have either answered, been rung a sufficient number of times, or been found busy because the appropriate piece of equipment was unavailable.

The final two parameters in the Communications Dialog box controls how long it takes for the agent to send a message on the communications channel. The first parameter is the constant startup time for the message and the second is the multiplier of the nominal message time.

Table A.3: Sample communication channel parameters.

	Conversation	Telephone call
Equipment type	Voice	Telephone
Same office?	Yes	No
Maximum #	10	2
Conversation startup time	4 seconds	10 seconds
per extra person	2 seconds	NA
Ring interval	4 seconds	6 seconds
Number of rings	3	4
Speaking time (constant)	1 second	2 seconds
(multiplier)	1.0 ×	1.1 ×

Two sample communication channels are shown in table A.3. Note that a 30 second message will take 31 seconds in conversation mode (which is about the optimal communication rate possible) and 35 seconds in a telephone call (because without the visual contact, information is transferred more slowly).

Transmissions Window

The Transmissions window displays the various transmission channels available in the simulation (see figure A.18). Open the Transmissions window by clicking on the Mail icon in the Main window.

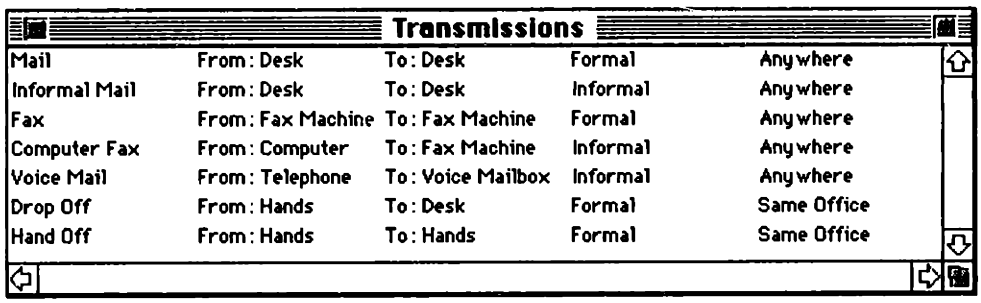


Figure A.18: Transmissions window.

Transmission channels represent asynchronous methods of sending information from one agent to another. Transmission channels allow an agent to send messages to another agent when the other agent is not available; they also allow the exchange of formal documents. Examples of transmission channels include mailing a letter, leaving voice mail, sending a fax, paging another person, sending e-mail, or just handing another agent a document.

The Transmissions window displays a list of the transmission channels in the simulation, as well as the equipment utilized, types of information that can be transmitted on the channel, and office restrictions. Select the transmission channel you wish to edit by clicking on it. Add a new transmission channel by choosing "Add" from the Items menu, and delete a selected channel by choosing "Remove" from the Items menu (or "Delete" from the Edit menu). Edit a transmission channel by clicking on it and choosing "Inspect" from the Items menu (or by double clicking on it).

The Transmissions dialog box allows you to name the transmission channel and set its parameters (see figure A.19). Note that the sending equipment type can be different from the receiving equipment type (for example, voice mail might use a Telephone to send and a Voice Mailbox to receive). The equipment chosen as the receiving equipment should be able to receive messages. Select the **Same Office?** checkbox if the sending and receiving equipment must be present in the same location for the transmission to occur (for example, dropping a message on someone's desk might use sending equipment Hands, receiving equipment Desk, and Same Office checked).

The **Formal** checkbox controls the type of message that can be sent on this channel. If it is selected, only formal documents may be sent. If it is not selected, only informal documents may be sent. If you desire a transmission channel that handles both cases (this is common), then create two versions of the channel where the only difference between the two is the selection of the Formal checkbox and the values of the time parameters. The distinction between formal and informal transmission channels is made for two reasons: first, some types of transmissions are inherently unsuitable for formal documents (e.g., voice mail), and second, because formal documents typically require less preparation time on the part of the agent sending the document.

There are four sets of time fields used to calculate how long a transmission takes: sending person time, sending machine time, transmission time, and receiving machine time. Each is in the form of a constant plus a multiplier. Each of the four types of time may be calculated by taking the sum of the constant plus the product of the nominal message length and the multiplier. **Sending person time** represents how long it takes the agent to send the message. **Sending machine time** represents how long the piece of equipment used to send the message will be tied up. **Transmission time** is how long the message is in transit; that is, how long it will take before the receiving machine gets the message. **Receiving machine time** is how

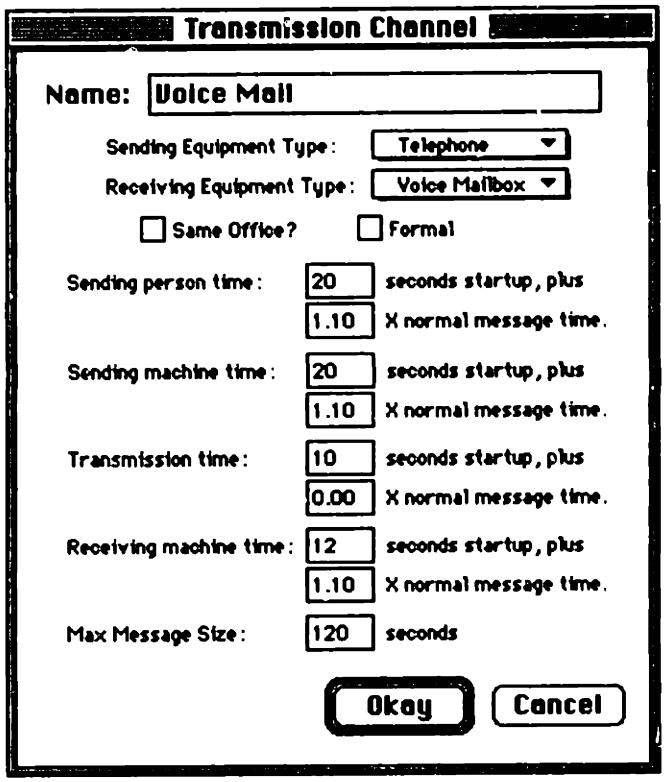


Figure A.19: Transmissions dialog box.

long the receiving machine will be tied up while it is getting the message

The final field, **Max Message Size** is the size limit for any one message going through this transmission channel. For example, an agent can't send a five-minute long message through a pager.

Table A.4 shows some sample parameters for three different transmission channels. Table A.5 shows transmissions times for a nominal 30 second message; these data are also graphically displayed in figure A.20. The overall amount of time it takes for the message to be transmitted (from when the sending agent starts to when the receiving equipment gets it) is equal to the sum of the transmission time and the receiving machine time. You can think of this as having the sending agent, machine, and transmission times all starting at time 0, and the receiving machine time starting when the transmission time ends. Hence it is allowable for the sending machine time to overlap the receiving machine time (as is the case with faxing or voice mail).

Note that the formal mail time is independent of the size of the message; it would take the same amount of time to send 2 hours worth of information as it would to send 30 seconds worth. The constants used in calculating times represent actions like dialing the phone, finding and addressing an envelope, or being carried by the mail system. The multipliers represent how long it takes to transfer the information onto the sending machine and from machine to machine; the multipliers tend to be low (or nonexistent) for formal transmission channels and they tend to be high for informal transmission channels where an agent has to write down information. Hence the informal fax transmission channel takes longer than the voice mail; the agent must locate a fax cover sheet, write out the appropriate message, and feed it into the fax machine. Approximately 230 seconds of the sending agent's time is spent just getting ready to send the message; the actual message goes out in just 16 seconds with an additional 10 seconds at the receiving machine's end. On the other hand, with voice mail, the individual spends just 10 seconds dialing the phone and reaching the voice mail; after that, information transmission takes place at a rate just slightly slower than in a face to face conversation.

Table A.4: Sample transmission channel parameters.

		Formal Mail	Informal Fax	Voice Mail
Sending equipment type:		Desk	Fax Machine	Telephone
Receiving equipment type:		Desk	Fax Machine	Voice Mailbox
Same office?		No	No	No
Formal?		Yes	No	No
Sending person	(constant)	120 seconds	120 seconds	20 seconds
	(multiplier)	0.0 ×	4.0 ×	1.1 ×
Sending machine	(constant)	120 seconds	120 seconds	20 seconds
	(multiplier)	0.0 ×	4.2 ×	1.1 ×
Transmission	(constant)	2 hours	110 seconds	10 seconds
	(multiplier)	0.0 ×	4.0 ×	0.0 ×
Receiving machine	(constant)	5 seconds	20 seconds	12 seconds
	(multiplier)	0.0 ×	0.2 ×	1.1 ×
Max Message Size		100 hours	15 minutes	120 seconds

Table A.5: Transmission times for a 30 second (nominal time) message.

	Formal Mail	Informal Fax	Voice Mail
Sending person	120 seconds	240 seconds	53 seconds
Sending machine	120 seconds	246 seconds	53 seconds
Transmission	2 hours	230 seconds	10 seconds
Receiving machine	5 seconds	26 seconds	45 seconds
Total time	2 hours, 5 seconds	256 seconds	55 seconds

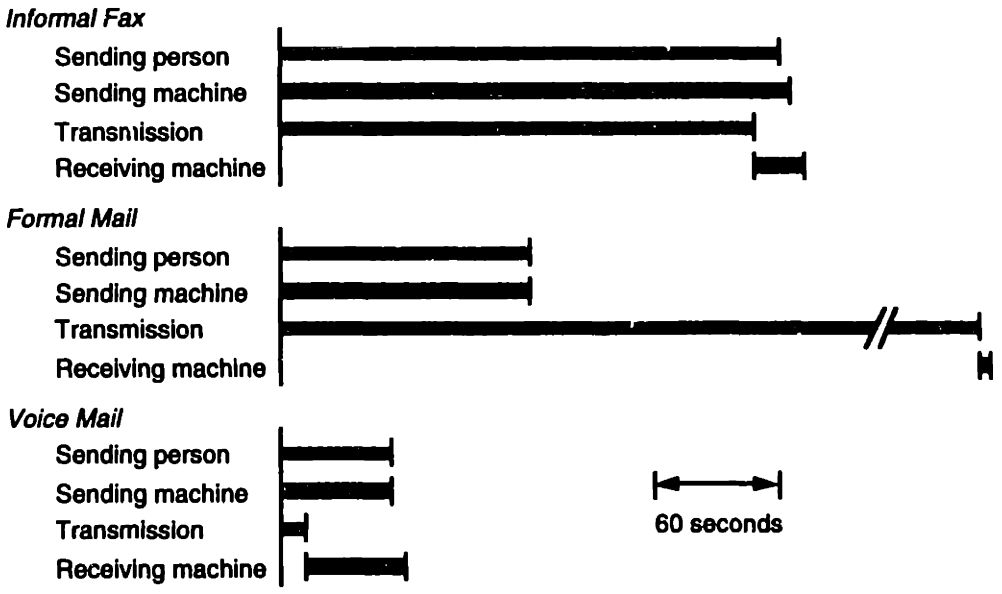


Figure A.20: Transmission times for a 30 second (nominal time) message. This figure uses the transmission parameters from table A.5.

A.2.3 Running the Simulation

When all of the tasks have been defined and all of the offices peopled and all of the equipment put in place, then it's time to simulate. Simulate mode is entered by selecting the "Simulate" radio button in the Main window (see figure A.21). Once in Simulate mode you may run and re-run the simulation, using the many different display windows to view what is happening.

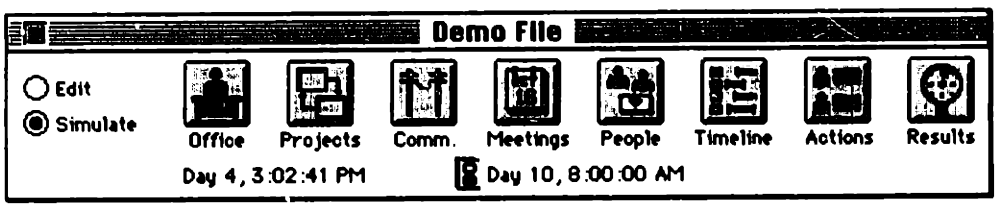


Figure A.21: Simulate mode of the Main Window.

It is possible that you will select Simulate mode only to have the program inform you that it is not possible to switch modes. This happens for one of two reasons: (1) a work task does not have an agent assigned to it, or (2) task dependencies conflict so strongly as to guarantee that the simulation has no chance of successfully terminating. If you receive one of these two errors, you must correct the problem before the program will switch to Simulate mode. Note that the dependency check used for the switch is not as extensive or accurate as the one available in the Verify command (on the other hand, it is quicker). Hence it is possible to run a Simulation that can never terminate.

This section of the User's Manual first describes the different types of viewing windows used to see what is happening in the simulation and then describes how to run the simulation.

There are eight different types of display windows in the simulation. Each of these windows is accessed by clicking on the appropriate icon in the Main window. The People icon is special: it is actually a popup menu of the people in the simulation. Use it to select the agent you wish to examine. You may have more than one agent open at a given time.

Each of the types of display windows are discussed in detail in the following paragraphs.

Projects Window


Projects

The Projects window displays an agent's knowledge of the work tasks in the simulation (see figure A.22). The **View Knowledge By** popup menu selects whose knowledge you examine. If you select "Responsible" from the menu, the computer will display for each work-task the knowledge of that task possessed by the agent responsible for that task. This is a convenient way to view the overall progress of the simulation. Note that there is no concept of knowledge of a management task, only knowledge of work-tasks.

Task knowledge is represented by five separate factors. First of all, if an agent is interested in a task, then a light gray outline of a box will appear inside of the task. This outline will be filled in as the agent gains knowledge of the task. Second, agents have knowledge of how many hours of work have been done on a task. A thin vertical black line marks the agent's knowledge of how complete the task is. It is quite common to know that a task is partially complete, but to not know anything about the task itself. For example, agents responsible for management tasks will ask for status reports on all of the work tasks within their management task.

The other three dimensions of knowledge represented are an agent's understanding of the information generated in a task. Completeness (hours of work) is represented by filling in the knowledge box from left to right. Scope knowledge is represented by filling the box from top to bottom. Thoroughness knowledge is represented by the darkness of the fill of the box; a black fill means 100% thoroughness. When viewing the knowledge by "Responsible", all boxes will be filling in black at 100% scope (because by definition the agents responsible for tasks work at 100% thoroughness and 100% scope at all times).

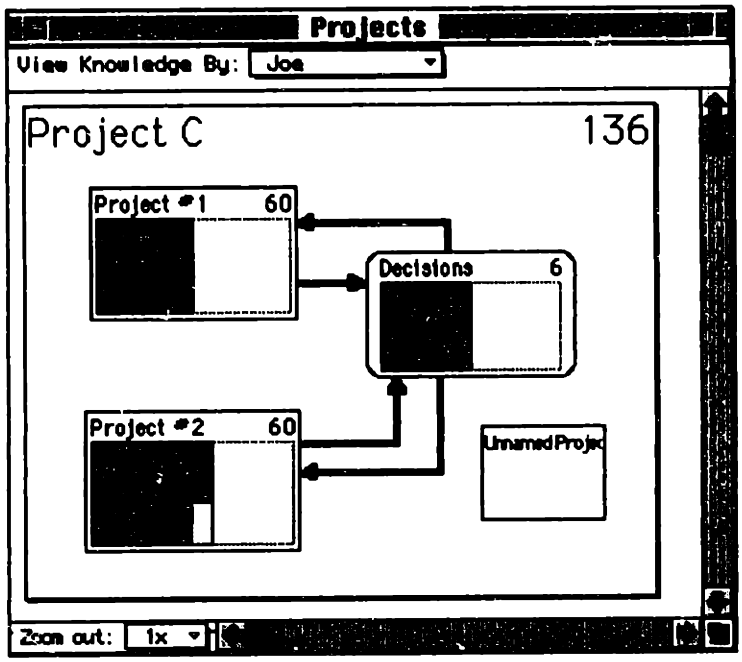


Figure A.22: Projects window.

People & Offices Window


Office

The People & Offices window displays where agents currently are located in the office building and what they are doing (see figure A.23). Agents who are not displayed in the office building may be in one of three states: they may be at home, they may be in transit from one room to another, or they may be in an office which has too many agents in it to display correctly. Note that agents will never enter rooms of type "other".

Hint: When laying out your office building, it is helpful to make each office large enough to hold one or two agents in addition to the agent or agents assigned to that office. Otherwise you will not be able to see when other agents drop by to talk. Since meeting rooms often can have many agents in them, it is useful to make them even larger.

Second Hint: If you select "Zoom Out = 1x", then not only will the agents in the office building be displayed, but their current activity as well.

Transmissions and Communications window


Comm.

The Transmissions and Communications window displays lists of all ongoing transmissions and communications in the simulation (see figure A.24). The double vertical line separating Transmissions from Communications is a horizontal resize bar; drag it to the left or the right to resize the panes.

The Transmissions list shows all transmissions currently active in the simulation. Data listed include source agent, destination agent, transmission channel, expected completion time, and the message content. The current state of the transmission is shown to the right of the completion time. Possible states include: Sending (the sending equipment is transmitting the message), Active (the sending agent is working on the message), and Receive (the receiving equipment is receiving the message).

The Communications list shows all communications currently active in the simulation. Data listed include communication channel, state of conversation, participants in the conversation, and message content. Communication channel stages are: Startup (trying to establish a communication), Begin Conversation (communication has been established), Active Pause (no one is currently talking), and Active Talking (someone is saying something). In Active Talking, the

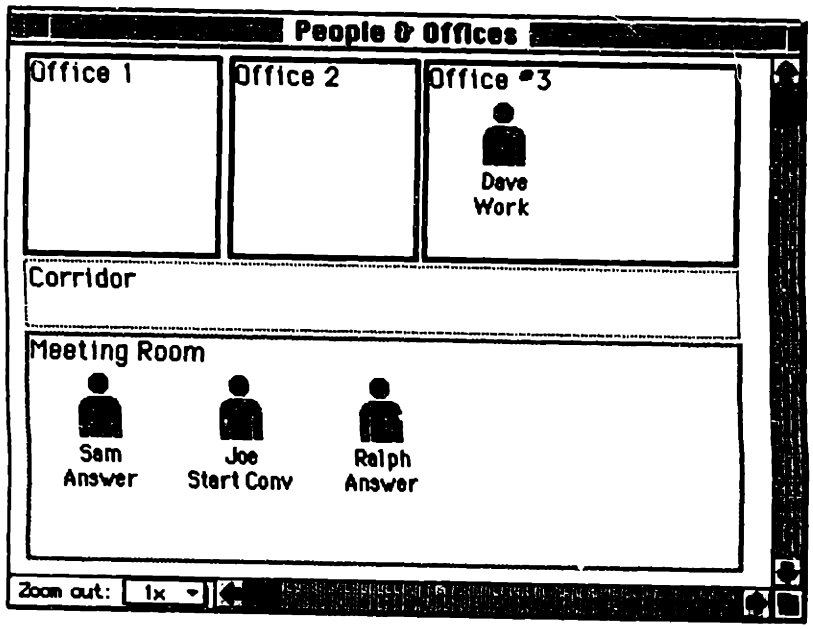


Figure A.23: People & Offices window.

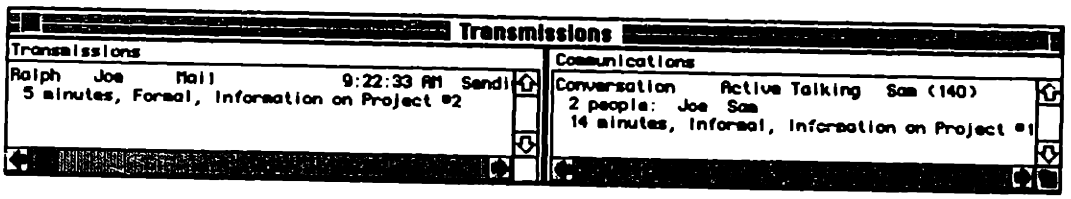


Figure A.24: Transmissions and Communications window

agent who is talking is displayed as well as the urgency of their communication (the number in parenthesis).

In both the Transmission and Communications list, the current message is displayed. The time displayed for the current message is the nominal communication time; the actual communication time depends on the channel.

Meetings Window

The Meetings window displays a complete list of all of the meetings currently scheduled in the simulation (see figure A.25). The list is ordered by date and time. The list also displays meeting duration, location, the communication channel to be used, the type of meeting (currently always "General Meeting"), the task to be discussed, and a list of the agents who will be attending the meeting.

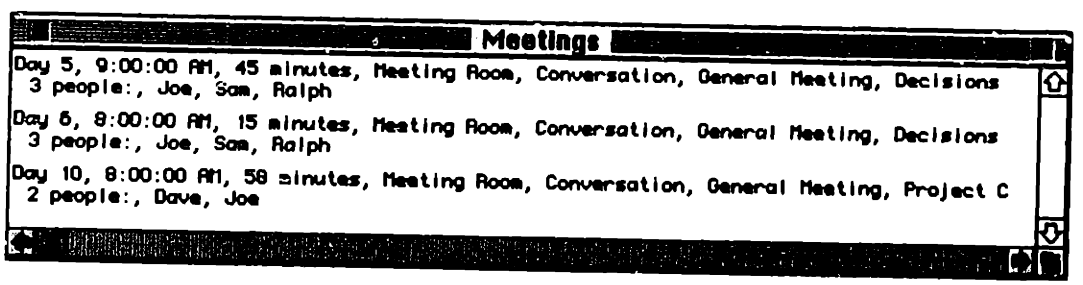


Figure A.25: Meetings window

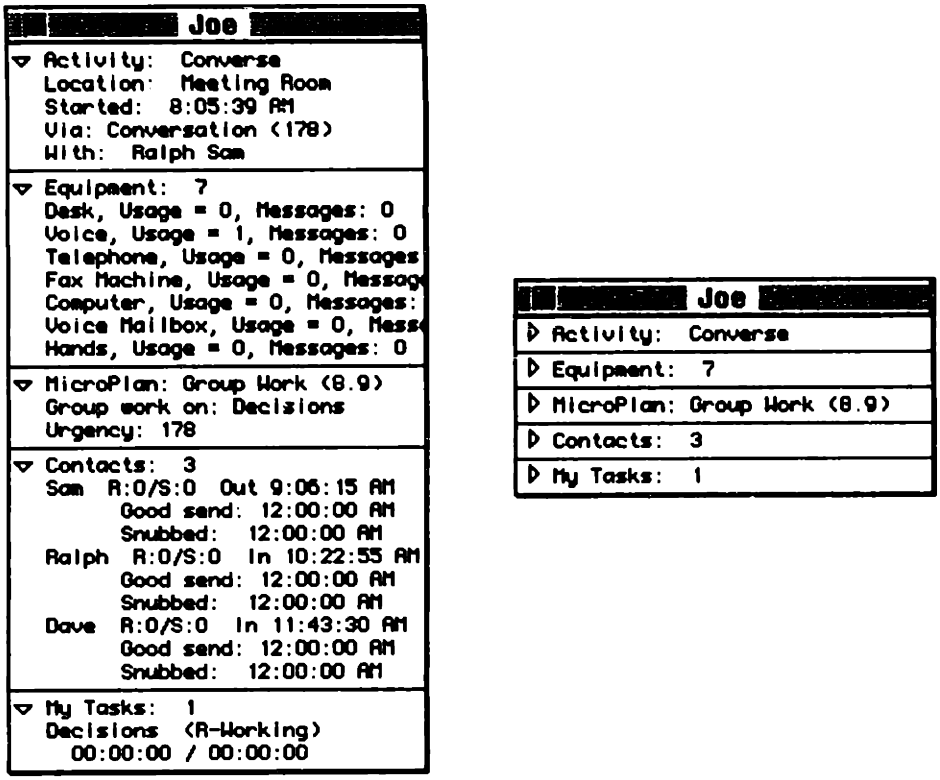


Figure A.26: Personal window — open and closed.

Meetings are removed from the list after they have started or, if they fail to start for some reason, they are removed once the scheduled finish time for the meeting has passed. Note that teleconferences (meetings via a communication channel that does not require the participants to be present in the same room) will not have scheduled meeting rooms. Instead, at the appropriate meeting time the agents will go to their offices.

Personal Windows

A Personal window is used to look at what a specific agent is doing and thinking (see figure A.26). The People icon in the Main window is actually a popup menu listing the people in the simulation. You may have one window open for each agent in the simulation. The window currently consists of five (5) "twisty" panes (there will most likely be more in the future). Current panes are Activity, Equipment, MicroPlan, Contacts, and My Tasks. Each twisty pane has a small triangle on the left side. Click on that triangle to open up that pane and see more detailed information.

The Activity twisty-pane shows what the agent is currently doing. All activities include a start time, others (e.g. "Converse") may contain extra information.

The Equipment twisty-pane shows the state of each piece of equipment that the agent possesses. Usage count on a piece of equipment is how many transmissions and communications are currently utilizing this equipment (this number may not exceed the maximum specified in the Equipment dialog box). Message count is how many messages are currently being held by that piece of equipment.

The MicroPlan twisty-pane shows the current script of the agent and how useful they consider this script to be (utility ranges from around 2.5 to 11). There are approximately thirteen types of scripts available: going home, reading mail, working on a task, contacting a person, sending a request, asking a question, attending meetings, and so forth. Each type of MicroPlan will display additional information about what the agent is currently thinking; for what they are

currently doing you should look at the Activity twisty-pane.

The Contacts twisty-pane displays what the agent has said to each other agent in the simulation and how they've contacted them. The R/S parameters refer to the number of requests they've received from this agent and the number of requests that have been sent to this agent. "In" or "Out" refers to the agents perception of whether or not this agent will be available for communication (have they been seen in their office recently). "Good Send" is the last time a successful transmission was sent to this agent. "Snubbed" refers to the last time this agent refused to answer a conversation request.

The My Tasks twisty-pane displays a list of all tasks that this agent participates on and the task status. The task name is listed first, followed by status in parenthesis. The status can be: Waiting, Working, or Done. If this agent is responsible for the task, an "R-" is prefixed to the status. The two times displayed are hours worked on this task on the current day and hours of work planned for this task for the current day. The plan is only updated at the start of the work day.

Progress Window


Timeline

The Progress window displays *when* work has been accomplished on the tasks within the simulation (see figure A.27). Each day in the simulation is displayed across the top of the window; each task that has had some work done on it is displayed down the left side. As the days go by, more days are added to the top of the window. As more work tasks start, they will be added to the list of tasks.

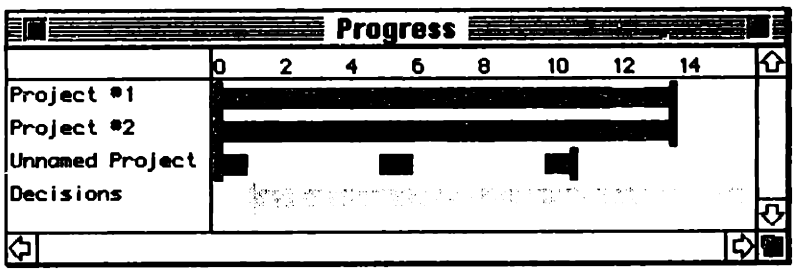


Figure A.27: Progress window.

The amount of work done in a task on a particular day is represented by a shaded rectangle. A black rectangle means that at least 4 hours of work was accomplished on the task on that particular day. Lighter rectangles represent fewer hours of work. The first and last days on a task are marked by slightly taller terminating bars; no terminating bar on the right means that a task is ongoing.

You may zoom the Progress window in and out. When over the window, the cursor will be displayed as a magnifying glass with a plus sign in it. Clicking on the window will zoom in, making each day twice as wide. Holding the option key down will change the cursor to a magnifying glass with a minus sign in it; clicking on the window will now zoom out, making each day half as wide. You can't zoom in and out indefinitely; when you reach the limits, the cursor will change to a magnifying glass with a dot in the middle.

The vertical line separating the names of the tasks from the days may be moved left and right; just click and drag on it.

Actions Window


Actions

The Actions window displays how each agent in the simulation has spent each day (see figure A.28). This window may be zoomed and rearranged in the same way as the Progress window. The radio buttons at the top of the window control what information is displayed. The General category shows the type of activity (work, conversation, transmission,

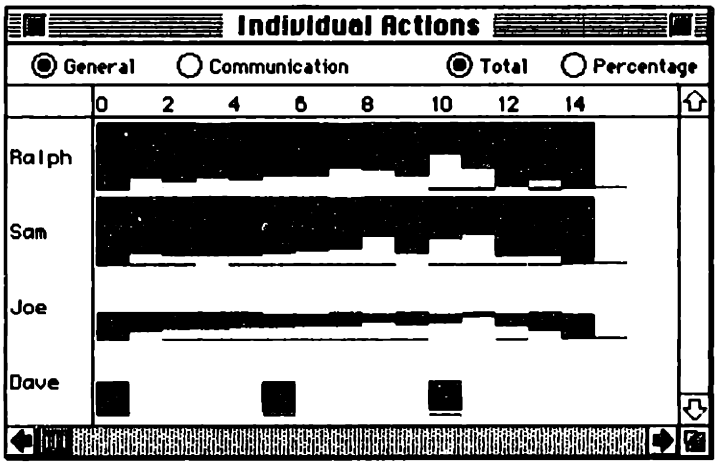


Figure A.28: Individual Actions window. This figure was originally in color.

etc.), whereas the Communication category shows the type of discussion that went on in a conversation (asking questions, gossiping, answering questions, or making decisions).

The colored bars represent how an agent spent his/her time on a particular day. Each horizontal bar (of height one pixel) represents approximately 15 minutes of time. A full-height bar is approximately 9 hours of time. Agents who work less than 9 hours in a single day will not have full-height bars. Agents who come in only once a week (or less frequently) will not have bars displayed for the days they did not come in. Selecting the Percentage radio button will scale each day's activities to fill the entire available space. This button is most often used in conjunction with the Communication radio-button.

This window needs a legend. However, the Results window displays matching summarized information and contains a legend (figure A.29) with the same color-coding scheme as the Actions window.

Examine the row labeled "Sam" in figure A.28. For the first 5 days, Sam spent most of her time working on a task, as represented by the broad patches of green. On Day 2 (and Days 12 and 14) she spent most of the day sitting around wondering what to do; this shows up as a broad patch of red. The light colored band at the bottom of most of her days is the yellow band of communication; it generally is at least 4 pixels high, representing at least an hour per day spent communicating. The other bands on Sam are barely visible.

Results Window

The Results window displays how an agent or all of the agents have spent their time in the simulation (see figure A.29). The categories on the left are a breakdown of the total amount of time spent in the simulation doing various types of activities. The pie chart on the right shows this same list as a percentage of the total amount of time in the simulation. Note that the colors and legend of the pie chart are exactly the same as those used in the Actions window.

The popup menu can select either everyone in the simulation or an individual agent. The radio buttons control whether general information is presented about the types of activities agents have engaged in, or about the types of messages were exchanged in synchronous communications.

Simulation Controls

The simulation is run by the commands within the Simulation menu. The simulation may run in fast mode or slow mode. Slow mode allows you to see everything that is happening in the simulation; fast mode runs things quicker so you can quickly get to results. Stop the simulation

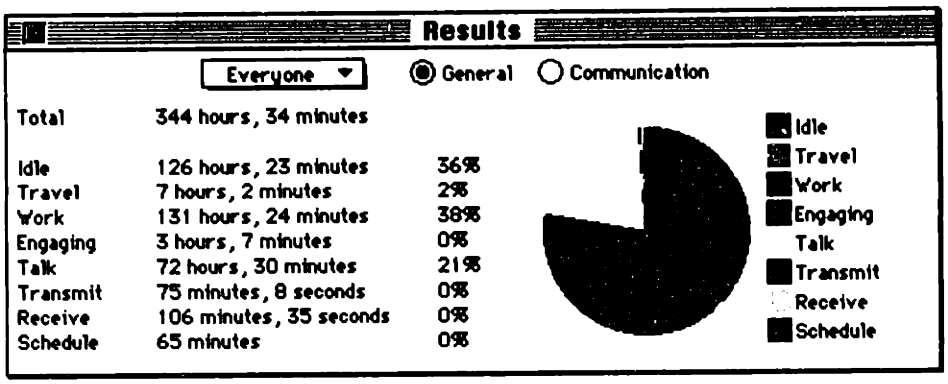


Figure A.29: Results window. This figure was originally in color.

either manually or by setting a breakpoint. Breakpoints are specific times or conditions that stop the simulation when they are reached.

Note: There is no "finish" to the simulation. The simulation will continue to run long after individual have run out of things to do (the hard limit on running time is approximately 50 simulated years).

Running the Simulation: There are three ways to advance time: "Run Slow", "Run Fast", and "Step" (all accessible from the Simulation menu). When you select Run Slow or Run Fast, the simulation will run until you select "Stop" or until it hits a breakpoint. When you select "Step", the simulation will move forward in time until something happens and then stop and redraw the screen.

In Run Slow mode, the simulation advances time by up to 10 simulated minutes and then redraws the screen. This mode allows you to see everything that is happening in the simulation. In Run Fast mode, the simulation runs as fast as it can for about 1/6 of a second and then redraws the screen. This mode allows you to get to the end of the simulation as fast as possible.

Note: At all times, the simulation will run faster with fewer windows open on the screen. Redrawing the windows on the screen takes a fair amount of processor time; to keep the simulation running just as fast as possible, close all windows (but not the Main window).

The "Reset" command in the Simulation menu will set the clock back to Day 0, 12:00:00 A.M. and clear all breakpoints. Use this only after you have finished examining the results of a simulation run because all data stored by the computer from the last run will be discarded. In addition, switching to the Edit mode will automatically reset the simulation.

Working with Breakpoints: A breakpoint stops the simulation when a certain criteria has been met (see figure A.30). There are two types of breakpoints; time-based and progress-based, and three ways to set them. The active breakpoint is displayed at the bottom right of the Main window. A time-based breakpoint consists of a date and time when the simulation will stop. A progress-based breakpoint consists of an agent and a task; the breakpoint stops the simulation when the specified agent knows that the specified task has completed.

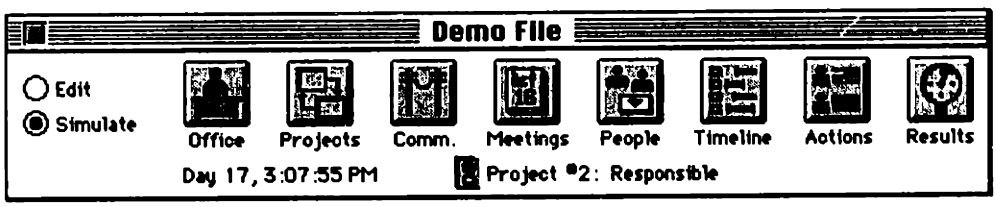


Figure A.30: Triggered progress-based breakpoint.

The status of a breakpoint is shown by a little stoplight next to the breakpoint information. When the green light (the lower one) is lit, the breakpoint is active, but not yet triggered. When the breakpoint "fires" (that is, stops the simulation), the stoplight turns red (the top light). The breakpoint has now triggered and will be deleted as soon as you continue the simulation. It will not stop the simulation a second time. You may clear a breakpoint at any time by selecting "Clear Breakpoint" from the Simulation menu.

The simplest way to set a breakpoint is to select a time-based breakpoint from the "Breakpoint" submenu of the Simulation menu. The time you select is relative to the current time in the simulation. Selecting a time of one day or more will always cause the simulation to stop at the beginning of the day selected. For example, if it is currently Day 4, 4:23 P.M. and you select a breakpoint in one day, the breakpoint will be set for Day 5, 8:00 A.M.

The second way to set a time-based breakpoint is to click on a meeting in the Meetings window and then choose "Set Breakpoint" from the Simulation menu. A time-based breakpoint will be set to the start time of the selected meeting.

A progress-based breakpoint may be set from the Projects window (figure A.22). First, select an agent from the "View Knowledge By" popup menu (or choose "Responsible"). Second, click on any task (work or management). The selected task will highlight. Third, select "Set Breakpoint" from the Simulation menu. The progress-based breakpoint will fire when the agent you have selected knows that the task you have selected has finished. For management tasks, the breakpoint fires when the agent you selected knows that all of the work tasks contained in that management task have completed. Note that it is entirely possible for a progress-based breakpoint to never fire.

A standard use of the progress-based breakpoint is as follows: select "View Knowledge By Responsible" from the popup menu and set a breakpoint on the top-level task. The breakpoint will fire when the agent in charge of the top-level task knows that everything has been completed; this is a fairly good way of deciding when the simulation has finished.

Run-time Factors

There are a lot of formulas and functions used deep within the *DiFS* simulation model that control how agents decide what to do and how long it takes. These can be lumped into two categories: system factors and personal factors. For example, the amount of time it takes to travel from one end of the building to the other end is a system factor ("Travel Time"). How strongly the nagging of other individuals influences a agent's desire to work on a task ("Nagging Bonus") is a personal factor.

All system and personal factors of the simulation may be edited from the "Run-time Factors" dialog box (see figure A.31). Access this dialog box by (1) being in Simulation mode and (2) selecting "Run-time factors" from the Simulation menu.

There isn't space within the User's Manual to explain each of the factors; for this information you will have to refer to the author's thesis. However, a few of the more interesting factors will be mentioned here:

Stochastic: A small, uniformly distributed random number to be added to an agent's assessment of utility. A value of 0.0 means that the simulation is deterministic. I have found that a value of approximately 0.5 works quite well.

Travel Time: How long it takes for an agent to travel from one end of the office to the other end. Note that this does not depend on the size of the office you have drawn; you should put in bigger travel times for bigger offices. By setting this factor to a very large number, you can simulate a group of agents who work in remote sites (be sure to provide methods of teleconferencing!)

Meeting Schedule Time: How long it takes an agent to set up a meeting. Each meeting only needs to be set up once, but if you put a large value in this factor, then an agent who keeps getting interrupted will find it difficult to set up a meeting.

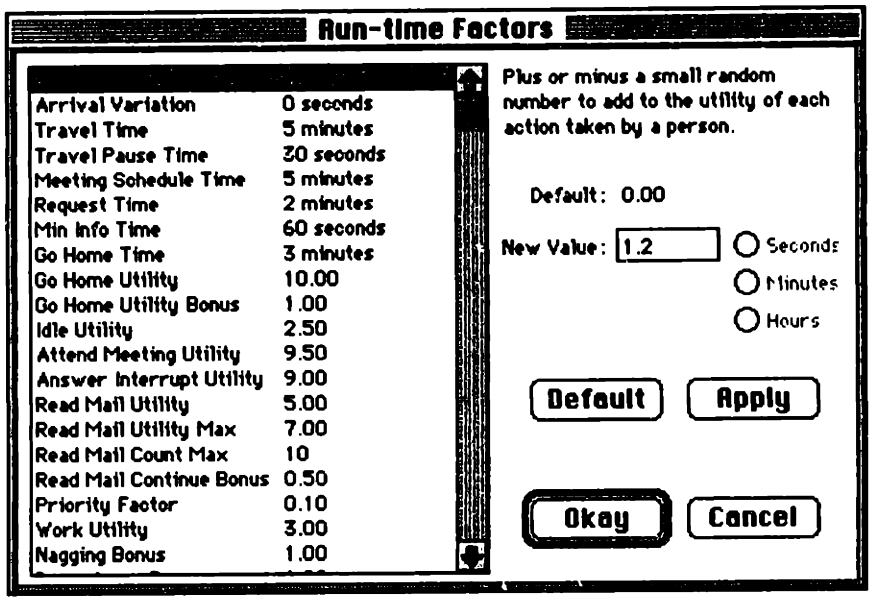


Figure A.31: Run-time Factors dialog box.

Min Info Time: The minimum size of an answer to a request. That is, even if an agent has nothing to say at all, he/she will talk for at least this long when answering a request from another agent.

Meta Meeting Duration: How long a management task meeting will be scheduled for. Normally around an hour.

Gossip Limit: An agent's tendency to chatter on and say nothing of import in a conversation. The larger the gossip limit, the more your agents will stand around and talk about nothing in particular.

Contact Delay: Roughly speaking this is the amount of time that agents will wait between failed contact attempts. That is, if Joe tries to call Sue and doesn't reach her, he's likely to wait about this long before trying to call her again.

A.2.4 Data Storage

The *DiFS* program can generate two types of data files that store results from running the simulation: general simulation statistics and a simulation project trace. Both of these files are in TEXT format. To create these data files, select either "Output File/General" or "Output File/Project Trace" from the File menu. These files may only be written after a simulation run and before the simulation has been reset.

General Data File

The general data file stores both static data and simulation results. Static data are measures of the information-flow model itself. Simulation results are statistics recorded from a single run of the simulation.

The two types of static data recorded are by task and by person. Static task data are a calculation of the total hours of work and communication required for each task, as well as the theoretically maximum amount of communication required for a task if one did not take into account job assignments. Static person data are a calculation of the quantity of work and communication that each agent will have to do.

Simulation results are stored for meetings, agent work, communication and transmission channels, and individual tasks. The number, total duration, and average duration of meetings are recorded. The amount of time spent by each agent on each type of activity (except going home) is recorded, as well as the amount of time in conversation sending each type of message. For each type of communication and transmission channel, the number of usages of that channel, number of usages that succeeded, maximum number of people on the channel, total amount of time, longest usage, and average usage are recorded. Finally, the start and finish dates and work hours of each task are recorded.

Project Trace Data File

The project trace data file stores the quantity of work done in each task on each particular day (in hours). This information is exactly the data that are displayed in the Progress Window of the *DiFS* program.

To facilitate comparing corporate data with simulated data, I wrote a small program called "Timeline" that can read a project trace data file and display a Progress Window. The Timeline program works just the same as the *DiFS* Progress Window with three additions: First, an Options dialog box allows the user to set the gray scale. Normally a black bar represents four or more hours of work in a given day; this may be set to whatever you desire. Second, a "Copy as PICT" command has been implemented in the Edit menu. This copies the displayed image onto the clipboard and thus may be pasted into any program the user desires. Third, more than one project trace may be opened at a time.

Project trace files are tab-delimitated text files, so they may be constructed by the user and read into the Timeline program. The format of the text file is as follows: First, a header block consisting of one or more non-blank lines. Second, a *single* blank line. Third, a line containing the data header. For a project with N tasks, this line should consist of the word "Day" followed by the N names of the tasks as they are to be displayed in the Progress Window. The word "Day" and the N names should be separated by tab characters, not by spaces. Finally, one or more lines containing the day number and the number of hours for each task, separated by tab characters. Note that the actual "Day" number is ignored by the Timeline program; days are assumed to start with 0 and increase by one for each row.

A.3 Controlling *DiFS* with AppleScript

The *DiFS* program can be controlled by high-level events called AppleEvents. AppleScript is a programming language distributed by Apple for the Macintosh that uses AppleEvents to control AppleEvent-aware programs. This section describes how AppleScript can be used to run a controlled experiment using the *DiFS* program.

A.3.1 Simple Example

Subroutine A.1 is a sample AppleScript that calculates the average completion time for an arbitrary *DiFS* project based on five trials. This subroutine requires (1) the "DiFS PPC" program to be running and (2) the desired project to be open with the main window on top. The main window of the desired project must be on top so that the AppleScript can determine the name of the desired project; the name of main window is the same as the name of the project.

Running subroutine A.1 generates the output dialog box shown in figure A.32. For this example, the sample project used is the human-factors experiment project from the author's thesis. The project durations of five trials were averaged to calculate the final result.

The sample program shown in subroutine A.1 can be divided into three parts for interpretation: preliminary setup, running the simulation five times, and the final calculation and display of results.

Subroutine A.1: AppleScript that calculates the average duration of a project.

```
set gExtremeLimit to 200 * days
set gStochValue to 0.5
set gRunningTotal to 0
set gTrialCount to 5

tell application "DiFS PPC"
  tell document (get name of window 1)
    set state to Simulating
    set value of factor "Stochastic" to gStochValue

    repeat gTrialCount times
      reset
      breakpoint on project 1
      with timeout of 1500 seconds
        set sim_result to (extreme maximum time gExtremeLimit)
      end timeout
      set gRunningTotal to (gRunningTotal + (runtime of sim_result))
    end repeat
  end tell
end tell

set sim_result to (gRunningTotal / (gTrialCount * 9 * hours))
display dialog ("Average runtime = " & sim_result & " days") buttons { "Okay" }
```

The setup phase consists of setting a few constants and preparing the simulation. The gExtremeLimit constant is used to tell the simulation to run no longer than 200 days. Simulation runs longer than that are cut off to prevent the computer from hanging. The gStochValue constant is the desired setting of the **Stochastic Factor** of the simulation; it is greater than zero so that run results will be randomized. The gRunningTotal variable keeps a running total of the sum of project durations. The gTrialCount constant is how many repetitions to run.

After specifying constants, the AppleScript asks for the name of the front window of the *DiFS* program and assumes that that name is the name of the document. It then sets the state of the *DiFS* program to "simulating" and sets the value of the **Stochastic Factor** to gStochValue.

A repeat loop for gTrialCount times loops through running the simulation. For each run the simulation is reset and a breakpoint is specified on project 1 (the top-level project). The "extreme maximum time gExtremeLimit" command tells the *DiFS* program to execute the simulation at

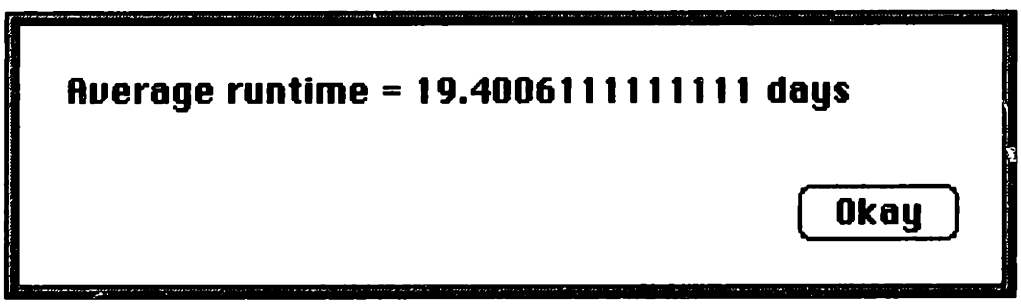


Figure A.32: Result of running subroutine A.1 on the human-factors project.

Table A.6: Resulting data file from running subroutine A.2 on the human-factors project.

Simple test for Sample Data File					
Variations of task Interface Software					
50	19.3	19.1	19.3	19.3	19.1
100	19.1	19.3	19.1	19.5	19.2
150	24.8	24.9	24.7	24.7	24.7

the maximum possible speed until the breakpoint has been hit or until the simulation time reaches gExtremeLimit. The timeout that surrounds this function call prevents the AppleScript from getting impatient and canceling the action; a large project may take 10 or more minutes to execute. The extreme command returns a set of project metrics after the breakpoint has been hit; only the runtime is used.

After the project trials have run, gRunningTotal contains the sum of the project durations, measured in seconds. This number is divided by the number of trials and by the number of seconds in a nine-hour day to get the average number of days for the project duration. This number is displayed in a dialog box and the AppleScript terminates.

A.3.2 Fancier Example

Subroutine A.1 is a sample AppleScript that systematically varies the work hours assigned to a particular task in a design project and runs a number of trials at each work-hour level. Results of the simulation runs are written to a standard text file. This subroutine requires (1) the "DiFS PPC" program to be running and (2) the desired project to be open with the main window on top. The main window of the desired project must be on top so that the AppleScript can determine the name of the desired project; the name of main window is the same as the name of the project.

Table A.6 shows a set of standard results from running subroutine A.2 on the human-factors project from the author's thesis. For this set of trials, the INTERFACE SOFTWARE task was varied between 50, 100, and 150 work hours. Five data runs were recorded at each work-hour level. Each line of data contains the work hours assigned to the INTERFACE SOFTWARE task and the project durations of each of the trial repetitions.

Subroutine A.2 is similar to subroutine A.1, but contains additional statements to open and write to the data file, and to vary the work hours assigned to the project. The output datafile is selected by a standard dialog box using the new file command. This datafile is opened for writing with a open for access command and then written to by a write command.

The AppleScript must specify the precise task in order to vary the work hours. Tasks are specified recursively, so the INTERFACE SOFTWARE task is specified as project "Interface Software" of project 1 of document doc_name. Note that the list of work hours is iterated over using the repeat command.

More complicated AppleScript examples are available from the author, including AppleScripts that run orthogonal array experiments.

Subroutine A.2: AppleScript that systematically varies the work hours of a task and measures the resulting project duration.

```
set gExtremeLimit to 200 * days
set gStochValue to 0.5
set gTrialCount to 5
set gTaskName to "Interface Software"
set gWorkHoursList to {50,100,150}

tell application "DiFS PPC"
  set doc_name to get name of window 1
  set my_doc to a reference to document doc_name

  set outfile to (new file with prompt "Select an output file" default name (doc_name & " Data"))
  set fileref to (open for access outfile with write permission)
  write ("Simple test for " & doc_name & return) to fileref
  write ("Variations of task " & gTaskName & return) to fileref

  tell my_doc
    set state to Simulating
    set value of factor "Stochastic" to gStochValue

    repeat with i in gWorkHoursList
      tell project gTaskName of project 1
        set work hours to (i * hours)
      end tell

      write ((i as string) & tab) to fileref
      repeat gTrialCount times
        reset
        breakpoint on project 1
        with timeout of 1500 seconds
          set sim_result to (extreme maximum time gExtremeLimit)
        end timeout
        set day_count to (runtime of sim_result) / (9 * hours)
        write tab & (day_count as string) to fileref
      end repeat
      write return to fileref
    end repeat
  end tell
end tell
```

Appendix B

Program Description

This appendix describes the implementation of the *DiFS* simulation. The first section outlines the structure of the program. The second section contains a complete description of the rules used to calculate agent behavior. The third section is a brief description of a sorting algorithm that orders tasks by their dependency relationships.

B.1 Program Structure

The program is composed of five layers of C++ objects: core PowerPlant code, visualization code, the information-flow model, the office simulation model, and the agent personal behavior model. Approximately 50,000 lines of code were written for this simulation.[1]

The purpose of this section is to give a brief overview of the program layout so that a person desiring to make small changes has an idea of what to modify. I would like to emphasize that this code was written as a part of the Ph.D. process; it is not commercial quality code and should not be treated as such. There are many places where I would like to tighten the code, rewrite large portions, and even change the core algorithms. I would also like to add two disclaimers: First, if you are unfamiliar with C++ or the concept of programming "objects", then this section may be difficult to understand. Second, this section is a very brief overview; to really understand the software you will need to look at the source code.[2]

This section is divided into a discussion of the basic layout of the software, the information-flow model layer, the simulation loop, and the structure of the agent mental model.

B.1.1 General Code

Conventions

Two conventions are generally followed throughout the code: first, each type of object in the system is encoded in one or more data files which have the same name as the name of the object (the ".cp" files). Each X object has an X.h file that defines the object and an X.cp file containing the X member functions. The second convention is that the prefix of object name normally indicates what layer the object falls within. The standard prefixes are as follows:

L- objects are PowerPlant objects. PowerPlant is the object class system provided with Metrowerks C++; it forms the basis of the user-interface code.

U- objects are utility functions. Some are part of the PowerPlant code; some are part of the *DiFS* simulation code.

[1] Okay, I went a little crazy and counted all of the lines: I wrote 50,015 lines of software that are divided into 329 files. 28,754 lines actually contain C++ code, 10,156 have comments, and 12,249 are blank (some lines contain both code and comments).

[2] Individuals interested in acquiring the source code should contact the author or Prof. Warren P. Seering.

C- objects are user-interface objects that display windows and dialog boxes. C-objects handle user commands and file-system interfaces.

A- objects are the core data layer of the simulation. The information-flow model is composed of A-objects. Some A-objects extend into the simulation layer; for example, the `AWorld` object not only acts as a container for the entire information-flow model, but also contains the event loop logic of the simulation. The information-flow model is described in chapter 2.

T- objects are "transient" objects used in the simulation layer. They are created and destroyed during a simulation run. T-objects represent such concepts as a conversation, an information message, or a specific action selected by an agent. Basic concepts of the simulation are described in chapter 3.

X- and `MicroPlan` objects are specific to the agent behavior model. Each agent's mental model is constructed out of X-objects that specify what the agent knows. `MicroPlan` objects are used to specify the current script that an agent is following. Basic concepts of the agent mental model are described in chapter 4.

Structure

The `CMainApp` object is the top-level object in the system. It handles all events and actions. It dispatches most actions automatically to objects of class `CMainDoc`. Each `CMainDoc` object represents a single project model. A `CMainDoc` object contains an `AWorld` object, a main display window (with icon buttons to dispatch other windows), and a set of editing and simulation windows. The `AWorld` object contains all of the objects that specify the information-flow model and the simulation data.

The `CMainDoc` object coordinates activities between the different display windows, menu commands, and the `AWorld` object. For example, assume that the experimenter modifies the name of one of the tasks (a field in an `AProject` object). This may be done from the work task dialog box (`CWorkDialog`). This dialog is a child of `CTaskView` which belongs to the Task Window (a `CZoomWindow`), which belongs to the `CMainDoc`. A `ChildDirty` message is passed up this chain to indicate that the name change will require redrawing other windows. `CMainDoc` executes the function `CMainDoc::AdjustChildWindows`; this function redraws all windows that may have been affected by the change in the original `CTaskView`.

The `CMainDoc` object executes the simulation code (located in the `AWorld` object). This may be a result of the "Step" menu command that advances the simulation a single time step, or it may be a result of a periodic action from the "Run Slow" or "Run Fast" commands. A single time step is executed by the `AWorld::DoTimeStep` function. The `CMainDoc` object executes one or more `AWorld::DoTimeStep` calls and then calls `CMainDoc::AdjustSimulationWindows` to refresh the display windows.

Finally, the `CMainDoc` object handles the file interface. A `CMainDoc` is always associated with a file (unless newly created, in which case the file is undefined). When the experimenter saves the file (using `Save` or `Save As`), the `CMainDoc` verifies that the file can be created and written, and then executes the `AWorld::Put` function. When opening a file, the `CMainDoc` object verifies that the file is of the correct type and can be read, and then calls a constructor for the `AWorld` object that builds the object out of the stored data.

B.1.2 Information-Flow Model

World Object

The information-flow model is contained within a single `AWorld` object. Many objects have a need to refer to the `AWorld` object; for these the mix-in class `AWorldReference` provides a convenient interface to the elements of the `AWorld` object.

The `AWorld` object contains:

1. A pointer to the top-level management task (`AMetaProject`). This top-level task contains all of the tasks and dependencies in the project model.

2. A list of rooms (`ARoom`) in the office model. An `ARoom` object specifies where the room is located in the office building, the room type (office, meeting, or other), and assigned agents (for an office).

3. A list of agents (`APerson`). The `APerson` object is a subclass of the `AStaticPerson` that contains information-flow model references, and a superclass of the `ARealPerson` that contains the agent mental model.

4. A list of equipment descriptions (`AEquipmentDesc`). An equipment description is an abstract class describing the properties of a particular type of equipment. When the simulation is run, each agent is assigned an instance of each type of equipment (`TEquipment`).

5. A list of asynchronous communication channels (`ATransChannel`). An `ATransChannel` is an abstract description of the properties of a particular type of transmission channel. When the simulation is run, instances of transmission channels (`TTransmission`) will be created and destroyed as required.

6. A list of synchronous communication channels (`ACommChannel`). An `ACommChannel` is an abstract description of the properties of a particular type of communication channel. When the simulation is run, instances of communication channels (`TComm`) will be created and destroyed as required.

7. A pointer to a set of personal behavior factors (`AWorldConstants`).

The `AWorld` object also contains other objects that exist only when the simulation is actually running (T-objects). These objects will be described in the next section.

Task Structure

Tasks within the information-flow model are either `AMetaProject` objects or `AWorkProject` objects. An `AMetaProject` object corresponds to a management task. An `AWorkProject` object corresponds to either a solo work task, a group decision task, or a milestone meeting task.

Tasks are arranged in a tree with `AMetaProject` objects at the nodes and `AWorkProject` objects at the leaves. Other than the top-level task, each task contains a pointer to its parent task (an `AMetaProject` object). Each `AMetaProject` object contains a list of child tasks.

Dependencies between tasks are represented by `ADependency` objects. Each `AWorkProject` contains a list of driving `ADependency` objects and a list of dependent `ADependency` objects. Each `ADependency` object has two pointers; one to the driving `AWorkProject` and one to the dependent.

Because all references to tasks ultimately start with the top-level `AMetaProject`, it is important that the software maintain the integrity of the data pointers at all times. For example, deleting an `AWorkProject` causes all driving and dependent dependencies to be deleted as well. Each of these dependencies upon deletion removes itself from the lists maintained by the driving and dependent tasks. Because it is desirable to be able to undo deletion activities, a `CCutArray` can be used to "unhook" a set of tasks and dependencies so that they can be "re-hooked" in if the experimenter decides to undo the deletion.

`AWorkProject`, `AMetaProject`, and `ADependency` objects store task and dependency parameter data as well as graphical information that determines where the task/dependency is to be drawn on the screen. These objects do not store simulation data; all information about task progress generated during a simulation run is stored in individual agent's mental models. Think of `AProject` and `ADependency` objects as being read-only during a simulation run.

Role Assignment

Agents in the simulation are represented by `AStaticPerson` objects (which are `APerson` objects and `ARealPerson` objects as well). Each `AStaticPerson` contains agent parameter information, graphical information about where the agent is to be drawn, and a list of `ARole` objects.

An `ARole` object specifies the relationship of a single agent to a single task. The two standard roles currently allowed are "responsible" and "participant". The role object contains a pointer to the `AStaticPerson` and a pointer to the `AProject`. Each `AStaticPerson` and `AProject` contains a list of assigned `ARole`. As in the dependency case, care must be taken when adding and deleting agents, tasks, and roles so that the correct `ARole` objects are added/removed and the agent and task lists are correctly updated.

Validation

Simple validation tests may be executed in the simulation using the `Verify` command. One validation test looks for overconstrained dependency relationships. The second validation test checks that each task has at least one assigned responsible agent. A third check, run right before simulation, ensures that each task may start and calculates a best possible start time estimate (this third test is less powerful than the first test, but is quicker to run).

The dependency check (`CTaskView::DoVerify`) is carried out by first extracting a list of all `AWorkProject` objects contained within the top-level `AMetaProject`. Each `AWorkProject` object in turn is requested to advance an internal variable representing the quantity of work done on the task by approximately 10 minutes.[3] The `AWorkProject` object checks with driving `ADependency` objects to calculate maximum possible completion level and advances by no more than that amount. The process is repeated until either all `AWorkProject` objects have completed all required work or until no `AWorkProject` object can advance. Two or more tasks failing to complete indicates an overconstrained dependency loop.

The assignment check (`AWorld::VerifyPeople`) simply scans the role list of each `AProject` object and verifies that there is exactly one "responsible" `ARole` object attached to the `AProject`.

The start-time check (`AWorld::VerifyDependencies`) calculates the estimated best possible start time for each task based on the **Estimate Fudge Factor** provided by the experimenter. This function attempts to calculate the start time of each `AWorkProject` object in turn. Each `AWorkProject` that can calculate its start time in turn attempts to start each of the `AWorkProject` objects that it drives.

The start time of a task may be calculated if the start times of all *active* driving tasks are known. A driving task is active if the downstream task may not be started before the upstream task. If the upstream task's start time is known, then the downstream task's start time may be estimated as the sum of the upstream task's start time and the hours of work in the upstream task required by the dependency multiplied by the **Estimate Fudge Factor**. A task with more than one active driving task sets its start time to the maximum of all of the calculated start times. Note that the start-time estimates do not consider agent role assignments or potential task loop requirements. These start times are optimistic estimates. Agent communications during the simulation correct these estimates.

B.1.3 Running the Simulation

Time Steps

The `AWorld::DoTimeStep` function advances the simulation by a single time step. The actions performed to advance the simulation are as follows:

1. Determine how far to advance the simulation clock. Each `APerson` object is asked when it would like to be awakened. Each `TTransmission` and `TComm` is asked when it would like

[3]The completion level is stored in a dummy variable that is not used during the real simulation run; in simulation, each agent stores task knowledge independent of the `AWorkProject` object.

to be awakened. Breakpoints and simulation time limits are also considered. The next useful event is selected and the clock is advanced to that time.

2. Each `TTransmission` object and `TComm` object is given a `Wakeup` message for the current time. Transmissions and communications use these wakeup messages to determine if something has happened since the last wakeup. For example, a communication representing an on-going conversation may determine that the last speaker has finished talking. If the speaker has terminated, the communication switches to a "between speakers" state.

3. Each `APerson` agent is given a `Wakeup` message for the current time. The agent interprets this message based on the agent's current `TActivity`. For example, if the agent was involved in a conversation where another agent just finished speaking, the agent grabs a pointer to the `TMessage` that the other agent provided and makes a mental note of the available message (to be interpreted shortly). Each agent also logs the current `TActivity` in a `TPersonLog` object.

4. Each `APerson` agent who has decided that he/she would like to consider a new course of action (1) absorbs any received messages (`APerson::HandleMessage`), and (2) selects a new `TActivity` (`APerson::DoWhat`).

   Agents specify the activity they would like to perform by creating a subclass of `TActivity` when asked `APerson::DoWhat`. There are eleven possible activities an agent can select, including `TActivityIdle`, `TActivityTravel`, and `TActivityTalkComm`.

5. Each `APerson` agent who has selected a new activity is given a `APerson::Commence` message. Note that all agents select activities *before* any of them commence; this is done so that agent activity choices are in effect simultaneous.

6. Dead transmissions and communications are removed from the system. A transmission dies when either the `TMessage` is successfully sent or when something goes wrong. A communication dies when there is no agent left who is actively participating in the `TComm`.

Data structures

A variety of data structures are used to represent knowledge, messages between agents, and transmissions and communications. These will each be briefly described.

Task knowledge is represented by a `TKnowledgeBit` data structure. Task knowledge is always represented by three axes: scope, thoroughness, and completion. These are implemented in the simulation by storing knowledge as a two-dimensional array of scope vs. thoroughness. The value stored in each array location represents the completion of the task at that particular level of scope and thoroughness. The scales used for scope and thoroughness are non-uniform and match those presented in pop-up menus to the experimenter, so the representation is relatively compact.

When exchanging task knowledge, agents send `TDeltaBit` data structures to each other. A `TDeltaBit` is the difference between two `TKnowledgeBit` data structures. It represents an incremental increase in task knowledge. This knowledge, when received by another agent is added to the agent's `TKnowledgeBit` understanding. A good graphical example of `TKnowledgeBit` and `TDeltaBit` data structures is figure 4.2 on page 69.

Agents exchange information by sending each other `TMessage` objects via synchronous communications (`TComm`) and transmissions (`TTransmission`). The `TMessage` object has four subclasses: `TMessageGossip`, `TMessageInfo`, `TMessageDecision`, and `TMessageRequest`. All types of message store data on who sent the message, who the intended recipients are, when the message was sent, and when it will be delivered. The specific types of message store information relevant to the particular message. For example, a `TMessageInfo` message contains a pointer to the `AWorkProject` being discussed and a `TDeltaBit` of information.

All TMessage objects are exchanged via either TTransmission or TComm communication channels. These channels are objects that are brought into existence when an agent specifies either a TActivityTransmit action or a TActivityStartComm action. Once in existence, they run themselves and are deleted when they are no longer needed. For example, assume that an agent desires to send another agent a message. The agent specifies a TActivityTransmit action using a transmission channel (ATransChannel) corresponding to inter-office mail. The agent commences the activity which creates the TTransmission, adds it to the global list, and assigns it the task of delivering a TMessageInfo object. After an appropriate interval of simulated time, the TTransmission object informs the agent that the TActivityTransmit action has concluded for that agent. The initiating agent is free to do other activities. Meanwhile, the TTransmission object waits until an appropriate amount of simulated time has passed, and then delivers the TMessageInfo object to the appropriate TEquipment object of the receiving agent.

TComm objects are more complicated than TTransmission objects because the state of the conversation may be changed at any time by agents joining the conversation, leaving the conversation, or interrupting other agents. A TComm object can be in one of two principle states: startup and active. During the startup state, the TComm object periodically attempts to gain the attention of other agents by "ringing" the appropriate piece TEquipment for each receiving agent. Agents respond to the ring by specifying TActivityAnswerComm. When all agents have answered or the ring period ends, the TComm informs all agents that the conversation is active.

An active conversation can be in one of two states: speaking or between speakers. When between speakers, each agent in the conversation specifies a TActivityTalkComm action, an appropriate TMessage to be sent, and an urgency for the message. Highest urgency wins. The TComm holds onto the highest urgency TMessage, switches to the speaking state, and waits the appropriate amount of simulated time. When the correct amount of time necessary to send the message has passed, the TComm switches to the between-speakers state and informs each APerson that a message has been received. Messages are lost whenever a new agent TActivityJoinComm joins the conversation, when a person interrupts the conversation (by specifying TActivityTalkComm while someone else is talking), or when an agent leaves the conversation.

Logging

During the simulation run a variety of types of data are recorded for agents, projects, communications, transmissions, and general world statistics. Each APerson maintains a private TPersonLog that records the amount of time spent in each type of TActivity during the course of each day. Each AWorkProject maintains a private TProjectLog that records the quantity of work done in that task on each simulated day. Each type of TTransmission and TComm also stores data in a log that keeps track of the total number of times a particular communication channel was used, the total number of hours logged using the channel, and other associated data. These data are stored in TTransLog and TCommLog objects respectively.

The TPersonLog objects are stored within each APerson. The remaining logged data is stored within a TWorldLog object. All of this logged data may be dumped to a data file after the simulation run. Some of the data may also be accessed by AppleEvents; in particular, the TProjectLog data is available in this fashion.

B.1.4 Agent Personal Behavior Model

Standard Interface

ARealPerson is a subclass of APerson. All agent knowledge and behavior is channeled through the ARealPerson object. The ARealPerson object receives a message each time something happens in the simulation that can affect the agent. Most of the time these messages are passed directly on to an XMentalModel object that represents the knowledge of the agent.

The following types of messages are sent to the ARealPerson object:

Do What: The agent must decide what to do. The agent selects an activity and returns a corresponding subclass of `TActivity` to specify what he/she will do.

DidXXX: After a `TActivity` finishes, an agent receives an appropriate `DidXXX` message. For example, if an agent attempts to work on an assigned task (`TActivityWork`), the agent receives a `DidWork` message when the work is complete.

Person Left/Entered Room: Whenever an agent sees another agent enter or leave a room, or an agent enters or leaves a room him/herself, one of these messages is sent to the agent. The agent uses this information to update the mental model of when other agents were last seen (`XContact` information).

Transmission Successful/Failed: After sending a transmission, this message is sent indicating whether or not it succeeded. Note that the agent may *believe* that it has succeeded although it may fail later on. For example, an agent may "successfully" send a `TMessageInfo` to another agent via a `TTransmission` corresponding to the inter-office `TTransChannel`, but "successful" in this case only means the message has gone out; there is no way of telling whether or not the message has been received.

TMessage Type XXX Sent/Received: When an agent receives or sends a `TMessage` (of type `TMessageInfo`, `TMessageRequest`, `TMessageDecision`, or `TMessageGossip`), the agent receives a "Message Type XXX Sent/Received" message. The agent uses this message to update his/her mental model.

Mental Model Structure

Each agent contains a single `XMentalModel` object that represents the accumulated knowledge of the agent and also contains the rules that the agent uses to determine what to do (as described in section B.2). The basic structure of the `XMentalModel` object has been described in section 4.4 on page 73.

Each `XMentalModel` object contains lists of the three types of tasks an agent is concerned with: `XMyTask` objects, `XOthersTask` objects, and `XMetaTask` objects. A work task that is the responsibility of this agent is represented by an `XMyTask` object. This object tracks what is known about the task, how much work can be done on the task before more information will be required, and daily planning information. A task that an agent desires information from is represented by an `XOthersTask` object. The `XOthersTask` object stores information about what is known about the task, when the agent responsible for the task has last been contacted for information, and what is needed to be learned about the task. A management task that an agent is associated with is represented by an `XMetaTask` object. It tracks meetings and the status of child tasks.

Dependencies between `XTask` objects are stored as `XDependency` objects. Each `XDependency` is associated with a specific `ADependency` from the information-flow model. The `XDependency` keeps track of information about how this particular dependency is affecting the downstream task. For example, it tracks the best possible completion level of the downstream task based on the current knowledge of the upstream task.

The `XMentalModel` object also contains a list of `XContact` objects. From an agent's perspective, another agent in the simulation is represented by an `XContact` object. The `XContact` object tracks when the agent was last seen, what he/she was doing, and what `XOthersTask` objects and `XSink` objects the contact is responsible for. The `XSink` object connects an `XContact` object to a `XMyTask` object; it tracks when that contact has requested information about the task and what information has been provided.

Scripts

As described in section 4.1.2 on page 66, the agent behavior model is divided into a high-level portion that translates knowledge into an objective and a low-level portion that translates

objectives into particular actions or `TActivity` objects. Within the program, the `MicroPlan` object type is used to represent particular objectives and the scripts that agents follow to carry out those objectives.

There are fourteen types of `MicroPlan` objects, as described in section 4.3, page 71. An agent selects the `MicroPlan` to follow by polling each `XMyTask`, `XMetaTask`, and `XContact` within the `XMentalModel`, as well as some default objectives specified in `XMentalModel`. Each of these mental-model elements may specify one or more `MicroPlan` objects and associated utilities. The `MicroPlan` with the highest associated utility is selected as the new active `MicroPlan`. Note that it is quite possible that no newly specified `MicroPlan` will have a higher utility than the current `MicroPlan`; in this case, the current plan continues as if it had never been interrupted.

Each `MicroPlan` maintains an internal state of where it is in the appropriate `MicroPlan` script (see figures 4.3 and 4.4 on pages 72 and 73). As each `TActivity` specified by the `MicroPlan` finishes, a new `TActivity` object appropriate to the current situation is specified. When the `MicroPlan` runs out of things to do, it is declared dead and the agent is forced to select a new `MicroPlan`.

B.2 Personal Behavior Rules

Agent personal behavior is controlled by 48 factors accessible to the experimenter and a large number of rules and weighting functions contained within the behavior level of the simulation. This section describes the formulas used in the personal behavior model. This section assumes that the reader is familiar with chapter 4, in particular section 4.5 starting on page 78. Figure 4.9 on page 81 may prove useful in providing an overview of the flow of the agent utility calculations.

Table B.1 shows the default settings for the agent and simulation factors. The first eight factors are simulation defaults; the remainder are personal behavior factors.

Subroutines B.1 through B.24 are a pseudo-code version of the personal behavior logic of an individual agent. They are written to look a little like C code, but have been simplified from the actual implementation to improve clarity. Each personal behavior factor is **emphasized**. Arrays and regular variables are not emphasized; note that all arrays are initialized at the start of each function by a 1-indexed list of values. For example, foo = { 10, 20, 30 } gives foo[2] = 20.

A few generic functions are used in the subroutine definitions: Min and Max take the minimum or maximum of all of the values specified. Interpolate is used to select from a range of values:

$$\text{Interpolate}(x, x_1, y_1, x_2, y_2) = \begin{cases} y_1 & x \le x_1 \\ y_1\left(\frac{x_2 - x}{x_2 - x_1}\right) + y_2\left(\frac{x - x_1}{x_2 - x_1}\right) & x_1 < x < x_2 \\ y_2 & x \ge x_2 \end{cases} \quad \text{(B.1)}$$

The Clip function is given by $\text{Clip}(x, x_1, x_2) = \text{Interpolate}(x, x_1, x_1, x_2, x_2)$. The Clip function simply restricts the range of its value. The Round function fixes a value to the nearest integral unit.

A few final comments on utility values are in order. The utility of doing nothing (i.e., being idle) is given by **Idle Utility**. It is normally defined to be 2.5; this represents the lower bound of what will be considered useful. Many functions return a utility of 0.0 to indicate that a particular objective is unsuitable; hence lowering the **Idle Utility** to below 0.0 will cause problems. The upper bound is normally approximately 10.0; this is also the default value of **Go Home Utility**. All normal objective utility values will range between these extremes. Finally, if an agent is trying to decide what to do and that agent has an objective that is in progress, then the utility of the objective in progress is equal to its originally calculated utility plus one. This effect can always be seen by opening up an agent window and looking at the currently selected objective; the displayed utility value will always be one greater than the originally calculated utility value.

Table B.1: Default agent and simulation factor values.

Factor Name	Default Value	Factor Name	Default Value
Stochastic Value	0.0	Group Work Bonus	3.5
Arrival Variation	0 minutes	Meeting Multiplier	1.25
Travel Time	5 minutes	Meta-Meeting Utility	4.0
Travel Pause Time	30 seconds	Meta-Meeting Duration	59 minutes
Meeting Schedule Time	5 minutes	Request Utility	5.0
Request Time	1 minute	Request Spacing	2 hours
Minimum Info Time	1 minute	Request Wait Time	16 hours
Go Home Time	3 minutes	Request Nag Bonus	0.5
Go Home Utility	10.0	Request Dependency Bonus	1.1
Go Home Utility Bonus	1.0	Answer Utility	4.6
Idle Utility	2.5	Answer Nag Bonus	1.0
Attend Meeting Utility	9.5	Valid Topic Bonus	0.4
Answer Interrupt Utility	9.0	Gossip Limit	10 minutes
Read Mail Utility	5.0	Gossip Max Utility	5.0
Read Mail Utility Max	7.0	Gossip Min Utility	1.0
Read Mail Count Max	10	Talking Bonus	2.25
Read Mail Continue Bonus	0.5	Talking Loss Per Minute	0.1
Priority Factor	0.1	Contact Delay	30 minutes
Work Utility	3.3	Transmit Delay	15 minutes
Nagging Bonus	1.0	Update Status Time	9 hours
Dependency Bonus	1.0	Start Talking Bonus	1.8
Planned Bonus	0.5	Request Meeting Proximity	1.5
Continue Work Bonus	0.55	Answer Meeting Proximity	1.5
Schedule Meeting Bonus	0.5	Estimate Fudge Factor	1.25

B.2.1 Task, Sink, and Source Utilities

Task, Sink, and Source calculations are used to find the intrinsic value of working on a particular task, asking for information, or providing information to another agent. They do not take into account the convenience of doing work or communicating.

Subroutines B.1 and B.2 are used to account for the effects of an agent frequently requesting information ("nagging") or a dependent task requiring information from an upstream task. Both of these subroutines return a value between 0 and 1; this value is later scaled in other functions.

There are a number of logical improvements that could be made to these function blocks; the biggest is to take into account the number of influences in the aggregate function rather than just the largest influence. For example, the TaskNagFactor() function could return a value proportional to the product of the largest SinkNagFactor() and the number of different sinks that are requesting information about this task.

Subroutines B.3 and B.4 calculate the intrinsic utilities of working on an assigned task and of scheduling a meeting for a management task. Note that value of working on an assigned task is affected by the number of outstanding requests for information (TaskNagFactor), the number of dependent tasks that are waiting for information (TaskDependencyPush), and the basic priority of the task itself.

Subroutines B.5 and B.6 calculate the intrinsic utilities of responding to a request for information and of asking for information from another agent about a particular task. To send information to another agent, it helps to have received at least one outstanding request, but it is also reasonable to know that the receiving agent would like periodic status updates. Note that there is a strong compulsion to send out information as soon as a task has completed. This compulsion to announce the completion of a task helps managers in the simulation keep up to date as to whether or not the entire project has finished.

Subroutine B.1: Nagging factors for a single sink connection to a task, and for the total effect of all sink connections to a particular task.

```
SinkNagFactor() {
  RecentRequests = { 0.5, 0.75, 1.0 };          // Nag factor of n recent requests
  OldRequests    = { 0.1, 0.3, 0.6 };             // Nag factor of n old requests

  if ( no requests ) return 0.0;
  if ( request within last 3 minutes ) return 1.0;
  recency = Interpolate( time since last request, 0 minutes, 0.0,
                                                  10 hours, 1.0 );
  requests = Min( number of requests, 3 )
  return Interpolate( recency, 0.0, RecentRequests[requests],
                               1.0, OldRequests[requests] );
}

TaskNagFactor() {
  return Max( of all SinkNagFactor() for this task );
}
```

Subroutine B.2: Pushing influence of a task that is dependent upon the task in question, and for the total effect of all of the tasks that are dependent upon the task.

```
DependencyPush() {
  TaskPriorityEffect = { -0.2, -0.1, 0.0, 0.1, 0.2 };     // Normal priority is 3 (or 0.0)

  value = 1.0 − ( Possible downstream work / 20 hours )
              + TaskPriorityEffect[ Downstream task priority ];
  return Clip( value, 0.0, 1.0 );
}

TaskDependencyPush() {
  return Max( of all DependencyPush() for this task );
}
```

Subroutine B.3: Intrinsic utility of working on an assigned task.

```
MyTaskUtility() {
  return Work Utility + TaskNagFactor() × Nagging Bonus                  // B.1
                      + TaskDependencyPush() × Dependency Bonus          // B.2
                      + Task Priority × Priority Factor;
}
```

Subroutine B.4: Intrinsic utility of scheduling a meeting for a management task.

```
MetaTaskUtility() {
  return Meta-Meeting Utility
           + Task Priority × Priority Factor;     // Task priority ranges from 1 to 5
}
```

Subroutine B.5: Intrinsic utility of providing information about a task to another agent.

```
SinkUtility() {
  if ( EITHER there is at least one outstanding request,
         OR ( this agent has requested updates,
                AND new information is available,
                AND ( EITHER this task has completed,
                       OR the time since the last update > Update Status Time )))
    return Answer Utility + SinkNagFactor() × Answer Nag Bonus;        // B.1
  else
    return 0.0;
}
```

Subroutine B.6: Intrinsic utility of requesting information from another agent about a task.

```
SourceUtility() {
  if ( EITHER no more information is needed,
         OR the time since the last request was sent < Request Spacing,
         OR I don't believe the task has started )
    return 0.0;

  bonus = TaskNagFactor() × Request Nag Bonus                          // B.1
        + TaskDependencyPush() × Request Dependency Bonus;             // B.2

  if ( I know there is information available )
    return Request Utility + bonus;

  value = Interpolate( time since last info received, 0 seconds, 0.0,
                                                  Request Wait Time, Request Utility );
  return value + bonus;
}
```

The utility of requesting information from another agent is designed to increase as time goes by. The assumption is that the other agent has been working on the task and hence information is more likely to be available as time passes (although the actual working hours of the other agent are not taken into consideration). A lower bound on request frequency (**Request Spacing**) is used to prevent agents from continuously requesting updates.

B.2.2 Solo Objectives

Solo objectives are goals that are pursued by an agent working alone. This section contains a discussion of the objectives that do not involve communication with other agents. The next two sections contain discussions of agent communication.

The two most basic objectives are to waste time and to go home. Wasting time always has a utility equal to **Idle Utility**. The utility of going home is given by subroutine B.7. Note that the utility of going home is zero before the designated departure time and increases monotonically after that. With the default factor values, agents typically leave promptly at the designated departure time. To permit some "slop" in this time, one can set **Go Home Utility** to 2.5 (the **Idle Utility**) and **Go Home Utility Bonus** to a number such that all agents leave within an hour or so (a value of 8.0 would work).

Subroutine B.7: Utility of going home.

```
GoHomeObjective() {
  if ( current time > normal departure time )
    return Go Home Utility + time in hours past departure time × Go Home Utility Bonus;
  else
    return 0.0;
}
```

Subroutine B.8: Utility of attending a meeting.

```
AttendMeetingObjective() {
  if ( I have a scheduled meeting,
      AND time until meeting start < 5 minutes,
      AND the meeting hasn't finished yet )
    return Attend Meeting Utility;
  else
    return 0.0;
}
```

The utility of attending a meeting is specified simply by the **Attend Meeting Utility** (subroutine B.8). Note that the agent model does not take into consideration the purpose of the meeting or how far the agent will have to travel in order to show up for the meeting. The utility of this objective is a function only of when the meeting will occur.

An agent may receive one or more interrupts at any given time in the simulation. The utility of answering each of the possible interrupts is calculated by the AnswerInterruptObjective function (subroutine B.9). To prevent agents from continuously interrupting each other, an agent already in a conversation will only answer an interrupt if he/she knows that the interrupting agent has a higher rank than anyone in the current conversation. A logical extension to this model would be to allow an agent to answer a telephone call or some other communication channel where caller rank can't be established, with the restriction that the agent would limit such interruptions to cases where an important phone call is expected. This extension requires some method of communicating the idea of a "brief" interruption to other agents in the conversation.

An agent may always read his/her mail (i.e., asynchronous communications). The value of this objective is given by subroutine B.10. The objective value is limited to considering the quantity of currently visible mail. A future extension would be to have the agent "thumb through" the mail and evaluate the importance of reading particular messages.

An agent may perform work on each of the individual work tasks he/she is responsible for. The value of doing this work is given by subroutine B.11. This objective function must be evaluated for each of the individual work tasks an agent is responsible for. Note that there is a strong bonus for continuing to work on a task that the agent was just working on; this is necessary because normally work is broken up into 10 to 15 minute chunks of time.

An agent responsible for a group decision task or a management task periodically needs to schedule meetings for those tasks. The value of these objectives is given by subroutines B.12 and B.13. The actual value of these objectives tends to be rather high because they only happen sporadically. Note that these functions return two values: the utility of scheduling a meeting and the proposed duration of the meeting to be held.

Subroutine B.9: Utility of answering an interrupt from an outside source. Note that this utility is dependent upon whether or not the agent can identify the interrupting agent.

```
AnswerInterruptObjective() {
  if ( I'm in a conversation,
       AND the interrupting person does not outrank
           the people in the current conversation )
    return 0.0;

  if ( I can tell who is interrupting )
    value = Start Talking Bonus + Max( of all SourceUtility() for that person,
                                       of all SinkUtility() for that person,
                                       Gossip Max Utility );          // B.6,B.5
  else
    value = Start Talking Bonus + Gossip Max Utility.

  return value;
}
```

Subroutine B.10: Utility of reading a piece of mail. Note that the particular piece of mail is not identified, just the quantity of available mail.

```
ReadMailObjective() {
  count = number of messages in my current location;
  if ( count == 0 )
    return 0.0;

  value = Interpolate( count, 1, Read Mail Utility,
                               Read Mail Count Max, Read Mail Utility Max);
  if ( last activity was reading mail )
    value += Read Mail Continue Bonus;
  return value;
}
```

Subroutine B.11: Utility of working on an assigned work task.

```
SoloWorkObjective() {
  if ( more information from driving tasks is needed )
    return 0.0;

  value = MyTaskUtility();                                          // B.3
  if ( work done today < work planned for today )
    value += Planned Bonus;
  if ( last activity was working on this task )
    value += Continue Work Bonus;
  return value;
}
```

Subroutine B.12: Utility of scheduling a meeting to discuss a group decision task. Only the agent responsible for the group decision task schedules meetings.

```
ScheduleDecisionMeetingObjective() {
  if ( EITHER this task already has a scheduled meeting ,
         OR no work can be done )
    return 0.0;

  if ( this task is a milestone task ) {
    if ( all dependencies have not been satisfied )
      return 0.0;
    duration = Task work hours × Meeting Multiplier;
  }
  else {
    duration = Maximum possible work time × Meeting Multiplier;
    Round( duration, upwards to integral number of hours ≤ four);
  }

  value = MyTaskUtility() + Schedule Meeting Bonus;          // B.3
  return value, duration;
}
```

Subroutine B.13: Utility of scheduling a meeting of the agents assigned to a management task. Only the agent assigned as responsible for the management task will schedule meetings. Note that no meeting will be scheduled if only a single agent is assigned to the management task.

```
ScheduleTeamMeetingObjective() {
  if ( work is in progress in at least one task within this meta-task,
       AND this meta-task has specified periodic team meetings,
       AND no meeting is currently scheduled ) {
         duration = Meta-Meeting Duration;
         value = MetaTaskUtility();                          // B.4
         return value, duration;
  }
  else
    return 0.0;
}
```

Subroutine B.14: An agent's perception of the probability of successfully sending a transmission to another agent. A transmission will fail if the receiving equipment is unavailable. The most common value is 1.0, which corresponds to 100% chance of success.

```
TransmitProbability() {
  if ( the last time I tried to transmit failed )
    return Interpolate( time since last transmission, 2 minutes, 0.0,
                                                    Transmit Delay, 1.0 );
  else
    return 1.0;
}
```

B.2.3 Communication Objectives

Communication objectives are objectives pursued by an agent who wishes to communicate with another agent. It is assumed that the two agents are not currently in conversation; if they are, then objectives from the next section (Conversational Objectives) are used.

Three functions are used to help evaluate how easy it will be to communicate with another agent and how advantageous that communication will be. The perceived probability of successfully transmitting an asynchronous message to another agent is calculated in subroutine B.14. It affects the likelihood an agent will attempt to send a request or information asynchronously. The perceived probability of an agent successfully contacting another agent is given by subroutine B.15. It affects the likelihood that an agent will attempt to synchronously contact another agent. The final routine, subroutine B.16, calculates how soon a meeting will be held to discuss a particular task. This influences an agent's desire to send requests or information about a particular task, because the agent assumes that there will be opportunity to do that during the scheduled meeting.

An agent has two primary reasons for communicating with another agent: sending information and sending requests. A sink of information is another agent who needs information about a particular task. Subroutine B.17 is the utility of sending information to a particular sink. This objective must be evaluated for each sink that an agent has. A source of information is another agent who provides information about a particular task. Subroutine B.18 is the utility of sending a request for information to a source. This objective must be evaluated for each source that an agent has. Note that both the sink and source objectives take into account the probability of successfully transmitting information and the proximity of a meeting that will provide an opportunity for discussion with the target agent.

An agent typically has many sources and sinks of information. However, often these sources and sinks correspond to just a few other agents. Hence it is sometimes advantageous for an agent to directly contact another agent rather than to send information or requests. Subroutine B.19 calculates the value of directly contacting another agent. Note that this value is dependent upon (1) the most important thing an agent has to say, and (2) the perceived ease of contacting the other agent. The number of things that an agent has to say is not included, although it does play a part in the choice of the communication channel.

B.2.4 Conversational Objectives

Conversational objectives are only selected when an agent is in direct conversation with one or more other agents. There are four conversational objectives: doing group work, saying information, saying a request, or gossiping. These objectives are only evaluated for agents who are in the conversation; objectives dealing with agents outside of the conversation are communication objectives (section B.2.3).

Long conversations extract a toll on their participants. Subroutine B.20 is a function that

Subroutine B.15: An agent's perception of the probability of successfully contacting another agent. This is strongly dependent on whether or not both agents are in the same room.

```
ContactProbability() {
  if ( my location == her location ) {              // She and I are in the same office
    if ( she is tied in a higher ranking conversation that I can't join )
      return 0.0;
    else if ( the time since I was last snubbed by her < Contact Delay )
      return Interpolate( time since I was snubbed, 5 minutes, 0.0,
                                                    Contact Delay, 0.8 );
    else
      return 1.0;
  }
  else {                                            // I don't know where she is currently
    if ( EITHER current time of day > her normal going home time,
         OR my location == her office )
      return 0.0;
    else if ( my last attempt to contact her failed )
      return Interpolate( time since last contact attempt, 5 minutes, 0.0,
                                                          Contact Delay, 0.8);
    else
      return 0.8;
  }
}
```

Subroutine B.16: An agent's perception of the proximity of a meeting that will discuss a particular task. It is used when deciding whether or not to contact an agent or send information/requests. A meeting more than 6 hours away is assumed to have no effect.

```
MeetingProximity() {
  return Interpolate( time until next meeting about this task, 0 minutes, 1.0,
                                                               6 hours, 0.0 );
}
```

calculates the ongoing cost of being in a conversation. Note that initially the cost is actually a bonus. With time, the bonus steadily drops until it is a liability.

An agent in the proper setting with the correct group of other agents may conduct work on a group decision task. The value of this objective is given by subroutine B.21.

Subroutines B.22 and B.23 calculate the utility of saying information to a sink or requesting information from a source in a conversation. Each of these must be evaluated for all possible sinks and sources in the current conversation. Note that these functions take into account the validity of a particular source or sink to the conversation at hand. In a scheduled team meeting, valid conversation topics are restricted to tasks that are subtasks of the management task. In a scheduled group decision meeting, valid conversation is limited to (1) group work, (2) requests about tasks that drive the group decision tasks, and (3) answers to those requests.

The final conversational objective is gossip (subroutine B.24). The utility of gossiping in a conversation is a function of the duration of the conversation and the quantity of gossip that has already occurred in the conversation.

Subroutine B.17: Utility of sending information about a task to an agent who is not currently in conversation with this agent.

```
SendInformationObjective() {
  value = SinkUtility()                                          // B.5
        - MeetingProximity() × Answer Meeting Proximity;         // B.16
  return value × TransmitProbability();                          // B.14
}
```

Subroutine B.18: Utility of sending a request about a task to an agent who is not currently in conversation with this agent.

```
SendRequestObjective() {
  value = SourceUtility()                                        // B.6
        - MeetingProximity() × Request Meeting Proximity;        // B.16
  return value × TransmitProbability();                          // B.14
}
```

Subroutine B.19: Utility of contacting an agent for the purpose of exchanging information or requests. Note that contact will be attempted only if this agent has at least one thing to say to or request of the other agent.

```
ContactObjective() {
  value = Max( of all SendInformationObjective() for this person,   // B.17
               of all SendRequestObjective() for this person );     // B.18
  if ( value > Gossip Max Utility )
    return ( value + Start Talking Bonus ) × ContactProbability();  // B.15
  else
    return 0.0;
}
```

Subroutine B.20: The cost of staying in a conversation as a function of how long the conversation has lasted. This bonus will be added to the utility of speaking in the conversation.

```
ConversationDurationCost() {
  value = Talking Bonus;
  if ( EITHER this is an unscheduled conversation,
         OR current time > scheduled meeting finish )
    value -= duration in minutes × Talking Loss Per Minute;
  return value;
}
```

Subroutine B.21: Utility of working on a group decision task.

```
GroupWorkObjective() {
  if ( I'm in a conversation,
      AND EITHER this conversation is part of a meeting about this task,
               OR this is an unscheduled conversation,
      AND all of the people necessary are present ) {
        value = MyTaskUtility() + Group Work Bonus;                    // B.3
        if ( last activity was working on this task )
          value += Continue Work Bonus;
        if ( this conversation is part of a meeting about this task,
            AND the meeting has not finished )
          value += Planned Bonus;
        return value;
  }
  else
    return 0.0;
}
```

Subroutine B.22: Utility of saying a piece of information to another agent while in a conversation with that agent. The validity of the conversation topic depends upon whether or not the current conversation is a scheduled meeting.

```
SayInformationObjective() {
  value = SinkUtility() + ConversationDurationCost();          // B.5, B.20
  if ( task is a valid conversation topic )
    value += Valid Topic Bonus;
  return value;
}
```

Subroutine B.23: Utility of requesting information about a task from another agent while in a conversation with that agent. The validity of the conversation topic depends upon whether or not the current conversation is a scheduled meeting.

```
SayRequestObjective() {
  value = SourceUtility() + ConversationDurationCost();        // B.6, B.20
  if ( task is a valid conversation topic )
    value += Valid Topic Bonus;
  return value;
}
```

Subroutine B.24: Utility of gossiping in a conversation.

```
GossipObjective() {
  value = Interpolate( gossip in conversation so far, 0 minutes, Gossip Max Utility,
                                                    Gossip Limit, Gossip Min Utility );
  value += ConversationDurationCost();                                  // B.20
  return value;
}
```

B.3 Matrix Partitioning

A collection of tasks and dependencies can be put into matrix form by assigning each task to a row and matching column (e.g., row 5 and column 5). A dependency between two tasks is represented by filling in a matrix cell where the row of the cell is the index of the dependent task and the column of the cell is the index of the driving task. This matrix formulation graphically displays task interactions in a compact package.

To improve the legibility of the matrix, it is normally put in a lower-triangular form or a close approximation to a lower-triangular form. This is accomplished by sorting the order in which the tasks appear in the matrix rows and columns. The sorted matrix form is not unique, but it does allow an experimenter to quickly pick out serially dependent tasks, parallel tasks, and tasks that are interdependent.

The purpose of this section is to describe the algorithm used by the *DiFS* simulation to sort the matrix. This algorithm is different from the ones described by Gebala [32]. It has the advantage that it is relatively easy to program and behaves roughly as $O(N^2)$ where N is the number of tasks in the system.

The basic idea of the algorithm is as follows:

1. Select any task x from the set of tasks.
2. Sort the task list into a *before* set and an *strongly after* set such that each task in the *before* set directly or indirectly drives x, and so that no task in the *strongly after* set drives any task in the *before* set. Task x is part of the *before* set.
3. Divide the *before* set into two subsets, an *interdependent* set and a *strongly before* set so that each task in the *interdependent* set is directly or indirectly driven by x and that no task in the *interdependent* set drives any task from the *strongly before* set.
4. The tasks in the *interdependent* set form a loop, and hence can be removed from consideration. Repeat steps 1 through 3 for the *strongly before* set and the *strongly after* set.

Figure B.1 is an illustration of how the algorithm works on a small sample project consisting of 6 tasks. Task A is selected as the initial pivot task, and placed in the *before* set. The first step is to identify all tasks that directly drive Task A; in this case, Tasks D and E. They are moved into the *before*. Now each of Tasks D and E are checked against the remaining tasks to see if they are driven by one of them. It turns out that Task C drives one of them (actually, both of them), so Task C is moved into the *before* set. Finally, Task C is checked against each of the remaining tasks; neither B nor F drive Task C, so the first step is complete. At this point we have checked that none of the tasks in the *after* set (B and F) drive any of the tasks in the *before* set, and we have also established that each of the tasks in the *before* set drive Task A either directly or indirectly.

The next step in the sorting process is to reverse the preceding actions on the *before* set and look for tasks that are driven by Task A (as opposed to driving Task A). Task D is identified as driving Task A, and then Task E is driven by Task D. This divides the *before* set into two pieces; a *strongly before* set and a *interdependent set*. Each task in the *interdependent* set has been identified as both driving and being driven by Task A. Each task in the *strongly before* drives directly or indirectly the tasks in the *interdependent* set.

The process may now be repeated recursively by sorting the tasks in the *strongly before* and *strongly after* sets.

This sorting algorithm is used by the *DiFS* simulation when displaying the Project Grid window. My experience has been that the sorting algorithm takes a negligible amount of time even when working with projects that contain over 100 tasks.

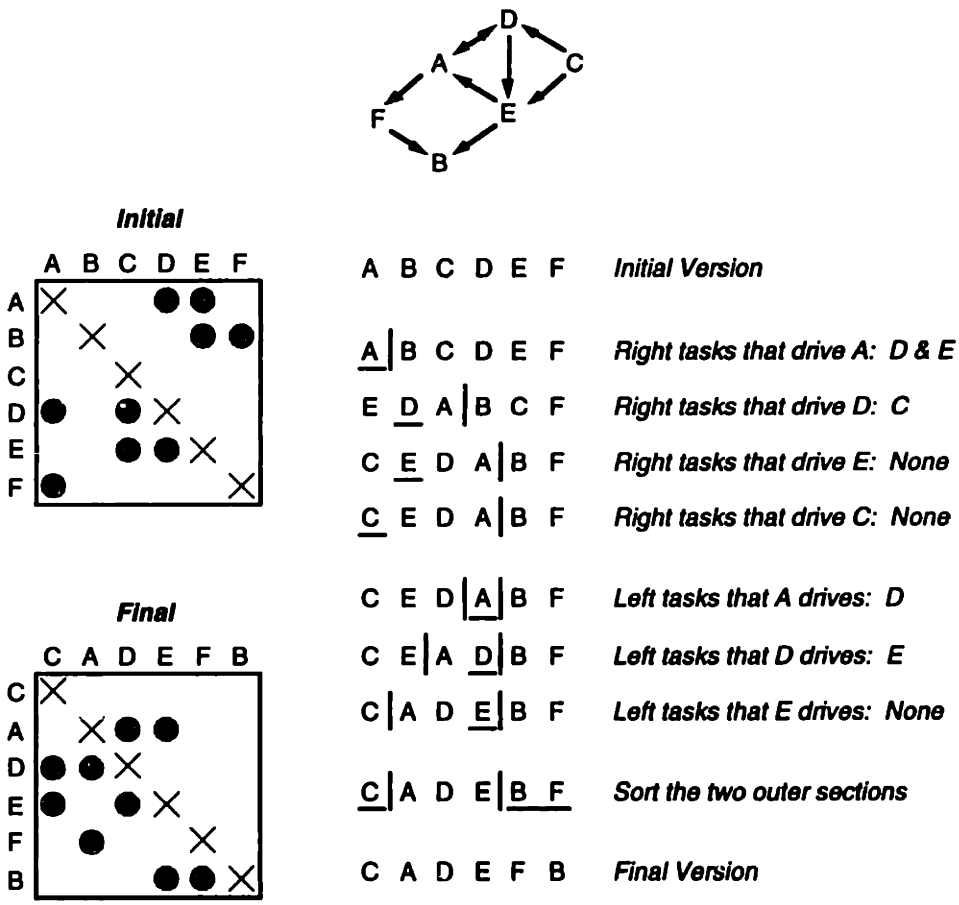


Figure B.1: Sorting a sample project.

Appendix C

Statistical Calculations

The purpose of this appendix is to describe the statistical calculations used in this thesis. The statistics used in calculating effects from the orthogonal array experiments are just unusual enough that it seems appropriate to describe the specific formulas used. For the reader unfamiliar with statistics, I suggest a good introductory book such as Larsen and Marx [42]. Orthogonal array experiments are described in detail by Phadke [61]; this appendix presupposes the reader is familiar with the basics of orthogonal arrays. The two statistics books that I highly recommend for any experimenter are Box, Hunter, and Hunter [8] and Mead [58].

C.1 Orthogonal Array Experiments

Suppose you wish to run a *DiFS* experiment and examine the effects of varying three separate factors at each of three levels. To run a full factorial experiment on this case would take $3^3 = 27$ trial runs. By assuming that the factors are linearly independent of each other, an orthogonal array experiment may be used. In this case, using the first three columns of the L_9 (3^4) array will suffice. The L_9 array looks like:

Expt.	Column			
No.	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Assume that factor A is assigned to column 1, factor B to column 2, and factor C to column 3. Let y_i be the data recorded on the i-th trial. The normal approach to calculate the effects of the different factors is to assume an additive model. That is, we assume that:

$$y_i = \mu + a_j + b_k + c_l + \epsilon_i \tag{C.1}$$

where there are $N = 9$ trials and μ is the overall mean ($\mu = (1/N)\sum_{j=1}^{N} y_i$). The factor effects a_j, b_k, and c_l are obtained by looking in the table and selecting appropriate values for j, k, and l based on the current trial. For example, if $i = 5$ then $j = 2$, $k = 2$, and $l = 3$. The final value, ϵ_i is the error associated with running the trial. It is assumed to be small and normally distributed with zero mean.

Each of the a_j, b_k, and c_l may be estimated from the data by averaging the rows that they appear in. That is,

$$\begin{array}{lll} a_1 = (y_1 + y_2 + y_3)/3 - \mu & a_2 = (y_4 + y_5 + y_6)/3 - \mu & a_3 = (y_7 + y_8 + y_9)/3 - \mu \\ b_1 = (y_1 + y_4 + y_7)/3 - \mu & b_2 = (y_2 + y_5 + y_8)/3 - \mu & b_3 = (y_3 + y_6 + y_9)/3 - \mu \\ c_1 = (y_1 + y_6 + y_8)/3 - \mu & c_2 = (y_2 + y_4 + y_9)/3 - \mu & c_3 = (y_3 + y_5 + y_7)/3 - \mu \end{array} \tag{C.2}$$

Note that the estimates are simple linear combinations of the experiments.

The variance associated of the error term can also be estimated because not all of the columns in the orthogonal array were used. Each factor being estimated contributes $p - 1$ degrees of freedom, where p is the number of levels of the factor being estimated. The grand mean μ contributes another degree of freedom, so in an experiment with N trials and q factors being estimated at p levels each, the error has $N - 1 - q(p - 1)$ degrees of freedom. In our example, $N = 9$, $p = 3$, and $q = 3$. Hence there are 2 DOF remaining to estimate the error.[1] An estimate of the error variance S_E^2 is given by

$$S_E^2 = \frac{1}{N - 1 - q(p-1)} \sum_N (y_i - \hat{y}_i)^2 \tag{C.3}$$

where $\hat{y}$ is the estimated value of each y_i as given by the sum of the grand mean and each of the factor effects. This is an estimate of the variance of the error ϵ from equation C.1.

The error associated with each of the factor effects may be estimated. Note that equation C.2 shows that each factor effect is linear combination of three trial runs. Each trial run has variance S_E^2 associated with it, so the factor variance will be one third as large as the trial variance. In other words, the factor variance S_F^2 is given by

$$S_F^2 = S_E^2 \left(\frac{p}{N}\right) \tag{C.4}$$

where the ratio N/p is referred to as the replication number of the factor. This is the number of trials that were run at each particular setting of the factor.

The value S_F is known as the *standard error*. For a replication number of at least 10 or so, a range of $\pm 2S_F$ is roughly a 92% confidence interval for any of the factor effects.[2] Hence many plots of factor effects place error bars equal to $\pm 2S_F$ to indicate the range in which that actual value is likely to fall.

C.2 Slope Fitting with Orthogonal Arrays

A number of the experiments run in this thesis rely on fitting a straight-line approximation to a set of data. These data are generated using a set of trials based on an orthogonal array. This section describes how to calculate slopes and errors associated with fitting a straight line to a set of data.

Given an orthogonal array A and a vector of trial results Y, we can calculate the effect of each factor in the array assuming a linear fit between trial levels and result. That is, the data is assumed to take the form

$$y_i = \mu + \sum_{j=1}^{q} a_{ji}\beta_j + \epsilon_i \tag{C.5}$$

where μ is the grand mean and there are q different factors being estimated. The effect of each factor is given by its slope β_j multiplied by the appropriate entry in the orthogonal array A. The error associated with each measurement is assumed to be given by ϵ_i, a normally distributed random variable with zero mean.

[1] In larger trials, often the error estimate is augmented by sweeping in the DOF from non-significant factors.

[2] Specifically, if the factors are linearly independent, then a $100(1 - \alpha)\%$ confidence interval for the factor value is given by $\pm t_{\alpha/2,n-1} S_F$ where t is a Student t distribution with $n - 1$ degrees of freedom.

To estimate each factor β_j, we can directly solve the matrix equation. Before we do this, we must write the orthogonal array as a true orthogonal array; the inner product of any two columns i and j should be zero unless $i = j$. For the standard set of orthogonal arrays, this only requires subtracting off a constant factor (for the L_9 array, subtract 2 from each entry). The basic equation in matrix form is then

$$\vec{y} = A\vec{\beta} + \epsilon \tag{C.6}$$

where the matrix A is the orthogonal array modified to be truly orthogonal. Assuming that A is of full rank and neglecting ϵ, we can write

$$\vec{\beta} = (A'A)^{-1}A'\vec{y} \tag{C.7}$$

where A' is the transpose of A. The advantage of using an orthogonal array should be clear from this equation: the term $(A'A)^{-1}$ is diagonal when A is orthogonal. For example, $\vec{\beta} = \frac{1}{6}A'\vec{y}$ for the L_9 array.

Each factor estimated contributes one DOF and the grand mean contributes another DOF; hence the degrees of freedom remaining to estimate the error term ϵ are $N - 1 - q$ where N is the number of experiments run and q is the number of factors estimated. The error variance may be calculated as

$$S_E^2 = \frac{\|\vec{y} - A\vec{\beta}\|}{N - 1 - q} \tag{C.8}$$

or the magnitude of the difference vector divided by the associated degrees of freedom. For example, $S_E^2 = \frac{1}{5}\|\vec{y} - A\vec{\beta}\|$ for our example using the L_9 array because only three of the four columns are being used.

Each individual β_i is a linear combination of experimental trials as shown in equation C.7. The estimated error variance for each trial is S_E^2. Because the trials are independent, the variance of the calculated β_i is a linear combination of the errors from the individual trials. The actual error variance for a given factor β_i may be shown to be

$$S_i^2 = (A'A)_i^{-1}S_E^2 \tag{C.9}$$

S_i is the standard error for the i-th factor. Calculating S_i^2 is made simpler if A is orthogonal. For example, $S_i^2 = \frac{1}{6}S_E^2$ for each estimated factor of the L_9 array.

The final step in calculating slopes and error bounds is to scale them correctly for the value being estimated. Normally an orthogonal array copied out of a book (e.g., [61]) has integral element values (the sample L_9 array has values of 1, 2, and 3). Estimating slopes normally requires both the factor value and the factor standard error to be scaled to an appropriate range.

Bibliography

[1] Valerie Abbott, John B. Black, and Edward E. Smith. The representation of scripts in memory. *Journal of Memory and Language*, 24:179–199, 1985.

[2] Tarek Abdel-Hamid and Stuart E. Madnick. *Software Project Dynamics: An Integrated Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[3] Robert P. Abelson. Psychological status of the script concept. *American Psychologist*, 36(7):715–729, July 1981.

[4] Beth Adelson. Modeling software design within a problem-space architecture. In Sandra L. Newsome, W.R. Spillers, and Susan Finger, editors, *Design Theory '88*, pages 56–80, 1989. Proceedings of the 1988 NSF Grantee Workshop on Design Theory & Methodology.

[5] Thomas J. Allen. *Managing the Flow of Technology*. The MIT Press, Cambridge, MA, 1977.

[6] Apple Computer, Inc. *AppleScript Language Guide*, 1993. Published by Addison-Wesley Publishing Company.

[7] Gordon H. Bower, John B. Black, and Terrence J. Turner. Scripts in memory for text. *Cognitive Psychology*, 11:177–220, 1979.

[8] George E. P. Box, William G. Hunter, and J. Stuart Hunter. *Statistics for Experimenters*. John Wiley & Sons, Inc., 1978.

[9] Frederick P. Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Publishing Company, 1975. Reprinted with corrections, January 1982.

[10] Louis L. Bucciarelli. An ethnographic perspective on engineering design. *Design Studies*, 9(3):159–168, July 1988.

[11] Richard M. Burton and Børge Obel. *Designing Efficient Organizations*, volume 7 of *Advanced Series in Management*. Elsevier Science Publishers, B.V., Amsterdam, The Netherlands, 1984.

[12] Kathleen Carley, Johan Kjaer-Hansen, Allen Newell, and Michael Prietula. Plural-soar: A prolegomenon to artificial agents and organizational behavior. In Masuch and Warglien [56], chapter 4, pages 87–118.

[13] Kathleen M. Carley and Michael J. Prietula. ACTS theory: Extending the model of bounded rationality. In *Computational Organizational Theory* [14], chapter 4, pages 55–87.

[14] Kathleen M. Carley and Michael J. Prietula, editors. *Computational Organizational Theory*. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey, 1994.

[15] K.M. Carley and A. Newell. On the nature of the social agent. Presented at the American Sociological Association Annual Meeting, August 1990. Washington, D.C.

[16] Tore Christiansen. The virtual design team—using simulation of information processing to predict the performance of project teams. CIFE Working Paper 15, Stanford University, July 1992.

[17] Geoff Cohen. *The Virtual Design Team: An Information Processing Model of Design Team Management.* PhD thesis, Stanford University, 1992.

[18] Geoff Cohen and Raymond E. Levitt. The virtual design team: An object-oriented model of information sharing in project design teams. In Luh-Maan Chang, editor, *Preparing for Construction in the 21st Century*, pages 348–353. ASCE, 1991. Proceedings of the Construction Congress '91.

[19] Michael D. Cohen, James G. March, and Johan P. Olsen. A garbage can model of organizational choice. *Administrative Science Quarterly*, 17(1):1–25, March 1972.

[20] Kevin Crowston. Modelling coordination in organizations. In Masuch and Warglien [56], chapter 9, pages 215–234.

[21] Kevin Crowston. Evolving novel organizational forms. In Carley and Prietula [14], chapter 2, pages 19–38.

[22] Andreas Drexl. Scheduling of project networks by job assignment. *Management Science*, 37(12):1590–1602, December 1991.

[23] James S. Dyer, Peter C. Fishburn, Ralph E. Steuer, Jyrki Willenius, and Stanley Zionts. Multiple criteria decision making, multiattribute utility theory: The next ten years. *Management Science*, 38(5):645–654, May 1992.

[24] Salah E. Elmaghraby and Jerzy Kamburowski. The analysis of activity networks under generalized precedence relations (GPRs). *Management Science*, 38(9):1245–1263, September 1992.

[25] Steven D. Eppinger. Model-based approaches to managing concurrent engineering. In *International Conference on Engineering Design.* ICED 91, August 27–29 1991. Zürich.

[26] Steven D. Eppinger, Daniel E. Whitney, Robert P. Smith, and David A. Gebala. Organizing the tasks in complex design projects. In *ASME 2nd International Conference on Design Theory and Methodology*, pages 39–46, September 16–19 1990. Chicago, Illinois.

[27] Steven D. Eppinger, Daniel E. Whitney, Robert P. Smith, and David A. Gebala. A model-based method for organizing tasks in product development. *Research in Engineering Design*, 6:1–13, 1994.

[28] Henry B. Eyring. *Evaluation of Planning Models for Research and Development Projects.* D.B.A. thesis, Harvard University, June 1963.

[29] Jay W. Forrester. *Industrial Dynamics.* Productivity Press, Cambridge, Massachusetts, 1961.

[30] Jay R. Galbraith. *Organization Design.* Addison-Wesley Publishing Company, 1977.

[31] Bezalel Gavish and Hasan Prikul. Algorithms for the multi-resource generalized assignment problem. *Management Science*, 37(6):695–713, June 1991.

[32] David A. Gebala and Steven D. Eppinger. Methods for analyzing design procedures. In *Design Theory and Methodology*, volume 31, pages 227–233. ASME, 1991.

[33] David A. Gebala and Steven D. Eppinger. Modelling the impact of organizational structure on design lead time and product quality. Working Paper #3301-91-MS, Alfred P. Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, May 1991.

[34] Forrest B. Green and Ronald M. Zigli. Pert probability: Network paths and completion time. *Project Management Journal*, pages 54–59, June 1984.

[35] Abbie Griffin and John R. Hauser. Patterns of communication among marketing, engineering and manufacturing—a comparison between two new product teams. *Management Science*, 38(3):360–373, March 1992.

[36] Crispin Hales. *Analysis of the Engineering Design Process in an Industrial Context.* Gants Hill Publications, Winnetka, IL, second edition, 1991. Ph.D. dissertation from the University of Cambridge, Department of Engineering, 1987.

[37] John R. Hauser and Don Clausing. The house of quality. *Harvard Business Review*, pages 63–73, May 1988.

[38] Ralph Katz. The effects of group longevity on project communication and performance. *Administrative Science Quarterly*, 27:81–104, 1982.

[39] Viswanathan Krishnan, Steven D. Eppinger, and Daniel E. Whitney. Aggressive overlapping: Accelerating product development by advance information exchange. Working document, Massachusetts Institute of Technology.

[40] Viswanathan Krishnan, Steven D. Eppinger, and Daniel E. Whitney. Towards a cooperative design methodology: Analysis of sequential decision strategies. In *Design Theory and Methodology*, volume 31, pages 165–172. ASME, 1991.

[41] Theresa K. Lant. Computer simulations of organizations as experiential learning systems: Implications for organization theory. In Carley and Prietula [14], chapter 9, pages 195–215.

[42] Richard J. Larsen and Morris L. Marx. *An Introduction to Mathematical Statistics and its Applications.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

[43] Bibb Latané, Kipling Williams, and Stephen Harkins. Many hands make light the work: The causes and consequences of social loafing. *Journal of Personality and Social Psychology*, 37(6):822–832, 1979.

[44] Sang M. Lee and David L. Olson. A zero-one goal programming approach to multi-project scheduling. *Project Management Journal*, pages 61–71, June 1984.

[45] R. Lesh and K. M. Ragsdell. The design environment simulator. In *Advances in Design Automation*, pages 467–472. ASME Design Technology Conference, 1988.

[46] Raymond E. Levitt. Superprojects and superheadaches: Balancing technical economics of scale against management diseconomies of size and complexity. *Project Management Journal*, pages 82–89, December 1984.

[47] Raymond E. Levitt, Geoffrey P. Cohen, John C. Kunz, and Clifford I. Nass. The "virtual design team": Using computers to model information processing and communication in organizations. CIFE Working Paper 16, Stanford University, July 1992.

[48] Raymond E. Levitt, Geoffrey P. Cohen, John C. Kunz, Clifford I. Nass, Tore Christiansen, and Yan Jin. The "virtual design team": Simulating how organization structure and information processing tools affect team performance. In Carley and Prietula [14], chapter 1, pages 1–18.

[49] Raymond E. Levitt, Clive L. Dym, and Yan Jin. Knowledge-based support for concurrent, multidisciplinary design. CIFE Working Paper 10, Stanford University, January 1991.

[50] Raymond E. Levitt, Yan Jin, and Clive L. Dym. Knowledge-based support for management of concurrent, multidisciplinary design. *AI EDAM*, 5(2):77–95, 1991.

[51] Zhiang Lin. A theoretical evaluation of measures of organizational design: Interrelationship and performance predictability. In Carley and Prietula [14], chapter 6, pages 113–159.

[52] Thomas W. Malone. Modeling coordination in organizations and markets. *Management Science*, 33(10):1317–1332, October 1987.

[53] James G. March and Herbert A. Simon. *Organizations*. Blackwell Publishers, Cambridge, Massachusetts, second edition, 1993.

[54] James G. March and Roger Weissinger-Baylon, editors. *Ambiguity and Command*. Pitman Publishing, Inc., Marshfield, Massachusetts, 1986.

[55] Michael Masuch and Perry LaPotin. Beyond garbage cans: An AI model of organizational choice. *Administrative Science Quarterly*, 34:38–67, 1989.

[56] Michael Masuch and Massimo Warglien, editors. *Artificial Intelligence in Organization and Management Theory*. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, The Netherlands, 1992.

[57] Edward H. McMahon. Evaluation of group design in engineering. In *Design Theory and Methodology*, volume 31, pages 235–240. ASME, 1991.

[58] R. Mead. *The Design of Experiments*. Cambridge University Press, 1988.

[59] G. Pahl and W. Beitz. *Engineering Design: A Systematic Approach*. Springer-Verlag, 1984. Original German edition published in 1977, English edition edited by Ken Wallace.

[60] Kenneth Alan Pasch. *Heuristics for Job-Shop Scheduling*. PhD thesis, Massachusetts Institute of Technology, 1988. Also available as Technical Report #1036, MIT Artificial Intelligence Laboratory, Cambridge, MA 02139.

[61] Madhav S. Phadke. *Quality Engineering Using Robust Design*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989. Copyright by AT&T Bell Laboratories.

[62] Michael Pidd. *Computer Simulation in Management Science*. John Wiley & Sons Ltd., West Sussex, England, third edition, 1992.

[63] Donald A. Pierre. *Optimization Theory with Applications*. Dover Publications, Inc., New York, 1986.

[64] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, 1988.

[65] Stuart Pugh. Design activity models: Worldwide emergence and convergence. *Design Studies*, 7(3):167–173, July 1986.

[66] Douglas T. Ross. Structured Analysis (SA): A language for communicating ideas. *IEEE Transactions on Software Engineering*, SE-3(1):16–34, January 1977.

[67] Marvin E. Shaw. *Group Dynamics*. McGraw-Hill Book Company, third edition, 1981.

[68] Herbert A. Simon. A behavioral model of organizational choice. *Quarterly Journal of Economics*, 69:129–138, 1957.

[69] Herbert A. Simon. *Administrative Behavior*. The Free Press, New York, NY, third edition, 1976.

[70] Robert P. Smith and Steven D. Eppinger. Identifying controlling features of engineering design iteration. Working Paper #3348-91-MS, Alfred P. Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, December 1991.

[71] Robert P. Smith, Steven D. Eppinger, and Amarnath Gopal. Testing an engineering design iteration model in an experimental setting. Working Paper #3386-92-MS, Alfred P. Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, February 1992.

[72] Ivan D. Steiner. Models for inferring relationships between group size and potential group productivity. *Behavioral Science*, 11:273-283, July 1966.

[73] John D. Sterman. Misperceptions of feedback in dynamic decision making. *Organizational Behavior and Human Decision Processes*, 43:301-335, 1989.

[74] Donald V. Steward. The design structure system: A method for managing the design of complex systems. *IEEE Transactions on Engineering Management*, EM-28(3):71-74, August 1981.

[75] Donald V. Steward. *Systems Analysis and Management Structure, Strategy, and Design.* Petrocelli Books, New York, 1981.

[76] Sarosh N. Talukdar and Steven J. Fenves. Towards a framework for concurrent design. In *Proceedings of the Special Session for Concurrent Design, ASME Winter Annual Meeting, San Francisco, CA*, December 11-14 1989.

[77] David G. Ullman. *The Mechanical Design Process.* McGraw-Hill, Inc., 1992.

[78] Karl T. Ulrich and Steven D. Eppinger. *Product Design and Development.* McGraw-Hill, Inc., 1995.

[79] Harko Verhagen and Michael Masuch. TASCCS: A synthesis of Double-AISS and Plural-Soar. In Carley and Prietula [14], chapter 3, pages 39-54.

[80] F. von Martial. *Coordinating Plans of Autonomous Agents.* Number 610 in Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, Germany, 1992.

[81] Lawrence M. Wein and Philippe B. Chevalier. A broader view of the job-shop scheduling problem. *Management Science*, 38(7):1018-1033, July 1992.

[82] Daniel E. Whitney. Designing the design process. *Research in Engineering Design*, 2:2-13, 1990.

[83] Jerome D. Wiest and Ferdinand K. Levy. *A Management Guide to PERT/CPM.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, second edition, 1977.

[84] Kippling Williams, Stephen Harkins, and Bibb Latané. Identifiability as a deterrent to social loafing: Two cheering experiments. *Journal of Personality and Social Psychology*, 40(2):303-311, 1981.

Index