Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

KNOWLEDGE-BASED MODELING AND SIMULATION COMPONENTS

Jeff Kaminski
Cindy Cosic
Gary Strohm
Joyce Kepner
Jim Bycura
Carnegie Group, Inc.
Five PPG Place
Pittsburgh, PA 15222

ABSTRACT

Carnegie Group, Inc. has developed a series of programming modules that can be used separately or *en masse* in the areas of modeling and simulation. This paper briefly describes the modules and the domains in which they can potentially be applied. The paper also includes a description of an actual application involving four of the six software packages.

1. INTRODUCTION

This paper discusses six knowledge-based modeling and simulation components developed and refined by Carnegie Group, Inc. over the past two years. Built in two phases, these components are:

1. **Phase One:**
   - Object-Oriented Programming
   - Simpak™
   - Statpak™
   - Graphpak™
2. **Phase Two:**
   - Timepak™
   - Modelpak™

These six tools extend the existing power of the object-oriented paradigm found in Carnegie Group's Knowledge Craft™ tool to the areas of modeling and discrete simulation. The systems lend themselves to both static and dynamic modeling techniques, addressing physical resources and materials, intangible operations and processes, and always-present temporal constraints. The components also satisfy the user's need for the generation, collection, and display of data and statistics to analyze the systems they are modeling, thus allowing them to develop and confirm the optimum configuration of the various aspects of their systems.

From the physical inception of the first four components in 1987 to the recent completion of the final two packages, these separate yet related tools have undergone a number of enhancements to attain their current level of utility and versatility. This paper will address each component in turn, describing first the initial four components created in phase one of product development, and then the two most-recent installments to the family of tools implemented in phase two. Each component will be described with respect to its prominent features and some of its more useful and interesting characteristics.

Potential applications of the products developed in the two phases will also be cited. In addition, this paper will describe a 1988 deployment of the first four components in a system developed to model and simulate a printed circuit board manufacturing plant. Finally, the paper will conclude with a description of the possible uses and benefits of the tools when considered and deployed both independently and collectively.

2. PHASE ONE

Phase one of development began in the third quarter of 1987. Its conclusion came in the second quarter of 1988 with the release of the Object-Oriented Programming, Simpak, and Graphpak components. Simpak has since been further decomposed into its natural subdivisions, Simpak and Statpak. The next sections describe these four components.

2.1. Object-Oriented Programming

The object-oriented programming (OOP) paradigm serves as the foundation for the subsequent components. Designed as an interface to Knowledge Craft's Carnegie Representation Language (CRL), OOP embodies the basic precepts upon which all object-oriented systems are based. Namely, all system components are represented as objects. The user defines the basic components with which they wish to work in terms of classes and subclasses of objects. Each subclass further refines or specializes its parent class as necessary to achieve the proper representation. The user creates instances, or specific occurrences, of their classes and subclasses to realize unique concepts or parts of a real-world system.

Objects communicate by sending and receiving messages. Messages allow objects to communicate without understanding each other's internal workings. They achieve this "black-box" functionality by referencing methods within the objects to which they are sent. It is these methods that manifest the behavior of their respective objects.

One important aspect of OOP is the concept of inheritance. Inheritance allows that, unless indicated by the user, subclasses inherit the inherent qualities of their parent classes. The user specializes or adds only those aspects of a subclass that differ from the parent class.

For example, a class of object defined as car might contain all of the basic characteristics of a generic automobile, such as a chassis, an engine, four wheels, and so on. Two possible subclasses of the car class might be domestic and foreign. These subclasses would for the most part inherit the traits of their common parent class, car. However, they would differ in certain key respects; for instance, the subclass domestic would represent speed in terms of miles per hour (MPH), while its foreign counterpart would represent the same quality with respect to kilometers per hour (KPH). Figure 2-1 presents the class/subclass hierarchy defined in this example.

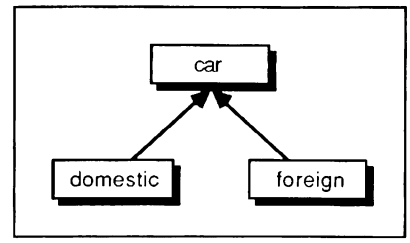

Figure 2-1: Car Hierarchy

When sent a message regarding its present speed, an instance of either subclass would initially access car's method for interpreting speed. However, each would then perform a specialized operation on the preliminary speed returned by car to obtain the value in the proper units. Once the proper methods had been defined within the two subclasses, the user could simply send a speed message to an instance of either subclass to obtain the correct interpretation of the value of speed, either MPH or KPH.

The use of an object-oriented paradigm offers the advantages of uniformity among sets of objects, abstraction with respect to objects' responses to the messages they receive, incremental definition in a class hierarchy, and explicit interfaces in the form of messages and the responses they elicit.

In addition to providing the user with all of the functionality they have come to expect from CRL (e.g., schemata and related functions), OOP also provides specialized functions, such as define-method, -> (send), and --> (send-super). Additionally, OOP provides many useful functions for the manipulation of methods (e.g., find-methods, get-messages, and describe-method).

Again, OOP serves as the premise upon which the ensuing tools are based. Defining the modeling and simulation tools with respect to this refined object-oriented paradigm provides the user with a consistent and powerful means of model development and simulation execution.

2.2. Simpak

Simpak allows the user to simulate a process via the construction and execution of a discrete event simulation. In discrete event simulation, system behavior is represented as a sequence of events; events are the actions that occur during simulation execution, such as loading a part onto a machine, tooling the part, and removing it from the machine.

The user schedules the events of the process being simulated onto the simulation event calendar in a distinct temporal order. The events on the calendar are then sequentially executed one at a time, again in temporal order, until the simulation terminates. The simulation is event-driven, meaning the simulation calendar only progresses with the occurrence of an event. Hence, the state of the model being simulated is stable between events.

To conduct a simulation, the user first creates an instance of the simulation object. This instance, referred to as the simulation executive, controls the simulation via management of the simulation calendar. Once the user has created the instance, they can then initialize it and create and schedule its events. Each event has an associated action, a time at which its action is to occur, a priority to be used in the event of conflicting event scheduling, and a status reflecting whether the event is active or has been deleted from the simulation calendar.

Simpak provides the user with a complete set of methods for manipulating a simulation and its events. The simulation object contains the instantiate and initialize methods necessary to prepare the simulation calendar, as well as methods to create, schedule, and reschedule events. The methods available for simulation calendar manipulation include methods for locating or deleting a specific event or any events that satisfy a predicate or contain a certain slot value.

Temporal progress is maintained during simulation by the simulation clock. The simulation clock progresses with the execution of events, maintaining the current simulation time in user-specified units. The simulation object includes methods that allow the user to execute a simulation in a variety of ways. The user can

- execute all of the events on the calendar without interruption,
- "step" through the simulation a specified number of events at a time,
- "tick" through the simulation a specified number of ticks of the simulation clock at a time, or
- run the simulation until a specified simulation time.

Finally, Simpak also includes instruments that enable the user to collect information from events executed during a simulation. Simpak offers two types of instruments, periodic and event. Periodic instruments perform their function at a specified interval of time, while event instruments monitor simulation activity associated with objects or messages. Coupled with their available methods, instruments allow the user to monitor and record a wide variety of data.

The next section, Statpak, includes information on generating random values to model non-linear system behavior. The section also includes a description of the data collection functionality available with the package.

2.3. Statpak

Statpak provides a number of tools for representing system behavior and collecting data during a simulation, and generating statistics as a result of the gathered data. Statpak's functionality comes in the form of random numbers, random variables, and data collectors. The use of these objects, especially as they relate to simulation, is described below.

Random numbers and random variables are used to represent the unpredictable behavior of a model's components during simulation. For instance, a sequence of random numbers might represent the number of parts a machine processes in a given period of time. Likewise, the value of a random variable might be used as the time at which an order should be queued.

Random numbers are drawn from random streams, which generate numbers between zero and one that do not follow a pattern. A random stream begins with a seed, which is a randomly chosen number within the acceptable range, and then continues to generate numbers within that range. Statpak provides eleven random streams to allow the user to maintain the integrity of their random numbers in simulations requiring multiple occurrences of such values.

To generate a random number, the user initializes one of the available random streams and sends it a generate message. Subsequent generate messages can be sent without re-initialization.

Random variables are taken from probability distribution functions. Probability distributions produce values designed to satisfy certain statistical parameters. Statpak provides eleven probability distributions, such as constant, uniform, and normal. Each produces numbers to satisfy unique

statistical requirements. For example, a uniform distribution produces numbers evenly distributed between its minimum and maximum values, while a normal distribution generates the majority of its numbers closer to its mean, with values further from the mean encountered in increasingly diminished proportions.

The user generates a random variable by instantiating a distribution and taking a sample from it. The user indicates the parameters within which the generated values are to fall at the time of a distribution's creation.

Data collectors are used to store or reduce the data gathered during a simulation. Statpak offers ten separate data collectors, each unique with respect to the combination of data it collects and the statistics it generates. For instance, one type of data collector is the 2-var-dc (two-variable-data-collector), which collects pairs of data and maps the approximate relationship between the values on a best-fitting line. The user can then access the slope and y-intercept of the line generated from the data, as well as the correlation, mean, maximum, minimum, standard deviation, and variance of the data.

To use a data collector, the user first creates and initializes an instance of one of the ten available classes. They then use the proper method to add data to the data collector, typically during simulation. The user can then generate statistics based on the collected data. Again, each of the ten data collector classes produces unique statistical data.

Graphpak can be used to display the collected data and generated statistics both during and after a simulation. Certain of its components can also be used to input data into a simulation. The next section describes Graphpak and how it relates to the simulation and statistical functionality already described.

2.4. Graphpak

Graphpak is an interactive graphics package that enables the user to create gauges and business charts to visually display data during or after a simulation run. Certain interactive gauges also allow the user to enter data during program execution. This section describes first the gauges and then the charts available with Graphpak.

Gauges. Gauges provide graphic input and output of numeric quantities during a simulation. Each basic gauge is composed of two primary components: a body and an indicator. The body of a gauge represents a fixed area containing graded tick marks and labels along the gauge's axis; the indicator, which moves along the axis of the gauge, represents the value being displayed.

Graphpak offers three types of basic gauges, two of which consist of a number of subclasses, as follows:

1. **Rectangular**:
    - line
    - bar
    - vertical
    - thermometer
    - vertical-thermometer
2. **Polar** (circular):
    - dial
    - meter
    - meter2
3. **LCD** (liquid crystal display)

A single basic gauge can display one or more values. A basic gauge with more than one value is called a multiple indicator gauge. There are two types of multiple indicator gauges: single value and multiple value. A single value multiple indicator gauge uses more than one indicator to display a single value. For example, a single bar-gauge might display a three-digit value, employing a separate indicator for the hundreds, tens, and ones of the value.

Conversely, a multiple value multiple indicator gauge displays more than one value, with a separate indicator for each value. For instance, a different bar-gauge might display two distinct numbers, using a separate indicator for each value.

Finally, multiple gauges can be grouped into a single macro gauge referred to as a composite gauge. As with multiple indicator basic gauges, composite gauges can be of the single value or multiple value type. A single value composite gauge displays the same value in more than one gauge. A good example of a single value composite gauge might be one that uses two basic gauges to display the

temperature in both degrees Fahrenheit and degrees Centigrade.

A multiple value composite gauge depicts more than one distinct value in different basic gauges. For example, a multiple value composite gauge attached to a simulation might use an LCD-gauge to display the current simulation time, a vertical-thermometer-gauge to indicate the number of orders completed, and a simple line-gauge to show the average cycle time of the completed orders. Figure 2-2 provides an example of this particular multiple value composite gauge.

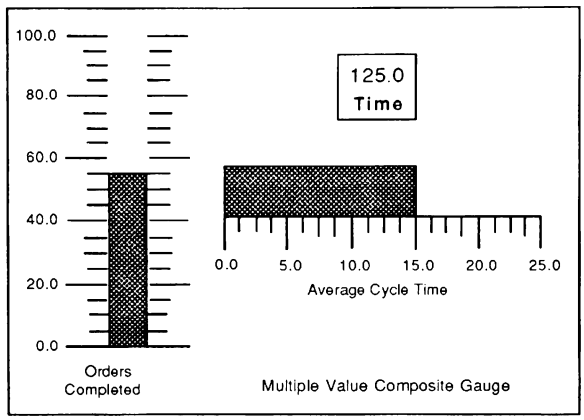

Figure 2-2: Multiple Value Composite Gauge

To create a gauge, the user first creates the canvas, window, and viewport in which the gauge is to reside. They then create an instance of the gauge, customizing its dimensions, labels, and other characteristics, and display it in the viewport. The user can then specify that the gauge be consistently updated to display the appropriate information, or use the gauge to provide input to the program to which it is attached.

Charts. Charts enable the user to display numeric values. Each chart is composed of two parts: a body and a curve. The body of a chart, like that of a gauge, represents the fixed portion of the device bearing its tick marks and labels. The curve portrays the data points associated with the value being displayed.

Graphpak provides three basic types of charts, the third of which can be represented in one of two ways, as listed below:

1. **Graphs**
2. **Pie-charts**
3. **Histograms**:
   - x-histograms
   - y-histograms

An example of a pie-chart depicting fictitious market statistics for workstations during the past year appears in Figure 2-3.

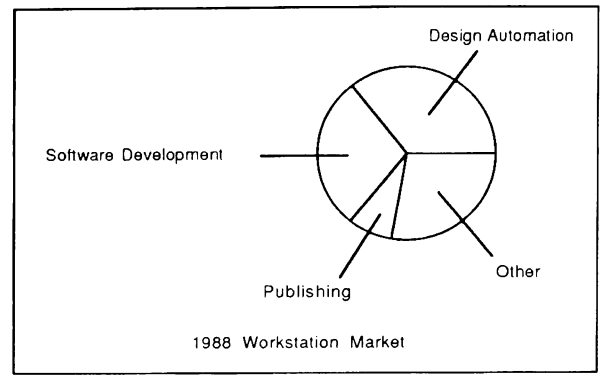

Figure 2-3: Pie-Chart of Workstation Statistics

The process used to create a chart is very similar to that used to create a gauge. The user begins by creating the canvas, window, and viewport. They then create an instance of the chart and display it in the proper viewport. The user is given a good deal of flexibility regarding the appearance of a chart. In most instances, the user can customize a chart's dimensions, its labels, the interpolation algorithm used to calculate its curve, and many other general characteristics.

2.5. Potential Applications

Simpak, Statpak, and Graphpak address the key requirements found in any solid simulation tool.

- *Planning.* The user can use Simpak to create and modify a simulation template, thus enabling them to easily test a number of different configurations. Using Statpak, they can collect data from and generate statistics about the various simulation runs. Graphpak then allows them to examine easily-understood visual displays of the statistical information they have obtained, graphically comparing the results of the simulations to make an informed, intelligent decision on their ultimate system configuration. This "what-if" functionality aids the user's ability to plan effectively.

- *Decision making.* As alluded to above, simulation allows the user to pinpoint the steps required to implement a procedure. Subsequently, necessary changes can be anticipated and verified before they are actually required. The user can foresee a change in demand, simulate one or more strategies to address the upcoming variation, and analyze the results of the simulation to effectively adjust the current procedures and schedules. Hence, the user can circumvent unnecessary disruption of their existing system, avoiding costly last-minute changes and saving valuable time and money.
- *Forecasting.* The available tools allow the user to test configuration hypotheses in a safe, risk-free environment. A change may prove to be beneficial, but there is also the possibility that it may be detrimental. The user can simulate the effect of varying certain conditions, analyze the results, and test their predictions without fear of eroded production.
- *Communication.* Visual information is a convincing medium. Graphpak allows the user to prepare visually effective statistics. These statistics can then be presented to management to illustrate the advantages and disadvantages of possible process alternatives.

The above four points, while individually impressive, collectively offer a valuable array of inter-related advantages. When used in conjunction with one another, Simpak, Statpak, and Graphpak can help the user realize four important simulation applications.

1. **Process planning**, which determines an effective sequence of operations for manufacturing a product.
2. **Facilities planning**, which uses throughput and cost constraints to assess the effectiveness of manufacturing facilities organization.
3. **Operations analysis**, which determines how effectively the current production process and facilities organization meet the end-product demand. Operations analysis addresses load balancing, lot sizing, and cost-analysis.
4. **Scheduling**, which generates a manufacturing sequence for the end-product based on the production process, facilities organization, operational decisions, and demand.

Applications constructed with Simpak, Statpak, and Graphpak can be used in a variety of domains, such as discrete machine job shops, flexible manufacturing, flow shop/transfer lines, and assembly shops. The following section describes an actual application that used these components to great advantage.

2.6. An Actual Application

Shortly after their release in 1987, Simpak, Statpak, and Graphpak were used to augment existing Knowledge Craft functionality in a knowledge-based simulation of a printed circuit board (PCB) manufacturing plant. The project was a collaboration of Italtel, a communications company located in Italy, and Carnegie (U.K.) Limited. The goal of the project was to simulate the production of PCBs for electronic switching equipment. The following two sections describe the system's initial requirements and the results of the deployed simulation system.

Requirements. Printed circuit boards begin in the Italtel plant as bare boards. Throughout a series of operations, hundreds of components chosen from a set of fifty thousand possible types are inserted in a board. Once assembled, PCBs endure a series of electrical tests and, if they pass, are sent to a storeroom to await shipping.

The plant at which this operation occurs consists of 85 distinct operations involving 120 employees. Eighty distinct boards are produced as the result of 165 possible plant routings. The operation produces approximately 2200 boards weekly. (The actual number and types of boards produced in a given week can vary considerably according to customer demand.) In addition to accurately modeling the above characteristics, the final simulation also had to handle dynamic part routing, worker scheduling, part aggregation and de-aggregation, machine downtime, engineering changes, and missing components.

The requirements of the simulation were typical of such projects. Italtel believed that a simulation system would assist production in the short term by enabling configuration experimentation without interfering with daily plant operation. Additionally, they believed that foreseeing possible bottlenecks would aid their operators in scheduling and dynamic planning. In the long term, management hoped

that the simulation would assist in the modification of plant set-up, as well as in the design of future plants.

Results. Based entirely on the Simpak, Statpak, and Graphpak technologies, the resulting system was a very encouraging preliminary tool for operations planning and scheduling, both static and dynamic. The final system effectively handled all of the initial constraints, proved to be easily understood and modified, and worked with better than anticipated speed. (The system was able to efficiently simulate two months of plant activity in less than an hour, better than twice the speed originally hoped for.)

The final system included an impressive interface that guided the user through each step of the simulation process. This interface included numerous gauges that allowed the user to conveniently alter predefined parameters for experimentation purposes. Each of the operation's modifiable parameters was represented by a separate, modifiable LCD-gauge. in addition, the system's many distinct parts were displayed in a vast array of x-histograms that enabled the user to alter a simulation during run time and immediately view the results of their changes. A single simulation could be executed with different boards, various lot sizes, the addition, deletion, or reassignment of workers, machines, and shifts, and routing and capacity alterations.

Furthermore, the system enabled the user to interrupt a simulation at any time to display graphs, histograms, and pie-charts reflecting its current state. The user could then continue the simulation from the point at which it was stopped or invoke a new simulation run.

More impressive possibly than the results themselves is the fact that this entire system was implemented by five full-time and two part-time individuals in a period of just fourteen months. Obviously, without the benefit of flexible underlying simulation and graphics packages, the span of the project would have been much greater.

3. PHASE TWO

Phase two of development began with the third quarter of 1987 and continued until the end of the second quarter of 1988, culminating in the completion of Timepak and Modelpak. These components are described in the following two sections.

3.1. Timepak

Timepak provides a foundation upon which users can represent and reason about temporal occurrences. Timepak consists of two primary subsystems: absolute time and relative time. The following sections describe these systems.

Absolute Time. Absolute time allows the user to create a temporal representation associated with fixed or definite occurrences. The system consists of four basic subcomponents:

1. **time units**, which are the terms by which all other absolute time objects are defined;
2. **time points**, which represent specific moments in time;
3. **fixed intervals**, which are intervals of time defined by definite starting and ending points;
4. **durations**, which are spans of time not associated with any particular points in time.

The development of an absolute time representation is dictated by two basic facts: (1) each time point and duration must be defined in terms of some time unit, and (2) each fixed interval must have both a start point and an end point to define its limits. Consequently the user typically performs a consistent sequence of steps when initially using absolute time to prepare a temporal representation.

1. The user begins by defining any time units they wish to use. The user can build hierarchies of time units by relating one unit to another and then defining methods for translating between the units. All time units must be based in some way on Timepak's special Universal Time Representation (*UTR*) variable, a time-line whose increments and limits are undefined until associated with a user-defined time unit.
2. The user then defines time points to be used in their representation. Time points denote precise, indivisible instances in time. Each point is defined in terms of a value/unit pair indicating the instance at which it occurs. Although they can stand alone, the primary use of time points is to delineate fixed intervals.
3. The user continues by creating fixed-intervals, which are references to specific intervals in time. The user creates a fixed interval by

defining its start and end points in terms of existing time points. Once defined, a fixed interval can be thought of as a bounded sequence of time points in which each successive point occurs immediately after the preceding point.

4. Finally, the user can define durations, which are fixed quantities of time points not delimited by specific points. Unlike fixed intervals, durations have no end points; they are instead defined similarly to time points, with a specific value/unit pair. However, where the value associated with a time point refers to a specific point in time, the value associated with a duration actually indicates a successive series of time points referred to as a span.

   The creation of durations after time points and fixed intervals is completely arbitrary; durations can actually be created any time after the creation of time units.

Timepak includes myriad methods for comparing and performing mathematical operations on time points, fixed intervals, and durations. Hence, once the user has defined a thorough representation, they can easily compare and manipulate the various objects in their temporal model, generating reports that can be used to evaluate the effectiveness of the current representation.

Relative Time. Relative time is a time reference that relates events, or occurrences, without necessarily associating them to any specific period in time. Events in relative time are related to each other, rather than to time units or points; they need neither occupy any fixed intervals nor span any set durations. This freedom from the constraints of absolute time allows the user to define a network of events solely with respect to their temporal relations to each other.

The process of creating a relative time network involves four steps.

1. The user first creates the events being used in their representation.
2. The user then gathers their events into groups, or clusters, of events by asserting reference events. Reference events serve as parent events to any number of children. Additionally, each reference event can have a reference event of its own. Thus, the user can create an intricate hierarchy of events organized by cluster.
3. The user continues the process by relating each event to first its reference event and then the other events in its cluster. In Timepak, relations consist of statements such as "event one occurs *before* event two." A full range of such relations is available to the user.
4. The user completes the representation by relating the various reference events to each other. The user can also assert relations between events from different clusters, providing such assertions do not violate any existing relations.

When the network has been completely defined, grouped, and related, Timepak provides the user with methods allowing them to determine the relationships that exist between any of the events in the network. This information can be used to draw conclusions and to reason about the process represented by the network.

3.2. Modelpak

Modelpak provides the user with the tools to model a real-world process. Although to date it has primarily been used to represent systems in the manufacturing domain, the tool can be used to model processes from any number of applications.

Modelpak allows the user to define a process as a series of discrete events referred to as activities. Activities correspond to the actual actions represented by the system, such as loading an object onto a tool or operating a piece of machinery.

Additionally, Modelpak supports activity aggregation in the form of complex activity groups. A complex activity is an activity that consists of a number of other events. The user can compose a number of activities in a single complex activity hierarchy, or they can decompose a single activity into the activities of which it is composed. Either way, the resulting activity structure allows the user to model a process at the desired depth. For example, starting a car could be modeled with a single activity, or it could easily be represented by a group of more precise events (e.g., opening the car door, placing the key in the ignition, turning the key, and so on).

In Modelpak, a physical substance required to perform an activity (such as the fuel required to start the car in the example above) is represented as a resource. Activities can be constrained by the lack of required resources. Like the physical quantities they symbolize, resources have capacities that diminish with use. Some are consumable, meaning they do not replenish after use, while others resume their original capacity once the user is finished with them. Furthermore, like activities, resources can be aggregated into complex resource groups (e.g., five workers can constitute a team).

The user creates a Modelpak representation by first defining the activities and resources to be used in their model, and then creating any complex activity or resource groups. Next, they link the activities to each other in the sequence in which they occur, as well as to the resources that they require. The result is a hierarchical model of activities and resources representing a discrete process.

The user can access the resulting model to perform static evaluation of the system being represented. They can also employ a simulation package such as Simpak to execute the model's components.

3.3. Potential Applications

The potential applications for Timepak and Modelpak are essentially very similar to those previously listed for the phase one components.

- *Planning.* The user can utilize either Timepak or Modelpak for planning. Both components allow the user to model a system, either temporally or physically, and draw conclusions about the representation.
- *Decision making.* Again, both packages allow the user to define the steps involved in a process and then modify their definition as mandated by real-world demands and constraints. As with the phase one components, Timepak and Modelpak both provide the user with the opportunity to effectively evaluate and adjust their current process without disrupting the actual system.
- *Forecasting.* Theories and hypotheses can be tested and verified in a safe environment. The benefits and drawbacks of any idea can be determined without the risk of harm from unforeseen side effects.

Process planning, facilities planning, operations analysis, and scheduling are all addressed. Timepak is especially useful alone or with Modelpak for scheduling applications. Additionally, when used together, relative and absolute time allow the user to first develop the relationships between events and then constrain the relationships by assigning them specific temporal values.

4. CONCLUSION

The components described in this paper address a wide array of applications. The primary advantage of this diverse collection of tools is the modular approach they bring to the arenas of planning and scheduling. Developed as separate units, they can be used that way or combined to great advantage.

- *For small problems*, the user need only purchase the component they need. There is no cause for them to purchase an immense system consisting of tools they may never use. This reduces their expenditure of both money to acquire the package and time to learn how to use it.
- *For large tasks*, Simpak, Statpak, Timepak, and Modelpak can be used in any number of combinations or collectively to build domain specific shells. Shells can be used to make the burden of solving large, intimidating problems that much lighter. And, once an effective shell has been developed for one application, it can easily be applied to other problems in that same domain. Obviously, the primary shell opportunities lie in the areas of scheduling, process planning, simulation, and design.

ACKNOWLEDGMENTS

The authors and Carnegie Group, Inc. would like to extend their appreciation for the support and input received from Digital Equipment Corporation throughout the development phases described in this paper. They would especially like to thank Frank Lynch and Chuck Marshall of Digital Equipment Corporation for their continuing contributions. Finally, they would like to express their appreciation to the various other employees of Carnegie Group, Inc. who contributed to the development of the phase one components.

AUTHORS' BIOGRAPHIES

JIM BYCURA is an Engineer in Modeling and Simulation Systems at Carnegie Group, Inc. He has worked at Carnegie Group since May, 1988.

CINDY COSIC is a Senior Engineer in Modeling and Simulation Systems at Carnegie Group, Inc. She has worked at Carnegie Group since July, 1988.

JEFF KAMINSKI is a Technical Writer in Modeling and Simulation Systems at Carnegie Group, Inc. He has been with the company full-time since January, 1988.

JOYCE KEPNER is a Quality Assurance Engineer in Modeling and Simulation Systems at Carnegie Group, Inc. She has worked at Carnegie Group full-time since February, 1988.

GARY STROHM is Manager of Modeling and Simulation Systems at Carnegie Group, Inc., where he has been employed since September, 1985.

All five authors can be reached at the same address and phone number.

Carnegie Group, Inc.
Five PPG Place
Pittsburgh, PA 15222
(412) 642-6900