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

IMPLEMENTATION OF RULE-BASED TECHNOLOGY IN A SHOP SCHEDULING SYSTEM

David P. Yancey, Ph.D.
Scott Peterson
Pritsker Corporation
1305 Cumberland Avenue
West Lafayette, IN 47906, U.S.A

ABSTRACT

FACTOR is a product which schedules and sequences activities based on discrete event simulation; recently two new software modules have been added. The Output Analysis System (OAS) module analyzes simulation output, determines performance problems, suggests solutions, and implements the solutions by changing the model. The Site Specific Tailoring (SST) module supports an in-model rulebase which can implement control logic.

This paper provides an overview of the rule-based approach which was used to implement these modules and a review of their operation. The focus of this paper is to explore what expert system technology was utilized and how it was implemented for scheduling. Development issues and application examples are discussed.

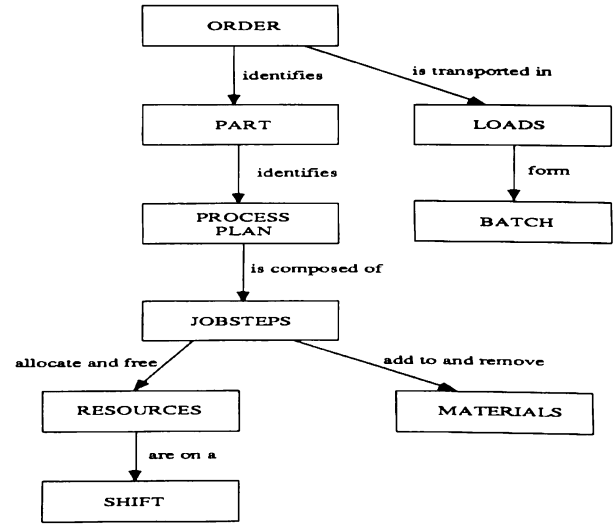
1. FACTOR OVERVIEW

Effective production scheduling has been an important area of concern for many years in manufacturing facilities. Products like MRP II have provided a measure of success in this area for high level planning and reporting. What has been missing, however, is effective finite capacity scheduling, taking into account the current status on the shop floor. FACTOR is a decision support software package which was developed to address the needs of the shop floor scheduling supervisor and provide effective predictions of production performance. FACTOR provides a tool for evaluating alternative responses to changes in production plans such as machine failure, expedited orders, or late material delivery.

FACTOR is designed to be integrated with other production management software. Modules interface to MRP and shop floor data collection to extract the information required to schedule production. FACTOR loads data into its model database either through standard or customized import programs. Information from MRP includes a list of orders and their due dates. Shop floor data collection provides the status of in-process orders released to the factory floor.

There are two users of FACTOR. The scheduler is the "end user" who applies the factory model to create schedules and solve scheduling problems. The details of the simulation model are typically transparent to him. The modeler sets up and maintains the factory model and the external data files used by FACTOR.

FACTOR provides a large variety of modeling capabilities with which the model builder can represent the physical production system and its associated operational procedures. A FACTOR model is a representation of these physical and operational aspects of the manufacturing system: machines, material handlers, personnel, process plans, orders, and so on. The figure below illustrates a few of the fundamental FACTOR model components and their interrelation. More detailed com-



ponents are used to represent resource groups, AGV systems, fixtures, tools, conveyor systems, SRS systems, and others which are not represented here.

FACTOR performs a discrete event simulation of the manufacturing process to generate a detailed schedule. The process begins when an order is released to the system. An order is a production request for a quantity of a particular part type. The quantity and the number of parts per load determine the number of load transactions introduced into the simulator. The load transactions are routed through the production system based on a process plan and the jobsteps which comprise that process plan. A jobstep is an action taken on (or because of) that load or batch of parts. An action may be move, process, palletize, setup, draw from inventory, etc. Time delays and resource requirements may be associated with a jobstep. A load transaction may also require that material be drawn from a material inventory area before it begins the jobstep. Material is a special type of consumable resource which represents raw material or completed subassemblies. Materials may supply assembly jobsteps and link the process plans of several subassemblies together.

In addition to the process plan, there are other aspects of a system which cause the status to change. These aspects include the production calendar, maintenance schedules, shift schedules, order arrivals, and sequencing logic rules. The production calendar, maintenance schedules, and the shift schedules determine at what times particular resources are operational. Order arrival initiates the production cycle of a part. Sequencing logic rules determine which load a resource will service when more than one load waits for that service.

The end user supplies alternative management data to control the production run of the model. The information specified controls the start and end times of the scheduling window, the sequencing and scheduling rules to use, and the output reports to generate.

FACTOR is typically used for detailed short interval scheduling with a horizon from 24 to 168 hours, although FACTOR has been used for medium to long term capacity planning. Pritsker Application Engineers report that typical FACTOR models contain roughly 500 orders, almost as many process plans, and 30 resources. However, it's not unusual to find models with 10,000 orders or 80 resources. Consequently, simulation times can vary greatly with the scope of the model. Application Engineers typically try to scope the model such that the simulation can be performed within 10 minutes on a minicomputer.

2. ROLES FOR EXPERT SYSTEM TECHNOLOGY IN FACTOR

In Version 4.0 of FACTOR, two new software modules were added which provide expert system capabilities: the Output Analysis System (OAS) module and the Site Specific Tailoring (SST) module.

The FACTOR base system provides detailed schedule generation capabilities. The amount of data generated, however, can easily overwhelm the end user who is searching for trouble spots in the schedule produced by FACTOR. The OAS module allows the modeler to construct an expert system module which detects problems and suggests solutions to improve the schedule. Solutions can be incorporated in a new alternative which is created automatically by an alternative generator. This analysis process can be iterated manually or automatically until user-specified tolerances are met, or the problems are alleviated.

The SST module also supports the construction of expert systems. The FACTOR base system is an open ended product which supports user written extensions in a variety of ways throughout the system. Each scheduling application tends to be unique in its requirements for implementing specific decision logic or other special processing within the model (e.g. custom job sequencing). FACTOR provides a number of entry points where the modeler can install his own C code. The SST uses the expert system approach to offer an alternative for the implementation of custom decision logic at key points in the simulation model.

3. CONSTRUCTING EXPERT SYSTEMS WITH FACTOR

An expert system is a special-purpose computer program which utilizes expert knowledge to analyze a narrow problem area. It is developed through the use of an expert system building tool. This tool provides a high level language for describing the expert knowledge and facilities to help the user interact with the expert system once it has been built. Expert knowledge is present in the form of facts and rules for decision making. FACTOR employs these elements of expert systems to incorporate decision logic to support the scheduling process.

In FACTOR, a rulebase is composed of a set of related rules which represent the expert decision making logic necessary to solve a particular type of problem. Several different types of rulebases are supported for the

various types of decisions that can be made; a rulebase is defined for each decision point to be programmed. The function of a rule is to infer new knowledge about the problem. New knowledge is inferred by the action part of the rule (THEN clause), while the requirements to be met are specified in the situation (IF clause). Thus, when a rule is tested, the knowledge base must be searched, and if the tests succeed, additional knowledge is inferred.

The knowledge base refers to the set of facts used and generated by the associated rulebase. In FACTOR, an **object** refers to the internal representation of a single fact. An **object class** is a way to specify a collection of objects (facts) of a certain type. For each type of rulebase supported, there is a set of object classes available for use. Each rulebase type and its set of associated object classes is called a **rulebase context**. Only those FACTOR object classes which are relevant with each context is known.

Object classes which are known by the expert system without having to be defined by the user are termed **predefined object classes**. In general, there is a set of predefined object classes for each rulebase type. These are used to represent simulation knowledge available in that particular context. Each object class has an associated set of attributes. For example, attributes of the object CURLOAD (current load) include load size, priority, remaining processing time, etc.

It is sometimes necessary to derive some additional information about objects. Additional attributes which may be determined for the purpose of effecting computations are referred to as **computed attributes**.

Additional objects (intermediate or inferred knowledge) may be added to the knowledge base. User-defined object classes are termed **temporary object classes** since their definition is known only within the rulebase in which they are defined. These will define the structure of the new knowledge added to the system.

To provide programmers complete control over their applications, the C programming language may be incorporated in the rulebase. Computed attributes can be based on function calls involving other attributes on an object or the simulation variables known within the current rulebase context. C code can also occur in the action part of a rule. C code may include calls to user-written functions, calls to predefined utility functions, or a code block to execute upon firing a rule.

A template for rulebases is illustrated below. The rulebase contains all the rules and special definitions appropriate for its context. User-written C code may also accompany the rulebase.

```
RULEBASE context
    TEMPORARY CLASSES
        object definition statements
    END

    COMPUTED ATTRIBUTES
        attribute definition statements
    END

    rule_name:
        IF EXITS situation THEN action;

    Additional rules;

    END
```

Rules are the key element for describing the decision making logic. The IF clause expresses a requirement to find an object (fact) in the knowledge base which meets certain criteria. Its form is illustrated by the following example.

```
rule_excess:
  if exists
  a resheld res_name where (res_name.capacity > 10)
  then ...
```

The key phrase **if exists** signals the beginning of the situation part of the rule. The construction "a resheld res_name where (...)" is the condition requirement which must exist. There can be as many match requirements (AND conditions) as needed to describe the situation. The keyword **resheld** refers to a predefined object class for this context. In this example, objects that are members of the resheld class refer to a resource held by the current load. The resheld object has many attributes, among them is **capacity**. If there were resources held by the current load, the variable **res_name** would be bound to the first resource in the list with a capacity greater than 10.

The keyword THEN signals the action part of the rule. The action can perform operations on an object found in the situation part, do something specific like display a message, or both. General actions which may be performed include the following:

Add - Add an object to a temporary class to represent intermediate or inferred knowledge.
Remove - Remove an object from a temporary class.
Goto - Try a specific rule next.
Message - Display a message.
Trace - Conditionally display a message if the trace option is on.
{C code} - Embedded C code actions, most commonly used to call a custom C code function.

Other special context specific actions are described in the OAS and SST discussions.

The basic strategy incorporated in processing the rulebase is to test each rule in the order specified. When a rule situation is found to be true, its action part is executed and testing of rules begins again with the first. Termination occurs either when none of the rules fire or when a special action terminates the inferencing process.

The construction of expert systems is illustrated for both SST and OAS applications in the sections that follow.

4. SST MODULE OVERVIEW

There are many points within the FACTOR simulation of a part's process plan where decisions need to be made. In many cases, the modeler is provided a number of options for decision logic. For example, when a load arrives at a jobstep, use option 17 for selecting between this or an alternate jobstep. FACTOR provides a number of standard options for these decision or processing points. The SST module provides the user the capability to install custom options via user written code. The rulebase components of FACTOR provide an alternative method, but are invoked and return information in a similar manner.

The rulebase contexts supported by the SST module are summarized in the following list.

load_rank Sequencing decisions for order releases, resource requests, material requests, AGV requests, and others are based on the ranking of the candidate loads. This rulebase returns a ranking value for the current load.

req_select When one or more units of a resource becomes free the resource attempts to satisfy the requests that have been placed by loads. This rulebase decides which request is selected next.

rgm_select Select a resource for allocation from a group.

res_alloc Determine the manner and order for allocating resources for a jobstep. Return true or false to indicate resource allocation success.

batch_sep At the batching jobsteps, arriving loads may be placed into one of several batch loads. This rulebase selects an appropriate forming batch.

batch_quan Determine how much a load should increase the quantity of a batch that it is accumulated in.

batch_ov Determine if forming batches should be released even though they have not yet reached minimum release quantity.

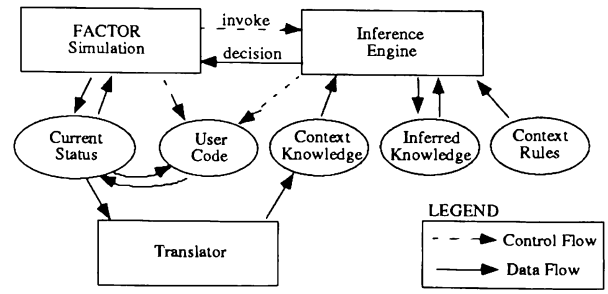
agv_select Determine which vehicle should be selected from a list of idle vehicles in an AGV fleet.

sjs_select Select a jobstep to process next from a given list of jobsteps.

js_select Decide whether to execute the current jobstep or an alternate.

step_time Determine the time duration for the jobstep by this load.

Whenever an inference engine is invoked, current simulation status information which makes sense for the decision context is mapped into facts for the knowledge base. Intermediate or inferred knowledge is maintained in temporary object classes. The final result of the rules processing is returned to the simulator in the form of a decision. The following figure illustrates this strategy.



Rather than providing a general purpose inference engine which can be used for any set of rules, FACTOR generates a custom inference engine and related utility routines which are optimized for each rulebase. The inference engine is linked with the simulation executable.

In order to support SST interaction with the FACTOR simulation the following additional actions are permitted in the THEN clause of rules.

Request - Register a request for a resource.
Allocate - Allocate an available resource.
Return - Halt inference engine processing and return a decision made by the rulebase. This action is required in every SST rulebase. The return value expected depends on the rulebase type (context).
Error - Declare a fatal error condition.

To illustrate these constructs, an example SST expert system is constructed in the section below.

5. AN SST SCENARIO

A manufacturing system manager has a sequencing philosophy which ranks all jobs according to priority. When a tie occurs, the jobs with a lower remaining processing time are given preference. Under this philosophy, however, it is possible for an order with relatively low priority at the beginning of the scheduling window to become critically late during the manufacturing process. The manager wants to avoid situations where an order with low priority and a long processing time sits on the floor and becomes critical. An order becomes critical if the expected completion date is greater than the due date.

This policy can be incorporated when units of a resource become available. If there are several loads wanting the same resource and there are not enough units of a resource to service all of the requests, the resource should be allocated to the loads that are critical. If no loads are critical then resources should be allocated to loads by the stated philosophy according to job priority and remaining processing time.

A resource request selection rulebase (req_select) decides between loads which have placed requests for a resource. This rulebase may be invoked when one or more units of a resource becomes available. The rulebase decides in what order the requests are examined, and in some cases, how many of the requests are to be processed.

When a resource becomes free, it will notify the loads which have placed requests for the resource. The requests are contained in a list owned by the resource, and are initially ordered by the appropriate load ranking rule. The resource notifies these loads in the order of the list until there are no more units of the resource available or all the requests have been satisfied. Loads which attempt allocation may or may not successfully allocate the resource, since complex multiple resource decisions may be involved.

Resource request selection rules differ from load ranking rules in that they may modify a load's position in the request list each time the resource becomes available. This allows the loads in the list to "age". This dynamic ranking of the list provides a means to incorporate considerations regarding changing system status and look ahead.

There are five knowledge classes which represent predefined facts available within the resource request selection (req_select) rulebase. One that is used in this example is **req**, a class containing one object (fact) for each request for the resource by a load. Other classes in this context contain facts about the resource or the requesting loads. Among the attributes provided for the REQ object class are:

priority	Ranking priority of request
lremproc	Remaining processing time
oduedate	Due date
remtim	Estimated remaining processing time
stslack	Static slack (oduedate-DATENOW)
dyslack	Dynamic slack (stslack-remtim)

A total of 57 attributes are available for the **req** object class. A similar, though less extensive, set of attributes is available for each object class in the context.

The rulebase for the selection policy was constructed as follows.

```
RULEBASE REQ_SELECT 14

  COMPUTED ATTRIBUTES
     ATTACH TO REQ
        (real rank priority*10000 + lremproc)
  END

Rule1:
! If expected time remaining is greater than
! time to due date, then select this request.
     if exists
         a req R where (R.remtim > R.stslack)
     then
         return (R);
```

```
Rule2:
! If no request from "critical" loads were found
! then implement the load ranking logic.
    if exists
        a req REQUEST_A
        no req REQUEST_B
          where (REQUEST_B.rank < REQUEST_A.rank)
    then
        return (REQUEST_A);
END
```

The computed attribute added to the **req** object class computes the rank value based on priority and remaining processing time. Low values for **rank** are "hot" orders. Rule **rule1** searches through the **req** object class looking for any loads that are critical. If an object is found which meets the condition, **rule1** returns this object and the rulebase stops executing. Rule **rule2** searches the request queue for the request **request_a** such no request **request_b** has a **rank** attribute which is lower. In effect, **rule2** returns the request with the lowest ranking value.

For purposes of illustration, the above rulebase was kept fairly simple. Although no intermediate facts were inferred, they may be developed for more complex scenarios. As a matter of practice, however, SST rulebases tend to be small (fewer than ten rules).

6. OAS MODULE OVERVIEW

This section describes the OAS module which is designed to aid the user in the analysis of output produced by a FACTOR simulation run. The output analysis process is broken into three phases. Phase I is the process of analyzing the FACTOR output database (and possibly the model in the input database as well) to detect problems in the generated schedule. As part of this phase, all FACTOR input and output database records are converted into knowledge objects and are stored as predefined facts in the expert system knowledge base. The first phase expert system is then invoked to derive a set of problem objects (facts) and store them in the knowledge base.

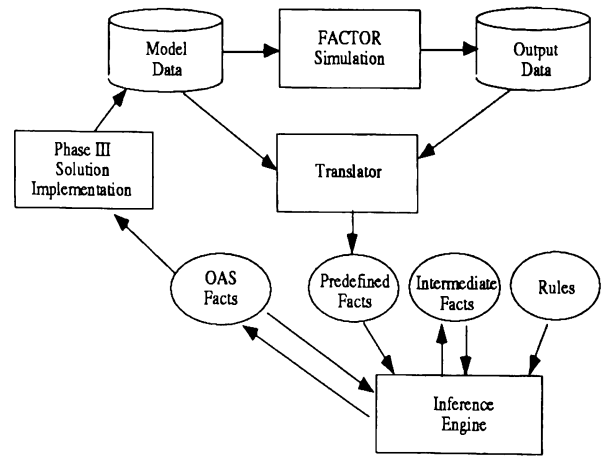
Phase II is a similar process which further analyzes the facts that represent FACTOR output and input along with the problem facts derived during Phase I. The end result of the Phase II expert system analysis is a set of solution facts which address the schedule problems.

Phase III is the process of implementing the set of solutions resulting from Phase II. The end result of Phase III is a new alternative generated from the current model. The OAS module supplies a standard set of solution implementation options which can alter the FACTOR model in a variety of ways. This standard set includes the following.

DUE-DATE	alter the due date of an order
LOAD-SIZE	alter the load size of an order
PRIORITY	alter the priority of an order
RELEASE-DATE	alter the release date of an order
PROCESS-PLAN	alter the process plan for an order
CAPACITY	alter the capacity of a resource
MAX-OT	alter the maximum overtime allowed for a resource
RANKING-R	alter the ranking rule used by a resource to satisfy outstanding requests
SELECTION-R	alter the selection rule to use for a resource by jobstep
SHIFT-SCHEDULE	move a resource to a new shift
ON-HAND-A	alter the on-hand amount of a material
RANKING-M	alter the ranking rule used by a material to satisfy outstanding requests
RANKING-A	alter the ranking rule for default order sequencing
RELEASE	set the order release rule for sequencing orders to be released at the same time.

The above solutions were determined to provide good coverage of techniques used by planners out in the field. However, an open ended interface is provided to support the programming of other solution implementation options which alter other elements of the FACTOR model.

A conceptual view of the OAS integration is illustrated in the figure below. Model and simulation output information is provided as predefined facts. Intermediate or inferred knowledge is maintained in temporary object classes.



The three-phased approach described above has two very important aspects. First, it breaks a complex problem into three manageable components. Second, since the interfaces between Phases I & II and between Phases II & III are accessible to the user (through standard editors), the user has the ability to interact with the expert system rather than just blindly following its advice.

The user is provided two modes of operation, manual and automated. In either case, an expert system is developed for each of the first two phases. In the manual mode, the user controls each phase of the analysis process. In the automated processing mode, an optional Iteration Control rulebase is constructed to describe stopping conditions for iterative analysis involving repeated applications of Phases I, II, and III.

A PARAMETERS section may be added to the rulebase structure for OAS expert systems. A parameter is a constant defined in the rulebase file and set to some initial value. In the rules, it can be referred to just as though it were a literal number, character or string. The advantage is that later, when that constant needs to be changed, the user may invoke the parameter editor and the value changed without recompiling the whole rulebase. Parameters can be used to set limits on late orders (two hours late for one analysis run, five hours late for the next), declare maximum utilization of bottleneck resources, and so on. The use of parameters is illustrated in the subsequent example.

The OAS also features a context specific action in the THEN clause of rules. The **stop** action may be incorporated in an Iteration Control rulebase to terminate simulation cycling.

The predefined facts available to the knowledge base include information which describe the FACTOR model and simulation results (whereas SST facts reference current status information during the course of the simulation). FACTOR model object classes include:

jobstep - a description of a jobstep as part of a process plan
material - a description of a consumable material
order - a description of an order
part - a description of parts
resource - a description of a resource

and many more. FACTOR simulation output object classes include:

loadsum - load summary data
ordsum - order summary data
resplot - resource plot data
resstat - resource statistics
resschd - resource schedule data

and a number of others. The OAS specific object classes include the following.

iteration - an object identifying the current iteration count
phase - an object identifying the current phase
problem - identified problems from Phase I
solution - identified solutions from Phase II

7. AN OAS SCENARIO

A manufacturing center uses the FACTOR system to schedule orders for its facility. It is now desired to alter the schedule by changing load sizing as necessary to minimize late orders. This is a schedule adjustment technique that up to now has been applied manually and has been reasonably effective.

The first step is to identify late orders from order summary information. Late orders are those orders that complete after their due date. The **ordsum** object has attributes which describe order due date, completion date, lateness in hours, and other order summary information. A test on the **lateness** attribute of **ordsum** will identify late orders. An additional test is added to determine that no problem has already been registered for this order. This test is added to prevent repeated firing of this rule.

We realize in formulating this rule that the threshold for declaring order lateness to be a problem may be somewhat subjective. Consequently, the LIMIT parameter is introduced which allows the scheduler to establish the point at which problems are identified and addressed.

The following illustrates this Phase I rulebase. Three attributes are used for the **problem** object class; **descr**, **part_name**, and **ordr_name**. (The FACTOR UIT module was used here to define mnemonic attribute names.)

```
RULEBASE PHASE_1

PARAMETERS
  LIMIT (param_label "Max Hours Late Allowed"
         data_type           real
         initial_value       0.0
         checking_type       NO_CHECK )
Rule1:
  if exists
     an ordsum OS where (OS.lateness > LIMIT)
     no problem PROB
        where (PROB.ordr_name = OS.order)
  then
     add problem
          where (descr = "LATE ORDER"
                 ordr_name = OS.order
                 part_name = OS.part)
     message ("Late order: %s\n", OS.order);
END
```

The next step is to outline a strategy for the expert system to use in determining solutions. For the sake of this example, we select a strategy in which a late order for a certain part will cause all orders for that part to have their load size halved. However, if more than one order for a given part is late, the load sizes will still only be halved once for the new schedule alternative simulation. This solution strategy is the reason behind saving the part number from the order object in Phase I analysis. This part number is needed to find all orders that are to have their load size changed.

The rulebase implementation of this logic is illustrated below. The part number is declared in the **part_name** attribute of the problem object class. From this, a solution object is stored for each of the orders (from the model database) with a matching part number that has not already been declared as a solution object. Four attributes are defined for the **solution** object class: **name**, **prob**, **order**, and **factor**.

```
RULEBASE PHASE_2

Rule1:
    if exists
        a problem P
            where (P.descr = "LATE ORDER")
        an order O where (O.part = P.part_name)
        no solution S where (S.order = o.id)
    then
        add solution
            where (name = "CHANGE-LOAD-SIZE-R"
                   prob = p.descr
                   order = o.id
                   factor = 0.5);
END
```

The solution "CHANGE-LOAD-SIZE-R" is one of the standard implementations provided by the output analysis system. It needs to know the name of the order to change and the factor by which the load size should be multiplied. The solution objects will drive the alternative generator of Phase III to make referenced changes in the model database.

After we have compiled the rulebases, and run the simulation, we can perform Phase I and II analysis. The new alternative can then be simulated and the analysis performed again. For those occasions where it is desired to automatically control the simulation iteration process, an iteration control rulebase is developed. For example, suppose that we want to iterate the output analysis process until either four iterations have completed, or there are no problems in the knowledge base following Phase I. The rulebase would appear as follows.

```
RULEBASE ITER_CONTROL

Rule1:
    if exists an iteration I where (I.count = 4)
    then stop;

Rule2:
    if exists
         a phase P where (P.number = 1)
        no problems
    then stop;
END
```

8. INDUSTRIAL APPLICATIONS

The FACTOR OAS and SST modules were first released by Pritsker Corporation in January 1989. As of this writing, it is too early in the process to report on applications of these modules in production. However, as part of the software development process, several corporations participated in a beta test program which sought to evaluate the product and make recommendations for improvement prior to commercial release. The discussion below reviews a few of the applications and customer observations.

A large aerospace corporation developed two OAS applications with a FACTOR system then being installed. The FACTOR system will be used to schedule the production of composite material parts for one production cell over a 24 hour horizon. This first OAS application was designed to identify resource bottlenecks and try to balance the load by increasing some resources while decreasing others. This reflected the condition where some men could perform more than one operation, and could be reassigned to other work stations depending on where they were most needed. For this application, 12 rules were used in the problem rulebase and 3 for the solution rulebase. The iteration rulebase stopped simulation cycling after a fixed number of iterations or when no problems were left.

The second OAS application was used to reduce the jobstep times of a few key jobsteps when certain production goals were not met. The problem rulebase contained 7 rules for identifying production goals that were not met. Seven rules were also used in Phase II. To implement Phase III, custom solutions were developed to change jobstep times.

On the basis of these applications, several observations were made regarding the OAS module and its approach. This customer's previous method of analyzing schedules required pouring over the standard reports,

manually reviewing statistics, and looking for various problem patterns in the data. The rule-based approach expressed the patterns directly. The biggest advantage cited was the ability to accelerate the cycle: identify problems, apply solutions, and re-simulate the model.

Similar beta tests were conducted for the SST rule-based module. Test applications were developed to support the manufacture of off-highway truck components and the implantation area of an electronic chip fabrication process. Several rulebases were developed to capture decision logic to be integrated with the simulation model. Rulebases tested resource allocation strategies based on jobstep setup conditions, alternate jobstep selection based on setup conditions, and special batch release decisions.

On the basis of these applications, a few observations were made. First, a programming knowledge was deemed necessary for developing SST rulebases, although reduced programming skills were required. It was also observed that the rules are expected to provide improved maintainability over standard C code.

9. CONCLUSION

The above discussion has shown how the expert system tools provided by the Site Specific Tailoring module and the Output Analysis System can implement a wide variety of decision logic within the FACTOR simulation framework. It is extremely important for a tool as powerful as FACTOR to be easy to use, yet it must be able to address the wide variety of needs and expertise in different manufacturing situations. The rulebase language provides a way to apply expert system techniques in an intuitive, English-like environment, and the support functions allow smooth integration of this environment into FACTOR. An important aspect of the SST module is efficiency. In a production scheduling environment, logic decisions may be required by the simulation thousands of times. By code generating an embedded inference engine, the use of rule-based techniques becomes a practical problem solving technique. In addition, the OAS support utilities which include a parameter editor, knowledge editor, and new alternative generator provide the scheduler with an environment tuned to solving everyday scheduling problems.

The OAS and SST modules demonstrate practical applications of expert system technology to the simulation process. With their application, we expect to realize greater efficiency and effectiveness in the process of scheduling with simulation.

REFERENCES

Grant, F. H., "Simulation and Factory Control - An Overview", 9th International Conference on Production Research, Cincinnati, OH, August, 1987.

MacFarland, Douglas and F. H. Grant, "Shop Floor Scheduling and Control Using Simulation Technology", *Modern Machine Shop Conference, Shop Control '87*, March, 1987.

MacFarland, Douglas, "Tutorial - Scheduling Manufacturing Systems with FACTOR", *Proceeding of the 1987 Winter Simulation Conference.*

Pritsker Corporation, *Output Analysis System*, FACTOR User Guide, Version 4.0, 1989.

Pritsker Corporation, *Site Specific Tailoring, Volume 1*, FACTOR User Guide, Version 4.0, 1989.

Pritsker Corporation, *User Interface Tailoring*, FACTOR User Guide, Version 4.0, 1989.

Pritsker Corporation, *Base System, Conveyor Systems, Tooling Systems Autoguided Vehicles, Storage and Retrieval Systems*, FACTOR Implementation Guide, Version 4.0, 1989.

AUTHORS' BIOGRAPHIES

DAVID P. YANCEY is a Vice President at Pritsker Corporation. He received his Ph.D. from Purdue University in Industrial Engineering in 1981. Currently, he leads the Product Research group and is involved in simulation research and long range product planning. He is a member of TIMS, SCS, and ACM.

SCOTT PETERSON is a Senior Systems Analyst at Pritsker Corporation. He received his MS from Purdue University in Computer Science in 1987. Mr. Peterson led the product team which developed the OAS and SST modules for FACTOR. He is currently working on new FACTOR extensions which will support the manipulation of schedules and collection of status data.

David P. Yancey, Ph.D.
Scott Peterson
Pritsker Corporation
1305 Cumberland Avenue
West Lafayette, IN 47906, U.S.A.
(317) 463-5557