Flexible simulation of a complex semiconductor manufacturing line using a rule-based system

by B. R. Tibbitts

Rule-based systems have been used to produce fast, flexible simulation models for semiconductor manufacturing lines. This paper describes such a rule-based simulator for a semiconductor manufacturing line, and the language in which it is written. The simulator is written in a rule-based declarative style that uses a single-rule "template" to move thousands of product lots through various process steps; the rule is customized as needed with data for each step, route, lot, tool, manpower skill, etc. Since line or product changes require only reading new data from a database, without reprogramming, this provides a modeling environment that is simple, flexible, and maintainable. The model is implemented in ECLPS (Enhanced Common Lisp Production System), also known as a knowledge-based or expert systems language. It handles very large models (thousands of data elements, or more) well and is very fast. Subsequent changes improved the speed several orders of magnitude over that of an older version of the model, primarily through use of a preprocessor to eliminate duplicate and redundant data, and by enforcing data typing to take advantage of special techniques for very fast processing of extremely large matches (hashed indices). ECLPS also provides a built-in simulated time clock and other constructs to simplify simulation applications. The model runs daily at the IBM semiconductor manufacturing plant in Yasu, Japan, where it has been in use for many years, currently on three different semiconductor manufacturing lines.

Introduction

While computer simulation of manufacturing lines is not a recent development, the past few years have witnessed renewed interest in simulations of semiconductor manufacturing lines. Semiconductor manufacturing systems typically include machine tools that are expensive, may be used for a wide variety of operation steps, and must be reconfigured between different uses. This highly reentrant flow of the routing means that a wafer lot may be processed many times on a single machine, with many setup changes on the equipment, such as changes in the recipe, temperature, and process time of a wafer batch. The complexity and cost of semiconductor manufacturing lines are also due to their sheer size, such as the number of steps necessary to produce a wafer (hundreds), length of production turnaround time (months), cost of work in progress (millions of dollars), value of manufacturing tools (millions of dollars), and the variety of employee skill levels required to operate the machine tools. Thus, it is difficult to assess the effect of different manufacturing techniques and managerial strategies on the manufacturing line. A very small change in productivity can mean a big difference in the total value of the output. Product contamination is more apt to occur during production line delays, reducing yields and thus increasing costs.

Because of the large size of semiconductor computer models, their complexity has led to difficulties in accurately describing the complex processes that occur at each step of the manufacturing process and how they are handled, difficulties in generating accurate input and keeping it up-to-date, and difficulties in analyzing the large amount of output and finding meaning in its results. For example, the body of input data can be extremely large and complex, such as the WIP (work in progress) data. Since the turnaround time for a single lot of wafers (total time to manufacture a lot, start to finish, in real time) can be months, there is obviously a large amount of WIP in the line at any point in time. Any model used for accurate predictions must be able to accurately input the actual WIP at any point in time in order for the model to be used on real manufacturing line data.

Traditional simulation languages, such as GPSS (General Purpose Simulation System [1]), construct a detailed model, require a detailed description of each step, in a sequential manner, and thus are difficult to construct accurately, and especially difficult to maintain as operations change. Each step in the manufacturing line is several "lines of code," and there are many blocks of similar procedural descriptions making up the entire manufacturing line. Understanding this, at a high level, or even making changes to small parts, is more difficult as the size and complexity of the manufacturing line increase. Even if the process can be documented in a model, the sheer size of the result often makes it difficult to run in a reasonable amount of time (say, a few minutes or even hours). In addition, the original authors of the model found traditional simulation languages (such as GPSS) difficult to use where the initial state of the model included actual WIP data. The simulation could not start with empty queues.

Newer computers and simulation tools offer increased capacity of models, but the real problem is being able to accurately describe how the manufacturing line operates, in the language of the model. A secondary problem is being able to make a simulation run in a short time. Ideally, a run should take a few minutes or less, so that "what if" scenarios can be played out in order to determine the effect of different manufacturing strategies. However, it is not unusual for even *hours* of CPU time to be considered acceptable in most simulation tools. Some tools use graphics and animation to show the operation of a simulation model and its results, but still fail to address the issues of size and complexity.

Other tools also utilize a separation of data and rules to assist in changes, as in another IBM semiconductor manufacturing model [2], but still require the considerable expertise of a programmer to make any changes other than to the data. Another IBM model [3] uses a modular structure that consists of a number of precompiled FORTRAN subroutines describing different specific operating strategies. This increases efficiency and allows algebraic computation, which is not available in the base modeling language. The user picks from among these modules to construct a larger model. This can require reprogramming if operating strategies other than those foreseen by the original authors are encountered, but it restricts the reprogramming to those areas needing change.

This paper describes a simple, easy-to-understand technique for modeling a semiconductor manufacturing line. A single-rule "template" is used to move thousands of product lots through various process steps; the rule is customized by the different input data needed for each step. Because the actual details of the line are read as input from a database, the number of machines, manpower skills and their availability, work in progress, wafer starts, etc. are dynamic input to the model and do not require reprogramming. A declarative style of programming, also known as rule-based programming, is used rather than the more traditional procedural style. A skeleton "module," consisting of input, process, and output, is simply described in a *rule*, and the input is varied to simulate many different processes with the same rule, or a relatively small number of rules. The language used in this model is ECLPS (Enhanced Common Lisp Production System) [4]. It is a rule-based language which was originally based on OPS5 [5], but offers many improvements in flexibility, expressiveness, and performance. Very large models are easily described in ECLPS (**Figure 1**), and they are

efficiently executable because of the performance strengths of ECLPS, which are described in this paper. The simulation model is currently in use for three different semiconductor manufacturing lines at the IBM plant in Yasu, Japan. It is used daily for a variety of planning purposes. Because of its success, it is currently being adapted for use by a customer's semiconductor manufacturing lines as well. The current model is based on one originally developed by Keiji Ohmori [6] and written in YES/OPS [7], an OPS5-like language with specific extensions for a built-in simulated clock and timed events for simulation models, and a predecessor of ECLPS. Subsequent modifications in ECLPS have simplified and improved the performance over that of the original simulation model.

Why the ECLPS rule-based language?

When the original model [6] was constructed, three criteria were used for tool selection:

1. Ability to provide detailed description of the semiconductor log.
2. Flexibility for making changes to routes, products, tools, etc.
3. Connectivity to the data collection system and basic database tools.

Some conventional simulation tools were tried, but none met the above requirements [8] for the anticipated production volume increases of the Yasu semiconductor manufacturing line.

ECLPS offers many features to make it a good choice for the simulation as a modeling technique in general, and for large models in particular. Based on compiled Common Lisp, it is an efficient language in terms of execution time.[1] The rules allow for a declarative style of programming, and the built-in simulated (or real) time clock makes simulation straightforward. ECLPS can be used to model events that are scheduled to execute at a specific time in the future (either real or simulated time), via the timer queue, *or* trigger on specific events (e.g., machine and manpower availability) via rule matching. Unlike many other rule-based languages, however, ECLPS is completely integrated with its underlying procedural language (in this case, Lisp). Because of this, it can easily be used to code procedural constructs, such as dynamic algebraic calculation of time required on a particular machine, as well as the rule-based declarative constructs, such as the description of how lots move from one step to the next. Another feature of the improved version of the model (see the section on simulator performance improvements) is the linking up of routing steps by the use of data pointers stored directly in the data elements. This enables the linking of the routing steps, for example, without requiring run time matching to find the next step. While this is a common practice in many languages, this feature, which is totally supported in Lisp, a well-known rule-based language for artificial intelligence applications, is not available in many other rule-based languages, which restrict the types of objects to numbers, symbols, strings, etc.

ECLPS, like other rule-based languages, provides for a division between the rules and data. This was exploited by coding the operation/dispatching knowledge as rules, and the specific layout, routing, and status of the manufacturing floor as data. This allows great flexibility in being able to change routes, numbers of tools and their logical grouping, manpower allocation, etc. without changing the rules (source code). The input data were simply updated.

The pattern matching of ECLPS rules is used to match product lots with the necessary tool, manpower, etc. required at each process step. ECLPS rule priorities are used to sequence and to resolve conflict among the lots in contention for available resources.

ECLPS provides a good interface to the database, in which the manufacturing line data and results are stored, and are easily accessible by other tools for analysis both before and after the ECLPS simulation model is run. Users have access to both the input data and the forecasting result of the model, including the actual event log, in database form. Finally, ECLPS is designed so that only a limited knowledge of Lisp is required to get started.

	Line A	Line B
Total number of WMEs (data elements)	4647	2233
Route steps	3218	428
Number of routes	148	38
Average number of steps/route	22	25*
Number of lots	727	162
Tool groups	203	51
Tools	276	497
Manpower schedule changes	1259	6
Number of rule firings	6479	5340
Typical run time:	12 min	8 min

*Not including rework steps

Figure 1

Scope of size of the model. Data for models of two different Yasu semiconductor manufacturing lines. (WME = working memory element).

[1] Older versions of Lisp have a bad reputation for being slow in execution time, because they were interpreted instead of compiled. Today's Lisp compilers rival other modern programming languages in their efficiency.

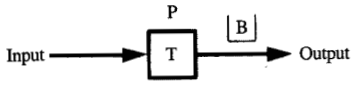

Figure 2

A simple step: input, process, and output via buffer. Process P takes input, processes it using tool T, then puts the output into buffer B before proceeding to the next step.

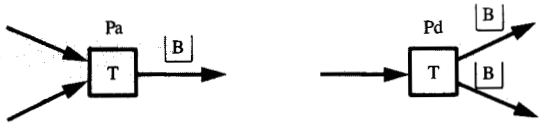

Figure 3

Assembly and disassembly steps.

How rules model production line steps

Each step in the semiconductor manufacturing line (or other assembly or production line) can be modeled by a single rule. (In very complex cases, more than one rule may be used.) All steps of the same basic type can be modeled by *one* rule. The data about each step are fed into the rule, causing it to act differently for each particular step, and customizing the parameters for that specific step, route, machine, operator skill, etc.

There are several types of production steps, of which a few simple ones are described here. Because of the flexibility of ECLPS, even very complex steps can be described by matching route/step and product lot data with a different type of rule(s).

• *Basic rule types*

A simple production step takes input, say a wafer[2] lot, from a buffer (a holding place for lots awaiting processing), waits for machine and manpower resources to become available, *matches* those items together in a rule, and then *fires* (executes) that rule by utilizing the resources for the appropriate amount of time, then putting the product lot in the buffer for the next process and freeing the resources for other operations. **Figure 2** shows a diagram of a simple step: A lot comes in, it is processed using tool T (and also other resources such as manpower, not shown), and then goes into the output buffer B to wait for the next step.

While most of the semiconductor manufacturing rules are *processes* that are applied to lots, as opposed to assembly of multiple parts into a larger unit, assembly and disassembly are also easily expressible. **Figure 3** shows such a step. Input could come from, say, two sources and be combined into a single output, or vice versa.

Several simple process steps can be chained together to form a small line, as shown in **Figure 4**. By limiting the buffer sizes between the steps, this manages the amount of work in process (WIP) and thus is a simple form of *kanban*, or control of in-process inventory. The product lot leaving the buffers is used as the "pull" signal to the previous step, telling it to continue.[3] Manpower resource (not shown) is also required, and may vary at each step with respect to both amount of time needed and operator skill.

Many production steps need not wait for a particular tool, but can be served by one of many. This is modeled by allowing the production step to use one of any number of tools in a *tool group*. Since tools are flexible, some tools can be a member of more than one tool group. **Figure 5** diagrams this situation.

We next demonstrate how ECLPS rules can be used to model these situations.

• *ECLPS basics*

The ECLPS language, in which the simulator is written, consists of three parts:

- Data (known as working memory elements, or WMEs). WMEs each have a class and any number of attributes (like a record with any number of fields within the record). Attribute names begin with an asterisk (*). For example, class product may have attributes *lot-num and *route.
- Rules, with patterns to match (left-hand sides) and actions to perform (right-hand sides).
- The inference engine, the part of ECLPS that matches data (WMEs) with rule patterns (left-hand sides), determines the order in which rules are fired, and executes the rules' actions.

In the factory simulator, WME data are entered for

- Product lots (both for start-to-build and work-in-progress).
- Tools.

[2] A *wafer* is a unit of semiconductor manufacturing.

[3] In pure *kanban*, by some descriptions, the step must also wait for a separate *pull* signal from the step ahead in order to go ahead and process a lot at the current step. In the case of this simple model, a buffer with available capacity is considered to be the "signal" to initiate a process step. That is, the fact that a succeeding step has pulled lots out of a buffer is a signal to put more lots into the buffer.

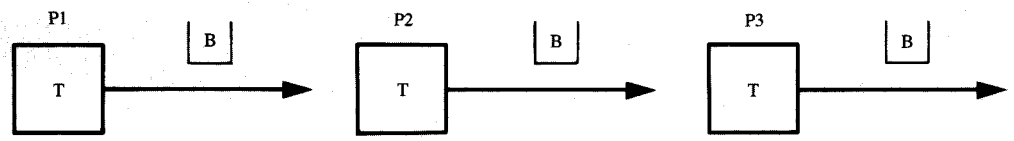

Figure 4

Three processes, each using a single tool and buffer. Limiting buffer sizes manages the WIP and thus is a simple form of *kanban*.

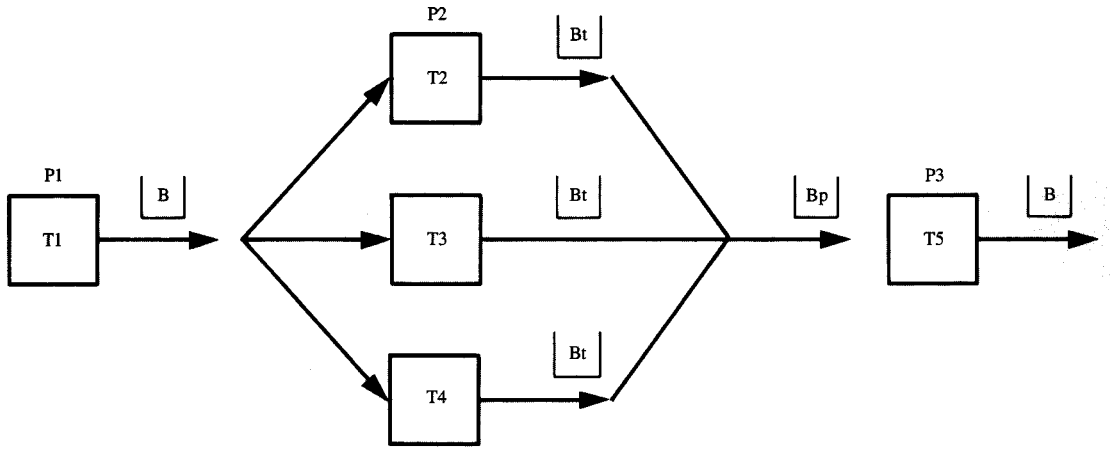

Figure 5

Three processes, the second process utilizing a tool from a tool group. Process P2 may utilize any of three tools from a tool group. Logical buffers are maintained both by tool (Bt) and by process step (Bp), so a lot will sit in both Bp and the appropriate Bt after being processed in step P2. Neither buffer may accept a lot if its maximum capacity is filled. This manages the WIP as in Figure 4 but for multiple equivalent tools.

- Tool groups of functionally equivalent tools.
- Manpower quantities available by hour, by skill level.

(Buffers and lists of the route steps are described later.)

- *ECLPS rules*

ECLPS rules can use these data structures to model the simple processes described earlier. **Figure 6** shows a very simple ECLPS rule that would handle the steps for simple single-tool processes, such as those in Figure 4.

The basic steps (buffers are added later) are as follows. The parentheses indicate what the ECLPS inference engine does at each step.

1. Wait for lot, machine tool, and manpower resources (match).
2. Utilize the machine and manpower resources to process the lot (fire).
3. Free the machine and manpower resources after the appropriate amount of time (timed events scheduled for a later time).

```
(defrule simple-step
   when
      (product-lot *lot-num <L> *tool <t> *skill <sk>)
      (tool *id <t> *status ready)
      (manpower *skill <sk>)
   then
      .... )
```

Figure 6

Simple ECLPS rule production step—product lot, tool, and manpower.

```
(defrule sample-toolgroup
   when
      (product-lot *lot-num <L> *tool <tg> *skill <sk>)
      (toolgroup *id <tg> *tool <t>)
      (tool *id <t> *status ready)
      (manpower *skill <sk> *max <mp> *in-use < <mp>)
   then
      .... )
```

Figure 7

Production step with tool group. Variable <tg> matches the tool group required by a lot to one instance of a tool group, that of a specific tool whose id is saved in variable <t>. Then variable <t> matches the tool with that id, as in a database *join* operation.

Steps 2 and 3, rule firing and timed-events scheduling, are discussed later. Here we concentrate on describing how patterns are written, and the match process.

Read the rule in Figure 6 as follows: When there exists a product lot of some lot number <L> which requires a certain tool (matched by variable <t>) and an operator skill of type <sk>, and a tool of that type and the appropriate manpower are available, the rule "fires." The <x> designates an ECLPS *variable*. The first time it is encountered, it *binds* to any value, but for subsequent uses, it must *match* data with the same value, effecting a *join* across data elements with the matching attribute values. Variables can also be used to do comparison tests between attributes in the same data WME, or across different WMEs (the *join* operation), as in the third line below:

```
(product-lot *lot-num <L> *tool <t> *skill <sk>)
(tool *id <t> *status ready)
(manpower *skill <sk> *max <mp> *in-use < <mp>)
```

In this case, the manpower value <mp> is bound to the maximum available manpower of a certain skill, and a test is performed to ensure that the amount of manpower of that skill currently in use is *less than* that amount.

Now let us add the ability to choose any available tool from a group of tools.

Read the rule shown in **Figure 7** as follows:
When there exists

- a product-lot of some lot-number <L> which requires a tool of some tool group <tg> and an operator skill of type <sk>,
- and there exists a tool in tool group <tg> named <t>,
- and tool <t> is available and ready for use,
- and there is at least one of the appropriate manpower available (*max is the number currently staffed; *in-use is the number currently being utilized, and thus not available for other work),

then the rule can fire.

When a rule fires, it marks the tool as busy, marks the manpower in use (increments the number in use), waits an appropriate amount of time (explained in more detail later), then releases the tool and manpower. (This part is coded after the then symbol in the rule and is described fully later.)

Each product lot utilizes this rule[4] at each step. Notice how the values that are matched by the variables state the particulars of the step to be processed. These values are contained in the input data. Some samples are shown, beginning with **Figure 8.**

- *Adding buffers*

Actually, the *kanban* implementation limits WIP data by restricting the sizes of the buffers between the steps. When the buffer testing is added, the basic steps become the following:

1. Wait for lot, machine, and manpower resources. The machine's output buffer must have available capacity left.
2. Take the lot out of the previous machine's buffer and use the machine and manpower resources to process the lot.
3. Free the machine and manpower resources after the appropriate amount of time. Put the lot into the output buffer of the machine just used.

[4] In this simple case where the same basic algorithm is used to move lots from one step to the next, a single rule handles most steps. In more complicated cases, many rules could compete for handling each process step, dynamically computing the "best-fit" of each potential way to handle the step, by each different rule.

ROUTE: Route1

OPER(step)	TOOL	NEXT-OPER(step)	SKILL-REQUIRED
0 (start)	-	1	
1	t1	2	skilla
2	tg1	3	skillb
3	SHIP	-	skilla

STEP-BUFFERs:

FOR(route)	STEP	SKILLID	QUANTITY
route1	0 (start)	-	0
route1	1	skilla	0
route1	2	skillb	0
route1	3	skilla	0

TOOLGROUP: tg1 (contains 3 tools)

TOOLID
- t2
- t3
- t4

TOOLS:

RESOURCE	NUMBER	OCCUPIED?	STATUS
t1	1	no	ready
t2	1	no	ready
t3	1	no	ready
t4	1	no	ready

TOOL-BUFFERs:

RESOURCE	BUF-SIZE	TOT-BUF-SIZE	QUANTITY
t1	6	6	0
t2	3	9	0
t3	3	9	0
t4	3	9	0

PRODUCT:

LOT-NUM	PLACE	ROUTE	OPER	NEXT-OPER
1234	buffer	route1	0	1
5678	buffer	route1	0	1

Manpower (at model startup. Will hold skills/quantities available,
and number in use, at any time)

SKILLID	MAX-MP	IN-USE
skilla	0	0
skillb	0	0

MANPOWER-SCHEDULE (indicates CHANGES to skill levels, and times)
Utilizes timer-queue to modify MANPOWER wme to show current staffing level at any time

SKILLID	DATE	TIME(HOUR)	NUMBER-AVAILABLE
skilla	7/1/92	8:00	6
skilla	7/1/92	9:00	8
skilla	7/1/92	12:00	10
skillb	7/1/92	8:00	4
skillb	7/1/92	10:00	4
skillb	7/1/92	11:00	6

Figure 8

Sample data for a four-step route. Data are matched with *rules* to simulate the actions that occur at each process step.

The production line is modeled by combining and building upon these simple building blocks.

The next section describes more fully how the actual rules work, including how the route/step information is added. Some sample data that fit the problem are also shown.

How the ECLPS rule-based factory floor simulator works

We now describe in more detail how the actual model works. As discussed earlier, the factory simulator includes WME data entered for

- Product lots (both for start-to-build and work-in-progress).
- Tools.
- Tool groups of functionally equivalent tools.
- Manpower quantities available by hour, by skill level.
- Routes (lists of operations through which a type of product lot progresses).
- Buffers (both by route/operation step and by tool).

(The latter two were not described previously.)

A few start-up rules take the initial data (product lots and manpower quantities) and utilize the ECLPS timed-event feature to put events on the timer queue that will execute when an event must be processed. For example, at a certain time, a manpower quantity for a certain skill may have to be changed. At another time, a new product lot may have to be available for start-to-build. A simulated clock within ECLPS keeps track of the time within the model.

Once these events are on the timer queue, six rules handle the basic running of the model, for a line with simple input-process-output types of operations. There are three types of rules:

1. Rules that take a start-to-build product lot and wait for the resources required for the first operation to be available (not shown; fires once per lot).
2. Rules that move a product lot from one step to the next, using *kanban* techniques. The majority of the product lot movement is the result of this and similar rules.
3. Rules that move a product lot from the last step and remove it from the model (completion, or shipping).

For each of these three rule types, there are two versions of the rule: one for unique tools, and one for tool groups. Tool groups are sets of functionally equivalent tools. Thus, a lot may be waiting in a buffer for a specific tool, or it may be waiting for any one tool that is a member of a specific tool group. The conflict resolution of ECLPS (the ordering of which rules, with which matching data, fire in what order) determines the order in which the rules fire, which is the order in which the lots are processed from one tool to the next. If several lots are ready and have all resources available, which one receives the (limited) resources first? (In this model, such events are controlled in detail by dynamic rule priorities based on the product lot and other data.) When the rule fires, the initial work on the RHS (right-hand side, or action part of the rule) marks the tools and manpower as "in use," then adds events to the timer queue to remove the lot when the process is complete, mark the tool and manpower free to be utilized by other lots, and put the lot into buffers for the next tool and operation step.

- *Route/step sequencing*

The actual steps that each lot type goes through are described via a set of route WMEs. One WME corresponds to each step, and each WME contains a pointer to the next step. Each of these contains information about the tool (or tool group) required by the lot at this step. Some examples are shown in Figure 8 and **Figure 9.**

- *How the timer queue schedules events*

The ECLPS *timer queue* is used to schedule events to take place at some predetermined time in the future, which is independent of the rules firing. For example, the manpower WMEs hold, at any point in time, information on the manpower available, by skill, at that time. In order to accomplish this, the manpower schedule, hour by hour, is input at the start of the model, and an event is scheduled for each *change of manpower available*, by skill. At each rule firing cycle, the timer queue is checked to see whether any events are ready to execute; if so, they are executed *before* the next rule fires. Thus, there could be hundreds (or more) of elements of manpower schedule changes, each eventually represented by a single WME that automatically reflects the right manpower available at the right time. The manpower changes are scheduled on the timer queue to ensure that they are processed at the right time.

Putting items on the ECLPS timer queue consists of specifying the time (relative or absolute) and the action to take place. Some examples are shown in **Figure 10.**

- *Detailed rule, with actions*

A typical rule for moving lots between process steps is now described in more detail. The basic type of rule in the factory simulator is one whose patterns (left-hand side) match a product lot waiting in a buffer, along with the tool and manpower needed to perform the next production step. The patterns also wait for the availability of buffers that have available capacity. There are actually two *logical* buffers in which the product lots sit after each step: buffers

```
; Routes:  3 steps, start, 1, 2(toolgroup), end
: (Note: skill-id required for the step is stored in the corresponding step-buffer)
 (MAKE route *r-name route1 *oper 0                  *next-oper 1)
 (MAKE route *r-name route1 *oper 1   *tool t1       *next-oper 2)
 (MAKE route *r-name route1 *oper 2   *tool tg1      *next-oper 3)
 (MAKE route *r-name route1 *oper 3   *tool ship                 )
; Step-buffers:
 (MAKE step-buffer *for route1  *step 0              *quantity 0)
 (MAKE step-buffer *for route1  *step 1 *skillid skilla *quantity 0)
 (MAKE step-buffer *for route1  *step 2 *skillid skillb *quantity 0)
 (MAKE step-buffer *for route1  *step 3 *skillid skilla *quantity 0)

; Toolgroup: tools t2,t3,t4 are equivalent tools in toolgroup tg1
 (MAKE toolgroup *tool-group-id tg1 *tool-id t2  )
 (MAKE toolgroup *tool-group-id tg1 *tool-id t3  )
 (MAKE toolgroup *tool-group-id tg1 *tool-id t4  )
; Tools:
 (MAKE tool *id t1  *num 1  *occupied no *status ready)
 (MAKE tool *id t2  *num 1  *occupied no *status ready)
 (MAKE tool *id t3  *num 1  *occupied no *status ready)
 (MAKE tool *id t4  *num 1  *occupied no *status ready)
; Tool-buffers:
 (MAKE tool-buffer *id t1  *buf-size 6 *tot-buf-size  6  *quantity  0 )
 (MAKE tool-buffer *id t2  *buf-size 3 *tot-buf-size  9  *quantity  0 )
 (MAKE tool-buffer *id t3  *buf-size 3 *tot-buf-size  9  *quantity  0 )
 (MAKE tool-buffer *id t4  *buf-size 3 *tot-buf-size  9  *quantity  0 )

; Product lots:
 (MAKE product *lot-num 1234 *place buffer *route route1 *oper  0 *next-oper 1 )
 (MAKE product *lot-num 5678 *place buffer *route route1 *oper  0 *next-oper 1 )

; Manpower at startup - will contain info for current time, by skill
 (MAKE manpower  *skillid skilla  *max-mp 0  *in-use 0)
 (MAKE manpower  *skillid skillb  *max-mp 0  *in-use 0)

; Manpower schedule for three hours, beginning 8:00AM on 7/01/92
 (MAKE manpower-schedule *skillid skilla  date 19920701 *time 8   *number-available 6)
 (MAKE manpower-schedule *skillid skilla  date 19920701 *time 9   *number-available 8)
 (MAKE manpower-schedule *skillid skilla  date 19920701 *time 12  *number-available 10)

 (MAKE manpower-schedule *skillid skillb  date 19920701 *time 8   *number-available 4)
 (MAKE manpower-schedule *skillid skillb  date 19920701 *time 10  *number-available 5)
 (MAKE manpower-schedule *skillid skillb  date 19920701 *time 11  *number-available 2)
```

Figure 9

ECLPS statements to create data described in previous figure. Other information is also stored in each of the data elements (*working memory elements*), but is omitted here for simplicity's sake. Examples are priorities, times for the steps, etc.

```
(at "3:00 pm 7/1/92" do (make manpower *skillid skilla *quantity 3))
(in "4 hours" do (make product *lot-num 1234 . . . ))
```

Figure 10

Samples of putting events on the ECLPS timer queue. At the specified times (absolute **at** or relative **in**), the actions will take place and will be included in subsequent matching and actions in the model.

```
; Sample rule—BUFFER to PROCESS and back
When
  there is a waiting product with a next-operation-step
  and that next-operation-step of the route needs a tool of a certain tool group
  and there is a tool of that tool group available
  and the buffer for this tool is not full (tool-buffer, for all steps requiring the tool)
  and the buffer for this step is not full (step-buffer, one per step)
  and the correct skill of manpower is available
Then
  Mark the lot as in-process
  Decrement the quantities in the step-buffer and tool-buffer
  Mark the tool as busy
  Mark one person (of the correct skill of manpower) as in use
  Calculate the amount of operator attended time
  Calculate the amount of machine-in-use time
  Schedule the following events to occur:
     Release of manpower after operator attended time is over
     Release the tool (mark it not-busy) when step is complete
     Mark the product as waiting for the next step
     Increment the next buffers in which product will wait
       (tool-buffer and step-buffer)
```

Figure 11

English version of a rule.

by tool, and buffers by process step. These are *output* buffers, which hold lots that have *completed* the associated tool and step. Only when both buffers are not too full can the product lot move through the next step (a form of *kanban* control policy). When these rule patterns find a complete set of data to match (product, tool, not-too-full buffers, manpower, etc.), the product lot is ready to move along to the next step. The rule *fires*, causing the *right-hand-side* actions to be performed: decrementing the buffer quantities, marking the lot, tool, and manpower as in-process (in use), and scheduling the release of these resources and then the incrementing of the subsequent buffers, in which the lot waits for the next step. **Figure 11** shows an English rendition of a sample rule, and **Figure 12**

```
; Sample  ECLPS rule to move product lot through one step: from buffer to process and back
;     <x> indicates an ECLPS variable

(DEFRULE transd02 WHEN
<prod>    (product   *lot-num <lot> *place buffer *route <r> *next-oper <op> *c-priority <cp>
                     *tool-buffer-wme <prev-tb> *step-buffer-wme <prev-sb>)

<route>   (route *r-name <r> *oper <op> *tool <cur-tool-grp> *next-oper <next-op>)

   <tg>   (toolgroup *tool-group-id <cur-tool-grp> *tool-id <cur-tool>)

  <tool>  (tool *id <cur-tool> *num <toolnum> *occupied no  *status ready)

   <tb>   (tool-buffer  *resource <cur-tool> *buf-size <step-max> *tot-buf-size <tool-max>
                        *quantity <tb-quan> & < <tool-max>)

   <sb>   (step-buffer *for <r> *step <op> *skillid <sk> *quantity <sb-quan> & < <step-max>)

  <mp>    (manpower *skillid <sk> *max-mp <mm> *in-use <ac> & < <mm> *status current )

          (task        *task-id usual *priority <pr>)

          (run         *go on  *from <tm> *start-shop <sts>)

THEN PRIORITY (0  -  (<pr> + <cp>)) ;Rule priority based on values matched above (task & lot)

   (MODIFY <prod> *place process  *oper <op> *next-oper <next-op>
                  *tool-buffer-wme <tb> *step-buffer-wme <sb>)

   (decrement-tool-buffer-quantity <prev-tb>)
   (decrement-step-buffer-quantity <prev-sb>)

   (MODIFY <tool> *occupied yes *status busy)

   (MODIFY <mp> *in-use (<ac> + 1) ) ;increment manpower counter

   ;  Set temporary variables for manpower time and tool time required
   (LET ((<mtime> (calculation-of-manpower-time <prod> <route>))
         (<ttime> (calculation-of-tool-time <prod> <route>)))
     ; Release manpower after attended operation time
     (IN  "<mtime> minutes" do (decrement-manpower-counter <mp>))
     ; Release tool and put it into appropriate output buffers
     (IN  "<ttime> minutes" do
       (MODIFY <tool> *occupied no *status ready)
        ;; put lot back in buffer  (priority computation omitted)
       (MODIFY <prod>  *place buffer )
       (increment-step-buffer-quantity <sb>)
       (increment-tool-buffer-quantity <tb>)))
)
```

Figure 12

ECLPS version of a rule.

```
(defrule remove-dups when
 <c1>  (step-buffer *for <route> *step <s> *skillid <k> *wme-time-tag <tt>)
 <c2>  (step-buffer *for <route> *step <s> *skillid <k> *wme-time-tag > <tt>)
   then priority 99999
   (remove-wme <c2>))
```

Figure 13

An ECLPS rule used to detect and remove duplicate buffers. This rule detects the existence of two step-buffers for the same step, same operator skill, and different timetag (to assume uniqueness) and removes the second one.

the actual ECLPS rule. Note that the entire simulation (moving lots from step to step) can be modeled by this one rule (or two rules if a special-case rule for tools not belonging to a tool group is used). In this case, the same basic algorithm is used to move each lot from step to step, using the data from the lot and the step to determine the details of each step.

The RHS (right-hand side, or action part of the rule—after the THEN symbol—contains the actions that are to take place when the LHS patterns are all matched consistently and the rule is chosen to fire. Typical actions are make, modify, or remove of working memory elements (data, often those matched by the LHS patterns). Scheduling events to occur in the future, via the timer queue, is also typical for an RHS action in a rule. Other actions can include calculations, such as the algebraic calculation of time required for the attended and nonattended portion of the process step, or any arbitrary procedural statements. Since ECLPS is based on Common Lisp, this means that any Lisp code may be included in the RHS actions and is executed when the rule fires. This allows for extreme flexibility in rule action expression. Additionally, external functions can be called, even those written in other programming languages. Match statements, which allow *on-demand* pattern matching of ECLPS patterns with WME data, may be included.

Simulator performance improvements

The original translation of one of the versions of the model from the earlier YES/OPS language to ECLPS did not meet the expectations for improvement in performance, primarily because of a combinatorial explosion in the match process. Thousands of data elements matched with a dozen or more rule patterns and created a huge combinatorial explosion of possible matches for each rule.

The model's performance was analyzed and some changes were made. The most significant changes made were the following:

1. Several short Lisp functions and ECLPS rules were written that acted as preprocessor programs, sorted through the data, and allowed identification of duplicate or otherwise redundant or unnecessary data. (The amount of data was so large that the model was difficult to analyze.) Elimination of these data greatly reduced the combinatorial complexity of the rule matching. Actually, ECLPS rules themselves are a natural way to express data redundancy checks, and are quite useful for this task. **Figure 13** is an example of an ECLPS rule used to detect and remove duplicate buffers. It matches any two *different* step-buffers for the same route, step, and skill id, and removes one of them. By testing the uniqueness of the timetag attribute (the built-in *wme-time-tag attribute is always unique), the rule ensures that the step-buffer is not matching itself.
2. Type information was added to the data, and the data types that could be used within the data attributes were only slightly restricted, thus enabling hashing of indices for speeding up the *very* large matches (as in this model). This *EQ-hashing* is a specific enhancement that ECLPS added to the original RETE algorithm [9]. The RETE algorithm is the essential pattern-matching algorithm part of ECLPS [10], making the matching (of many patterns with large amounts of data) efficient. **Figure 14** shows how this is made possible with the :type keyword of the data structure declaration, defwme.

These two changes resulted in large (but expected) performance improvements. Other, more subtle, changes

```
(defwme route   *r-name      (: type symbol)     ; name of route
                *oper        (: type fixnum)     ; step (operation) number
                *next-oper   (: type fixnum)     ; next step number
                *tool        (: type symbol)     ; resource (tool/toolgroup)
                )
```

Figure 14

Type declarations increase match efficiency. Simple declarations can produce tremendous decreases in execution time.

```
(defrule detect-stuck-lots
 when
  <p>  (product  *lot-num <m> *route <r> *place buffer *next-oper <op>)
       - (step-buffer *for <r> *step <op>)
 then priority 99999
   (say "Step-buffer missing for route" <r> "step" <op>))
```

Figure 15

ECLPS rule to detect an artificial bottleneck (stuck lots). This rule finds missing buffers — that is, whenever a product is in a buffer, such that there does not exist (the "-" sign) a step-buffer for the next step in that route, a message is displayed. The high priority of this rule (999999) ensures that the warning is displayed promptly after the condition appears.

reduced the amount of matching done by moving some matches to the RHS of the rule, or eliminating the matching altogether by simply storing pointers to the values that must be tracked and modified. This is a well-known programming technique, but it is not available in many rule-based languages. A variety of measurement tools were used to find data redundancies, artificial bottlenecks in the model, and other criteria that were hard to see and measure otherwise. **Figure 15** is an example of an artificial bottleneck (missing buffers) that appeared as "stuck lots" in the model and was detected with the ECLPS rule shown.

The model execution speed improved so much that the original body of data was expanded to include more of the actual manufacturing line. Because of the original slowness, only about half of the steps of each of the routes had been included in the old model, which added to the estimates of the time required to account for the omitted steps. With the increased capacity, this was expanded to reflect the actual operations more accurately.

Improvement, while impressive, is difficult to measure exactly. Because the original model was so slow, significant detail had been removed in order to obtain even a rough estimate of the factory line's function encoded into the model, and run. Yasu has been very satisfied with the results, but has not, to our knowledge, run the exact same data on the old and new versions of the model in order to have a direct comparison.

In summary, however, the new model is approximately *10 times faster*, for 15 times as many product lots, three

	Old	New
PROBLEM SIZE ATTEMPTED:		
Product lots	42	641
Route operation steps	554	1602
Simulation days	3	15

Summary: New problem being run is much larger (several orders of magnitude) than previously run.

COMPUTING RESOURCES USED:		
Type of CPU	3090 30S	3090 20E
MIPS (CPU speed)	55.5 MIPS	31.3 MIPS
CPU time	45:37 (2737 sec)	8:28 (508 sec)
Time*MIPS	151,903	15,900

Summary: New model is approximately
10 times faster, for
15 times as many product lots,
3 times as many route operation steps,
5 times as many simulated days.

CPU MIPS information from *Computer Price Watch*, Computer Information Resources, P.O. Box 13176, Arlington, TX 76094.

Figure 16

Summary of improvements made to the factory simulator.

times as many route operation steps, and five times as many simulated days. The details are shown in **Figure 16.** The improved model is now in daily operation in the Yasu plant. Since the successful operation of this model, Yasu manufacturing support personnel have converted another line to the ECLPS model, for a total of three semiconductor manufacturing lines being modeled with the ECLPS simulator at the Yasu plant. A version of the model is currently being developed to model the semiconductor manufacturing line of a customer as well.

The results of simulated lot completion, WIP, and tool and manpower utilization data are used to optimize lot start allocations in each process, as well as the general management of WIP, tools, and manpower. Simulation results are stored in local databases and are available for analysis and report by a variety of other tools.

Conclusion

Large complex simulation models, such as those needed for semiconductor manufacturing lines, require a modeling environment that supports a large number of elements (steps, lots, machine tools, etc.) and is easily maintainable. This model, with its declarative rule-based style, encodes the part of the system that describes the daily changes (lot starts, WIP, machine and manpower availability) in the *data* read from existing manufacturing database files. It handles a very large amount of data efficiently, and is easy to modify by reading in the appropriate data describing the appropriate line at any point. While the Yasu model consists mostly of simple input-process-output rules, much more complex rules[5] could be modeled because of the flexibility of ECLPS and its underlying language, Lisp. Procedural calculations can be done directly in Lisp from ECLPS, or external language subroutines can be called if necessary. The built-in simulated clock and timed events queue make it a very powerful and flexible simulation language.

Our experiences in improving the performance of this particular ECLPS simulation model stressed the importance of being able to take advantage of even more pattern-matching efficiencies in large models by understanding how ECLPS pattern-matching works, and by using *measurements* to find sources of inefficiencies in rule-based (and other) systems. A variety of measurement tools were used, some for Lisp, others for ECLPS, still others for this specific application. All are useful in targeting the source of performance problems.

"Expert system" tools are traditionally suited for encoding of "human expertise" that is not easily described in procedural languages. ECLPS is also good for describing procedural, but combinatorially complex, combinations of possibilities, such as that found in the semiconductor manufacturing line model. Its ability to describe and run very large models, with simplicity and maintainability, in a reasonable amount of time, makes it a powerful tool for the simulation model.

Acknowledgments

Many thanks to Keiji Ohmori, who developed the original model in 1986, and Mitsuru Takahara, who currently continues to improve and adapt the model to other semiconductor manufacturing lines. Both discussed the model and answered many questions over E-mail for this paper. The work on the performance improvements could

[5] While there is only one basic control rule in the model as described here, the dynamic calculation of (ECLPS rule) priorities for firing the rules (which in effect orders the lots that are handled by the machines) includes calculations based on product type, due date, most work remaining, shortest process time, etc. This uses the same basic algorithm, or rule (template) to handle many situations in a similar manner. More rules could be written to handle very diverse cases, and specific data in the product lot information, or route/step information, could be used such that the rules would "compete" for the best next-step or next-machine possibility. The rule that "won" would exclude the alternatives as the product lot continued along the production line.

not have been done without the cooperation and assistance of Takeo Matsuura of IBM Yasu, Japan, who maintained the model at that time. Thanks also to others who have maintained various versions of the Yasu models and have provided their insight, including Yuko Fujii and Yoshinori Hirohata. Additional thanks also to Marshall Schor for much discussion about the model improvements and the internals of ECLPS, and to Marshall Schor and John Kastner for proofreading the initial manuscript for this paper.

References

1. Thomas J. Schriber, *Simulation using GPSS*, Wiley, New York, 1974.
2. David J. Miller, "Simulation of a Semiconductor Manufacturing Line," *Commun. ACM* **33**, No. 10, 98–108 (1990).
3. Sarah Jean Hood, Amy E. B. Amamoto, and Antonie T. Vandenberge, "A Modular Structure for a Highly Detailed Model of Semiconductor Manufacturing " *Proceedings of the 1989 Winter Simulation Conference*, American Statistical Association, Washington, DC, pp. 811–817.
4. ECLPS (Enhanced Common Lisp Production System): *Enhanced Common Lisp Production System User's Guide and Reference*, Order No. SC38-7016 (formerly IBM Licensed Program 5685-063, now available from Lucid, Inc., 707 Laurel St., Menlo Park, CA 94025). For information about ECLPS, contact Lucid at (415) 329-8400 or E-mail: sales@lucid.com.
5. Charles Forgy, *OPS5 User's Manal*, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1981.
6. Keiji Ohmori, "Simulator for Event-driven System," *Proceedings of the 1986 Japan–USA Symposium on Flexible Automation*, Osaka, Japan, July 14–18, 1986, pp. 517–520. Sponsored by the Japan Association of Automatic Control Engineers and the American Society of Mechanical Engineers.
7. Marshall Schor, Timothy Daly, Ho Soo Lee, and Beth R. Tibbitts, "Advances in RETE Pattern Matching," *Conference Proceedings of AAAI-86* (American Association of Artificial Intelligence), Philadelphia, PA, 1986, pp. 226–232.
8. Mitsuru Takahara, IBM Yasu, Japan, "Floor Operation Planning and Simulation," *Technical Report TR-81.0113* (in Japanese); presented at the IBM Manufacturing Symposium, Thornwood, NY, May 1989.
9. Ho Soo Lee and Marshall I. Schor, "Match Algorithms for Generalized Rete Networks," *Artif. Intell.* **54**, No. 2, 249–274 (1992).
10. Charles Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artif. Intell.* **19**, 17–37 (1982).

Received March 19, 1992; accepted for publication February 29, 1993

Beth R. Tibbitts *IBM Programming Systems, 250 W. Main St., Lexington, Kentucky 40507 (BETH at LEXVMK, btibbitts@vnet.ibm.com).* Ms. Tibbitts is currently an Advisory Programmer for IBM Programming Systems, Lexington, Kentucky, working on object-oriented debuggers and other programming tools in C++. At the time this work was done, she was a member of the IBM Research Division at the IBM Thomas J. Watson Research Center in Yorktown Heights, New York, in the Environments for Expert Systems group (1985–1992). This group developed ECLPS (Enhanced Common Lisp Production System), an expert systems language based on Lisp, the language on which the simulator in this paper is based. Ms. Tibbitts received a B.S. in computer science and mathematics from Western Kentucky University in 1977. She joined IBM in 1977 and was a programmer, analyst, and manager in IBM Lexington, Kentucky, before transferring to Research in 1985. At Research she was part of ECLPS development and taught many classes on ECLPS and its forerunner, YES/OPS. She was also active in marketing and promotion of ECLPS both inside and outside IBM, wrote most of the ECLPS product manuals, and has advised and assisted users in working with ECLPS and other expert systems tools in various applications. She joined Lexington Programming Systems in 1992 and is now a member of a group implementing a C++ class library for debuggers and other programming tools. Ms. Tibbitts' interests include object-oriented tools and design, interactive languages and environments, and expert systems. She was chairman of the IBM Corporate APL ITL (Interdivisional Technical Liaison) Committee from 1983 to 1985 and was active in the use and promotion of APL. She is a member of the Association for Computing Machinery, the IEEE Computer Society, and the American Association for Artificial Intelligence.