Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Simulation modeling of JUST-IN-TIME assembly systems using an information-based methodology

Michael G. Ketcham

Dept. of Industrial Engineering &
Operations Research
University of Massachusetts, Amherst
Amherst, Massachusetts

ABSTRACT

This paper describes the application of an information-based modeling methodology to assembly systems in manufacturing. This approach defines information relationships among objects in a simulated system. A dynamically maintained database is explored as a simulation runs to control entity flow and requests sent from one entity to another. This allows a simulation to represent multi-product, multi-level assembly relationships, and to represent "pull" production scheduling as a special case of a more generalized request mechanism. In addition to manufacturing applications, assembly relationships characterize a large class of systems where the system logic depends on the simultaneous coordination of multiple entity flows and control depends on request messages.

1. Introduction

Assembly systems are among the most difficult to model in conventional simulation languages for manufacturing, since the product flow and assembly points for each part need to be separately modeled for each subassembly. Adding a new product or changing a bill of materials can mean significant reprogramming. Similarly, if several products with different requirements are assembled at the same location, the modeling network can become extremely complex. Assembly systems, however, constitute a significant portion of the U.S. manufacturing base, and assembly relationships characterize a larger class of systems, including communications systems, where the system logic depends on the simultaneous coordination of multiple entity flows and the control of systems based on requests to initiate an operation.

The basic information needed to represent an assembly system is quite simple. Representing such a system requires a list of work stations in the system (that is, a list of locations at which operations can be performed), a list of operations, and a list of entities involved in each operation, along with a bill-of-materials to represent product assembly requirements, and inventory control policies used to manage the availability of product subassemblies. A "bill-of-materials" is simply a hierarchically organized tree indicating which entities need to be simultaneously available before an operation can take place. By having a simulation read directly from an information model implemented as a database, we can easily model complex assembly systems along with related control policies, such as just-in-time (JIT) production scheduling.

This paper describes a modeling methodology we have called "information-based" modeling that defines relationships among objects in a simulated system using a dynamically maintained database. The database can be flexibly explored as a simulation runs to control entity flow and control requests sent from one entity to another. This allows a simulation to represent multi-product, multi-level assembly relationships in manufacturing, and also to represent "pull" production scheduling as a special case of a more generalized request mechanism.

This methodology is implemented in a system called IBIS. In the user's view, the IBIS database has a form similar to that of a production database that specifies bill-of-material and bill-of-resources relationships, product routing, product processing requirements, and work station configurations. The user's view of the database overlays highly flexible data structures that incorporate capabilities similar to those in object-oriented programming. These include class and inheritance relationships, request messages, and flexibly defined attributes that can be specified for any object in the system.

The paper briefly discusses the characteristics of flexible assembly systems and presents a hypothetical assembly system, and then discusses (1) the structure of the bill-of-materials, (2) modifications to queuing disciplines and and queue modeling used to synchronize entity flow, and (3) the request mechanism used to maintain inventory levels. Finally, the paper discusses elements of the simulation output and their implications for system control.

2. Problem Domain

2.1. Just-In-Time Flexible Assembly Systems

In many assembly facilities, assembly operations are accompanied by flexible assembly techniques and by demand pull production scheduling.

In flexible assembly systems, as found in electronics assembly, for example, multiple products may share assembly work stations but may require different types and numbers of subassemblies. Assembly requirements for multiple products can, of course, be specified using information relationships defined in a database schema. The schema can be defined to represent multi-level bills-of materials, and product routings for multiple products and subassemblies. Furthermore, by incorporating class relationships in the database design, the database can hold production requirements for classes of products in addition to requirements for specific products.

With demand-pull scheduling, such as just-in-time, production is initiated and subassemblies are drawn from inventory based on product demands, not material arrivals. In a traditional "push" production environment, subassemblies are "pushed" from operation to operation until they are either combined with a product or moved into inventory. In a "pull" environment, subassemblies are "pulled" from operation to operation in response to requests generated elsewhere in the system. Re-

quests can be enchained backwards from one operation to another until, eventually, subassembly materials are drawn from external suppliers.

These issues are closely related, since combining flexible assembly with "pull" scheduling offers significant benefits to industry by reducing work-in-process and increasing utilization for production resources. They also raise general issues in system modeling, such has the synchronization of multiple entity flows, flexibly defined "fan-in" for entity flow, and event scheduling based on demand.

2.2. Example

A hypothetical (and simplified) assembly facility will illustrate an information-based approach to modeling flexible assembly systems. The example facility assembles three types of products into housings. It is divided into six departments: a warehouse, three production lines for subassemblies, a main assembly spine, and an inventory for finished products. The production lines prepare panels, braces, and facings for product housings. The main spine includes twelve assembly and finishing operations needed to produce finished products. The three products require different types and numbers of subassemblies.

Scheduling for the final products is established by a daily production list, as will be explained later. Scheduling for subassemblies, and ultimately orders for raw materials, established based on just-in-time production policies. Subassemblies of different types will be pulled from various points in a production line, and requests can be enchained backwards as required to maintain inventory status. Requests are controlled by restrictions on queue levels and on the number of requests outstanding at any point.

Figure 1 shows the system layout for the sample facility. Parts flows are shown by solid arrows. Request flows are shown by dashed arrows. The details of its operation are explained in the sections that follow.

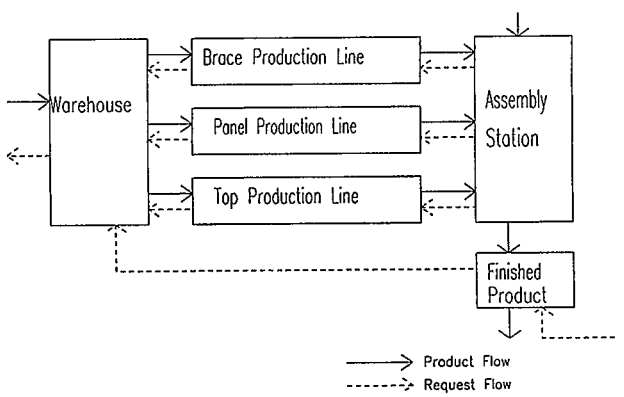

Figure 1. Assembly Plant Layout

3. Basic Information Structure

3.1. Representing the Overall System in IBIS

To represent the example system, IBIS maintains a list of work stations, a list of operations to be performed, and a list of entities involved in each operation. There is a many-to-many relationship between stations and operations, since each operation may be potentially performed at many stations, and each station may perform many operations. Similarly, there is a many-to-many relationship between operations and the entities that participate in those operations, so that information relationships between stations, entities, and operations are actually defined by an information network. An IBIS display showing the work stations for the sample problem and the basic data elements for the assembly station is given in Figure 2.

StaName	Class	Parent
AssySta	-	Plant
BraceSta	-	Plant
PanelSta	-	Plant
Plant	-	-
ProdDelivery	-	Plant
TopSta	-	Plant
WAREHOUSE	-	Plant

File: Entity

EntName	Class	Type	Fixed
FacePlate	MATERIAL	Temporary	No
Fastener	MATERIAL	Temporary	No
MATERIAL	-	Temporary	No
PanelEdging	MATERIAL	Temporary	No
Prod1	PRODUCT	Temporary	No
Prod1Brace	Brace	Temporary	No
Prod1Panel	Panel	Temporary	No
Prod1Top	Top	Temporary	No
Prod2	PRODUCT	Temporary	No
Prod2Brace	Brace	Temporary	No
Prod2Panel	Panel	Temporary	No
Prod2Top	Top	Temporary	No
Prod3	PRODUCT	Temporary	No
Prod3Brace	Brace	Temporary	No
Prod3Panel	Panel	Temporary	No
Prod3Top	Top	Temporary	No
PRODUCT	-	Temporary	No
worker	-	Permanent	Yes

File: InitialLocation

Entity	Station	Number	Status
Prod1Brace	AssySta	24	Active
Prod1Panel	AssySta	24	Active
Prod1Top	AssySta	6	Active
worker	AssySta	4	Passive

File: Operation

OprName
AttachFacePlate
AttachPanelEdging
AttachPanels
AttachTop
Finish1
Finish2
FinishPrep
InsertFasteners

Figure 2. Basic Data

As can be seen from Figure 2, the Plant is the parent for each department in the facility. For all stations, the Class is left undefined since each station has a distinct production sequence. Entities, however, are grouped into classes with similar production requirements. IBIS will operate both on individual entity types and on classes of entities, and class relationships may be extended to any level. The Entity file also distinguishes temporary from permanent entities, as indicated by the entity Type. The field indicates specifies whether or not an entities is fixed to a specific station. For example, several drill presses are used to prepare subassemblies at different production lines with each drill press fixed to a specific line.

The number and location of entities in the system is defined in an InitialLocation file. The Status value Passive for permanent entities indicates that they are awaiting requests for their services when the simulation begins.

Finally, the Operations file identifies the operations that occur in the facility.

In the example facility, the flow of entities is quite simple, since there is only one sequence of stations visited by each product or subassembly type and only one sequence of operations performed in each station. These are defined by a routing list showing the flow of entities from station to station, and a processing list showing the sequence of operations at each station. While "push" routings are simple, however, production requests are fairly complex and will be discussed in detail.

4. Request Mechanism

Production flow is controlled in two ways: by product assembly requirements and by inventory control policies associated with system queues.

4.1. Product Assembly Requirements

The requirements for each IBIS operation are defined by an KeyComponent file that identifies the products that initiate an operation and control its timing, and by an OprComponent file that identifies entities required before an operation can begin. The details of OprComponent entries specify where the required components for an operation are stored, the sequence in which they are requested, and the priority for each request, along with the number of entities required as inputs to the operation and the number of entities released (or generated) as outputs. An assembly operation may require several types of operation components, including both permanent resources and subassemblies that will be consumed during the assembly process. Before an operation begins, a key component will request operation components either simultaneously or in sequence, and will wait to begin the operation until all required operation components are available.

As an illustration, Figure 3 shows the key component and operation component entries for the AttachPanels operation. Prod1, Prod2, and Prod3 will all initiate the operation AttachPanels, but with different assembly requirements and different assembly times. Since panels and braces are joined to a product as subassemblies, and therefore consumed during operation, the number output is zero. Panels and braces are requested simultaneously, based on their ReqSequence number. A worker, however, will be requested with a higher request sequence value, guaranteeing that a worker will not be allocated to the operation until all of the required subassemblies are available.

```
File: KeyComponent
Operation      Entity     ProcTime
=============================================

AttachPanels   Prod1      NORMAL(.0350,.0035)
AttachPanels   Prod2      NORMAL(.0330,.0033)
AttachPanels   Prod3      NORMAL(.0333,.0033)

File: OprComponent
Operation      ReqSequence  KeyComponent
=============================================

AttachPanels   0            Product
Entity:        Brace
NumInput:      2
SourceQue:     A1Prod2BraceQue
NumOutput:     0

AttachPanels   0            Prod1
Entity:        Prod1Panel
NumInput:      4
SourceQue:     Prod1PanelQue
NumOutput:     0

AttachPanels   0            Prod2
Entity:        Prod2Panel
NumInput:      2
SourceQue:     Prod2PanelQue
NumOutput:     0

AttachPanels   0            Prod3
Entity:        Prod3Panel
NumInput:      2
SourceQue:     Prod3PanelQue
NumOutput:     0

AttachPanels   1            Product
Entity:        worker
```

Figure 3. Assembly Requirements

4.2. Queue Specifications

Whereas the OprComponent file defines a bill-of-materials, the Queue file specifies the characteristics of storage locations and inventory control policies in the Plant.

Queues can serve two purposes in IBIS. They serve as intermediate storage locations for work-in-process and as inventory points for parts awaiting production requests. The distinction is that between "push" production scheduling in which WIP is pushed from operation to operation, and "pull" scheduling in which parts remain in inventory until requested.. The ENDProd1Que as shown in Figure 4, for example, serves as an inventory point for finished Prod1 units from which units will be pulled to meet customer demand. Before being stored in the ENDProd1Que, each Prod1 moves through several intermediate queues in the Assembly Department that serve as WIP storage points as products are process through a series of assembly operations.

As Figure 4 shows, the Queue file specified the queue reorder point. When needed, it also holds the queue review cycle for inventory management. For example, the ENDProd1Que has a reorder point of 2, and is assumed to be under continuous review. The Reorder file holds detailed specifications for parts reorders.

```
QueName        Station            Discipline
==========================================
ENDProd1Que    ProdDelivery       FIFO
ReorderPt:     2

Entity         SourceStation      SourceQue
==========================================
Prod1          WAREHOUSE          Prod1Que
Number:        1
DestStation:   ProdDelivery
DestQue:       ENDProd1Que
Condition:     if: IVAL("REORDER:*;-NR") < 2
```

Figure 4. Queue and Reorder Specifications

Specifying reorder points, reorder quantities, and review cycles will define many inventory management policies. They are not sufficient, however, to defining kanban levels for JIT production. A kanban represents an authorization to produce a part or batch of parts. WIP is maintained so that the number of parts in process for each part type plus the number of parts in inventory is kept equal to the maximum kanban level for that part type. This restriction on WIP can be modeled using the queue reorder point and the value "-NR," which is used to indicate the number of reorders outstanding at any point in time. By setting the reorder point for a queue equal to the kanban level, and allowing a reorder to be issued any time the value of -NR is less than the reorder point, IBIS duplicates JIT controls using simple queuing specifications.

As an example the EndProd1Que has a reorder point of two. In other words, when the number of Prod1 units drops below two, an order to release one Prod1 unit for production is sent to the Prod1Que in the Warehouse. As a result, there will never be more than two Prod1 units in process or in finished products inventory. Using JIT terminology, there are two kanbans associated with Prod1.

Another adaptation of its queuing specifications allows IBIS to model the releases of parts waiting for requests to be filled. By specifying a queuing discipline as "POOL," entities that enter a queue will be removed from it once all of their pending requests have been satisfied, regardless of the order in which they have entered the queue. This allows several parts with different assembly requirements to reside in the same queue. Multiple requests can be pending simultaneously and parts will not be blocked even if parts ahead of them are starved for subassemblies.

5. Controlling Production Scheduling

Controlling product scheduling involves several issues. One has to do with setting working kanban levels and establishing other inventory policies to insure that subassemblies are available. A second has to do with establishing the sequence of job releases for product units. A third issue involves the introduction of new products, which will place new demands on the production schedule. All of these issues can be modeled through the IBIS information structures.

IBIS includes a set of output variables that allows a user to trace inventory status and product levels through the entire production sequence. These include the queue status values **-NQ** used to monitor the number of items in queue; **-TNQ** used to monitor item time in queue; and **-PCT0** used to monitor the percentage of time a queue is completely empty. -PCT0 is valuable in tracking subassembly levels to detect points at which an assembly process may be starved for components.

IBIS records similar statistics to monitor reorder status. These include **-NR** to indicate the number of reorders outstanding at any one time; **-TNS** to indicate the time a reorder remains in the system before delivery is completed; and **-TMBR** to hold the time between reorders. Reorder statistics allow a user to trace requests backward through the production sequence in much the way that queue statistics show parts flow forward through the system.

Finally, the values **-TNS** and **-UTL** record entity time in system and utilization. Because of its internal structure, IBIS can record utilization for both permanent and termporary entities. Utilization for temporary entities shows the percentage of time an entity is actually being operated on rather than remaining idle as WIP or in inventory.

As Figure 5 shows, in one 8 hour sample run, Prod1 units remain in the system 0.43 hours and are actually in process approximately 40% of that time. The inventory queue for finished Prod1 units is completely empty 8% of the time. Requests for Prod1 wait 0.04 hours on average before they are filled.

In experiments with the example system, utilization for product units and consequently utilization for worker resources is extremely sensitive to kanban levels and inventory control policies. Worker allocation levels, inventory reorder policies, and kanban levels, however, can all be controlled by changing entries in the simulation database. Lowering the reorder point for finished product queues, for example, has the effect of removing a kanban from the system. Control over production sequence can also interact more indirectly with the database through programming code that calls IBIS database routines. As currently formulated, the simulation does not schedule product releases strictly in the order they arrive. Instead, when more than one customer order is pending, a separate scheduling module sorts requests in an order that will level-out production requests for different subassembly types. That is, the kanban levels control the amount of work-in-process; the scheduling module controls the order in which product units are released for production. To establish subassembly requirements, the scheduling module directly reads the bill-of-materials from the OprComponent file.

Finally, changing production demands by adding new products or changing the bill of materials involves simple changes to the database. A new product is defined by creating a Entity entry of the class Product, defining its bill-of-materials, and the production sequence for each subassembly. The simulation automatically adjusts to modifications in the database and automatically understands the processing requirements for entities in a known entity class.

6. IBIS Implementation and Object-Oriented Constructs in the IBIS Database

IBIS has been implemented in C. Certain features of object-oriented programming, however, have been particularly important in developing the IBIS database. These include the ability to add data fields as attributes to most IBIS records, the definition of class relationships and inheritance of processing

```
Experiment    Run      StopTime
==========================
PLANT         14       8.0000

Record: Entity:EntName=Prod1
Attribute:      -TNS
CurrValue:      0.8744
Average:        0.4302
StdDev:         0.2510
Maximum:        1.1392
Minimum:        0.0010
Count:          47

Record: Entity:EntName=Prod1
Attribute:      -UTL
CurrValue:      1.0000
Average:        0.3689
StdDev:         0.6747
Maximum:        2.0000

Record: Entity:EntName=worker
Attribute:      -UTL
CurrValue:      4.0000
Average:        0.7924
StdDev:         1.4439
Maximum:        4.0000

Record: Queue:QueName=ENDProd1Que,
Station=ProdDelivery
Attribute:      -NQ
Average:        0.4409
StdDev:         0.7403
Maximum:        2.0000

Record: Queue:QueName=ENDProd1Que,
Station=ProdDelivery
Attribute:      -PCT0
Average:        0.0826
StdDev:         0.2753
Maximum:        1.0000

Record: Reorder:Entity=Prod1,
SourceStation=ProdDelivery
Attribute:      -NR
Average:        0.2459
StdDev:         0.5855
Maximum:        3.0000

Record: Reorder:Entity=Prod1,
SourceStation=ProdDelivery
Attribute:      -TNS
Average:        0.0419
StdDev:         0.0705
Maximum:        0.2604
Count:          47
```

Figure 5. Sample Simulation Results

specifications through class hierarchies, the ability to write C-code procedures within a data field, and the ability to pass IBIS records as "messages" to simulation procedures. The statistics variables -NQ, -NR, -UTL, etc., are in fact attributes that are optionally appended to queue, reorder, or entity records as special-purpose data fields. IBIS allows users to expand the attribute list for almost all database entries and will monitor the values of user-defined attributes for statistics collection.

IBIS allows the simulation to access and manipulate database entries of any type as distinct objects. The IBIS request mechanism, for example, is part of the underlying object-oriented structure of the IBIS software. Each request is maintained as an entry in the IBIS database that can be created, accessed, and manipulated through IBIS data management commands. As data fields, each request holds a pointer to a record specifying the entity being requested, a pointer to the source of the requested entity, a pointer to the destination for the requested entity, and a pointer to a "control" record, which is either an OprComponent or Reorder entry that holds specifications for the current request, along with a value indicating the number requested from the source, and a value indicating the number being expected by the destination. Because requests are structures maintained independently of the requested entity, a request can be pending while the entity is moved through any number of intermediate processing step. This allows for extremely flexible system control policies to be modeled by the simulation and represented simply by specifications entered in the IBIS database.

7. Conclusion

In conclusion, an information-based methodology provides one strategy for representing the complex system control issues raised by multi-product just-in-time assembly systems. The appropriate information model will define production specifications. Implementing this information model using object-oriented constructs allows for a flexible request mechanism that can be modeled independently of the principal forward flow for products and subassemblies.

BIOGRAPHY

Michael G. Ketcham
Department of Industrial Engineering &
Operations Research
University of Massachusetts, Amherst
Amherst, Massachusetts 01003
(413) 545-2851

Michael Ketcham's research has concentrated on advanced simulation techniques for manufacturing and incorporation of simulation into integrated decision support environments. His recent research includes the development of techniques for information-based simulation, a study of relationships between simulation modeling and database design, the development of parametric models for simulating electronic assembly systems, and application of artificial intelligence to simulation modeling.