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

WATMIMS JIT/KANBAN BENCHMARK SUMMARY AND RECOMMENDATIONS

Kenneth N. McKay
Michael Rooks
WATMIMS Research Group - Department of Management Sciences
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

ABSTRACT

Just-In-Time/KANBAN manufacturing concepts result in models that must simulate pre-process inventories at each stage, delayed processing until downstream operations indicate work should be performed, and comprehensive analysis on production orders (make/move Kanbans) versus inventory and production activities. This style of manufacture is different from the normal push environment where machines will work on anything in the work queue, send the finished parts to the next stage, and keep working until the pre-process inventory queue is exhausted; the material arriving is itself an order to make parts. The two styles imply different control logic and statistics. Many packages support push production directly, and easily provide the necessary controls and information, but there are no packages known to the authors that directly support JIT-pull. It is the purpose of this paper to report on a JIT-pull benchmark comparison of eight simulation tools and the resulting methodological recommendations.

1. INTRODUCTION

The pull style of Just-In-Time manufacturing with Kanbans is gaining in popularity among companies that have the appropriate profile of demand, supply, process, and organization to support the JIT concepts and even amongst companies where it is not necessarily a good fit. In many cases where the textbook situation does not exist, it is important to simulate or analyse the dynamics of the manufacturing environment to ensure that the concepts are feasible and to note what attributes of the system should be monitored and tuned.

During the past few years, the WATMIMS Research Group has been involved with many JIT simulations and factory analyses (e.g., Moore et al (1986), Moore and McKay (1986)) and the development of special tools or platforms for JIT analysis (McKay et al (1989)). In each case, the basic JIT concepts had to be implemented from scratch as none of the simulation tools in our research laboratory had either low-level or high-level primitives that directly matched the type of control and/or analysis needed in a JIT environment. As a result of these experiences, the authors decided that an in-depth benchmark and comparison between a number of the tools was warranted, resulting in general recommendations regarding methodology. Methodology issues centered on the ability to model the JIT process, as well as accurately representing the problem definition. The latter is a concern with higher level tools or built-in functions. For example, the tool might provide a construct for an AGV, but the construct's use implies a number of assumptions that could result in changing the problem definition and/or ignoring part of the definition and thus endangering validity and usefulness.

The research literature regarding simulation methodology in the JIT situation is sparse. Some papers such as Schroer et al (1985), Dyck et al (1988), Sarker and Harris (1988), and Moll (1988) describe a JIT model and its use, but do not critique in detail the methodology or simulation concepts used to create the model. Thus, it is the goal of this paper and associated full benchmark report to concentrate on the simulation methodology and present the different techniques used to support JIT modelling and analysis.

2. BASIC JIT PROCESS

In a typical pull production system, a machine resource can be characterized as follows:

- order queue - first in/first out, each order identifies the delivery point and quantity
- bill of material information - for each part to assemble or fabricate, the number of subcomponents which are necessary - type and quantity
- pre-process stores - for each sub-component or raw material item, there exists one or more bins - each containing a certain number of parts and an identifier of order location for bin replenishment

- production logic - when an order arrives, sufficient raw or sub-component parts are drawn from their repective bins and if possible, the desired item made - when a bin is emptied, a signal (commonly called a Kanban which can be physical or electronic) is sent to the appropriate location for more parts

The above is a simple view of one possible structure; there are many variations and extensions which can exist.

3. PROBLEM DEFINITION

The complete problem definition is fourteen pages in length and is briefly summarized in this section.

The production area is composed of a final shipping area with certain stocking levels of six final products stored in a two-bin Kanban system. As one bin empties, an order (via electronic means) is placed at the appropriate final output machine to start making another bin. Each production machine has two Kanban bins of each sub-component which are ordered electronically in turn from the preceding processing point. There are two final output machines, a paint line that operates in a push fashion between two pull points, and four other production machines. The first machines in the process order raw material from an input stores also using a two-bin Kanban method. The machines are constrained by a limited output area and can be blocked or starved depending on the arrival of orders, parts, and the material handling system. One of the parts has a minimum order quantity which causes the machine to make ahead and store the excess in its output area until the next request.

A total of seventeen parts in a three-level bill of material are specified - storage points, Kanban quantities, order points, initial bin quantities, etc. With the three-level bill of material, it is possible to have: the final output area pull from the final machine which pulls from a middle machine (which then implicitly pushes the part through paint on the way to the final machine) which pulls from the first machine which in turn pulls from the raw input stores.

The build ahead subcomponent and paint push area provide elements of complexity and randomness. An automated guided vehicle system is specified to provide the material handling. The major questions directed to the vendors pertained to the number of AGVs required to meet certain levels of on-time shipments.

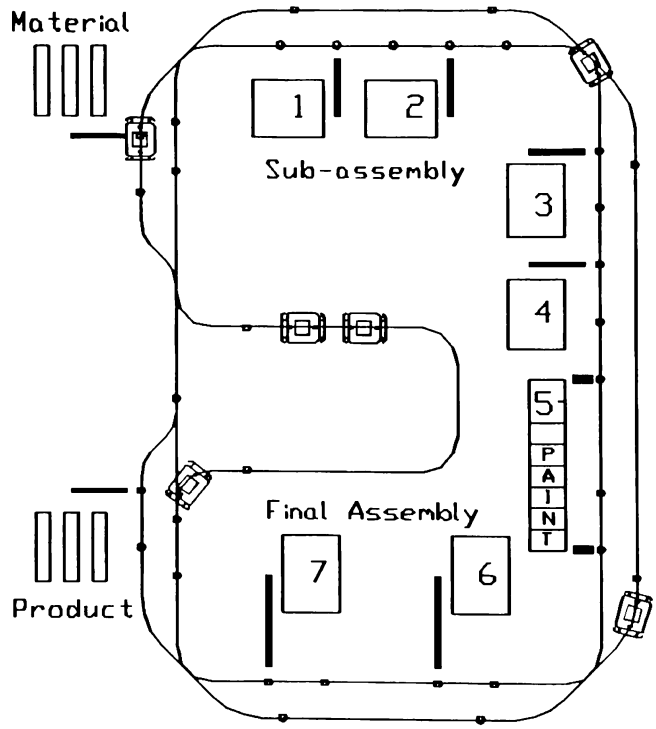

Figure 1: Factory Layout

Figure 1 shows a simplified diagram of the manufacturing layout. This final problem definition was non-trivial, yet it was not as complex as many real-world simulation situations. This simplicity was dictated by the desire to have solutions that were manageable and comparable.

4. METHODOLOGY

Three general approaches to simulation tool comparisons have historically been used. First, a survey as conducted by Law and Haider (1989) can tabulate and summarize basic features and functions. Second, the researchers conducting the comparison can create a project description and then develop solutions in each language (e.g., Armstrong and Sumner (1988)). Third, a cooperative venture can be established whereby a project description is created and then each vendor can provide a recommended solution to the problem. We adopted the third style since it is possible to have experience with a large number of simulation packages, but it is not as easy to claim expertise in all. Thus, to be fair, we allowed each vendor the opportunity to submit the best solution.

Armstrong and Sumner (1988) propose a representative project strategy utilizing templates for comparing simulation languages. They propose a five step approach: establish ground rules, select languages, select representative project, build evaluation template, and evaluate languages. This approach is similar to the way software is generally selected: establish the requirements, identify the evaluation criteria, solicit or select contenders, develop each solution, and then compare the results. These approaches sound simple and straightforward; however, expertise is required in both simulation methodology and in the problem domain in order to prepare and conduct the comparisons. Furthermore, it is very difficult to quantitatively say that one package or tool is better than another. If a tool cannot perform the task, it may be eliminated from contention - but what happens when all contenders can be made to execute the task? How does one quantitatively compare tools as different as SLAM II, PROMOD, and GENETIK?

4. 1. Comparison Strategy

Primarily, one must separate the issue of language from that of supplied solutions, by asking: "what are the basic concepts in the language used to solve the problem, independent of how accurately and completely the vendor answered the questions?" When languages are similar in general structure and philosophy (e.g., SIMAN, SLAM II, GPSS/H), a scoring system can be developed that counts the numbers and types of concepts used to solve the problem. For example, the ratio of work-arounds or creative sequences of commands can be compared between the languages and the theoretical minimum as represented by the real-world case. Thus, the cleanest and simplest implementation can be quantifiably stated. However, when the structures and systems are drastically different, numeric marking schemes and feature tables are not appropriate. We have tried to address this problem in two ways:

- This summary paper addresses the basic simulation methodology and concepts in each language. This description is relatively straightforward and is devoid of head-to-head comparisons. We provide what we believe is the simplest and most straightforward approach to the problem which is not directly implementable in any of the tools.
- The benchmark report documents our attempt at applying a relatively quantifiable metric on the solutions as supplied by the vendors. This was difficult to perform and had to take into account assumptions made by the vendors and the many different ways of addressing the problem definition.

4. 2. Vendor Participation

A preliminary problem definition and model were developed by the authors before the vendors were contacted in order for us to understand the complexity and requirements of the solution. Subsequently, the feasibility of the study was discussed with the vendors, who expressed enthusiasm and offered full support to the project. Without vendor support, it would not have been possible to obtain expert level solutions in the different packages in a realistic timeframe - it was expected that each complete solution would take approximately two man-weeks of effort.

After the complete problem description was finalized, a restricted number of vendors were contacted and invited to participate in the benchmark project. Unfortunately, only about half of the simulation tools we have in the laboratory could be included - this was not an indication of the software's capability or suitability. The vendors were provided the problem definition, access to telephone consultation to clarify issues related to the description and requirements, and a time schedule for submitting the solutions. Of the initial ten tools solicited, eight solutions were returned, representing a major investment and commitment by the vendors.

The problem solution was solicited as two major parts:

- Part A included the basic model construction, several questions pertaining to the model which would require experimentation to derive, and a number of statistics that could be used to tune the production area.
- Part B asked the modeller to identify and address a bottleneck, and in addition indicate how the model could be changed given a number of additional requirements.

The statistics and additional requirements were based on real-world JIT simulation projects involving the research group. All vendors agreed to have part of their supplied solution (i.e. program listing) published in the full benchmark report which would be distributed to all participants.

4. 3. Restrictions and Limitations

While all of the tools include concurrent graphics or post-animation options, and most include material-handling in their repertoire, the graphic capabilities and material-handling features were not directly evaluated or commented upon. The material handling analysis was limited to the AGVS interface to the scheduling (JIT) control logic. The development and execution times for the models were also not critiqued.

As mentioned in the introduction, there were two aspects of the benchmark - the language/tool characteristics and the model accuracy and completeness as generated by the vendor. This summary paper deals with the former and the full benchmark report addresses both aspects.

5. THE SIMULATION TOOLS

The simulation tools cover the range of relatively low level languages (GPSS/H, SIMAN, SLAM II w/MHEX), integrated systems (GENETIK, WITNESS, PROMOD), and specialized hybrid tools (AUTOMOD, PCMODEL).

5. 1. AUTOMOD

AutoMod (version 3.0) is a high-level language based on GPSS/H and is available on a number of platforms. It is designed to facilitate fast and accurate modelling of systems in which material- handling is a central factor. It achieves this in a unique way, using a non-procedural language and allowing detailed procedures to be included where necessary. The model consists of static entities such as resources and material-handling systems, and dynamic entities which they process, called loads. The material-handling systems are defined using an accompanying CAD package called AutoGram, which converts a CAD drawing of the systems into data files which describe the system to AutoMod. Loads are routed through resources and the material-handling systems by user-written processes which describe the details of their activity. Within these processes, detailed process procedures and AutoMod subroutines can be included.

AutoMod's modelling environment makes its representation of the JIT system unique in several ways. The AGVS system definition is done quickly and accurately in AutoGram. The user is provided with a rich set of options for its definition, and a high degree of accuracy can be achieved in the generation of AGVS logic. Once defined, however, control points must be referenced directly, making it necessary to code explicitly each movement within the AGV system.

After defining the AGVS, the user then writes processes which guide loads through the system. The coding is simplified with a macro coding feature, which allows definition of similar processes within one block of code (for example, all sub-assembly and final assembly machines are modelled within one block of code). JIT order generation is modelled in a fashion similar to other block-oriented languages. A load is created to introduce a customer order into the system, and when parts are ordered, a Clone clause is invoked, sending a new load to a process which "routes" it to the correct machine. Thus, the future events chain becomes an order queue, and all details pertaining to the order are carried with the load. After initializing several statistical counters, the load then waits for the machine to become available.

The interface between the production processes and the AGVS is quite clean, facilitated by the AutoMod constructs for placing loads on the material-handling system, and removing them. AGV orders are generated in an indirect fashion, however, since material-handling resources must be guided by loads; a load sends a clone to the AGV parking area which brings an AGV back to the ordering machine. Once this is achieved, the clone is terminated, and the parent continues on to its destination on the AGV. This indirection and the Clone usage mentioned above resulted from part of the problem definition that did not exactly match AutoMod's built-in capabilities.

When it was introduced, AutoMod provided a unique means of producing factory simulations with excellent facilities for layout definition and animation. While these characteristics remain, tradeoffs occur when logic such as JIT is considered, specialized extensive statistics are required, and

indirect programming practices result. AutoSimulation's new product, AutoMod II, may provide solutions to these drawbacks.

5. 2. GENETIK

GENETIK (version 7.10) is a PC-based modelling system enjoying popularity in Europe, and gaining in use in North America. This is the third visual modelling system its developers have been involved with and this has resulted in a very extensive platform that combines database, user interface, and modelling capabilities. These capabilities permit the modeller to perform model maintenance (e.g., subroutines, libraries), to create sophisticated user dialogs (e.g., for turnkey applications), and to completely control the modelling activity. The programming language is high-level and supports local/global variables, subroutines with parameters, procedural logic, and a number of built-in data structures. The modelling is performed with events which are scheduled and conditional events which are possible consequences of a scheduled event. If necessary, the modeller has access to the dispatch logic and event queues - a design philosophy of GENETIK is to make everything accessible.

One of the most powerful and interesting structures is TABLE. This structure is two dimensional and can have the number of rows and columns increased or decreased under model control. The columns can include any of the GENETIK primitives such as routine names which can be accessed and executed, and pointers to other tables, facilitating indirection. The table construct allows the modeller to easily build data-driven models in a clear and concise fashion - in a close approximation to object-oriented programming.

The generic pull process involves a significant amount of logic to control the machines and manage inventory. In languages supporting procedural constructs, the logical representation can be explicitly stated - conversely, the representation in a language not supporting high level procedural functions can be obscure and difficult to follow. The JIT functionality is implemented in GENETIK using a combination of procedural logic and tables. The tables provide a direct way to maintain and manipulate the bill of material, orders, Kanban status, bin contents, and routing. The program logic is separated into a number of events to make parts, start machines, and request AGV services. The events make calls to a number of subroutines that check the contents of bins, request bin replishment, and track production.

GENETIK does not directly implement queues and multiple queue manipulation which is needed to avoid an algorithmic solution to the pull problem. However, the object oriented approach to modelling, combined with the high level procedural language means that models are flexible, easy to change, and understandable. While not recommended for absolute beginners, GENETIK offers an attractive alternative to experienced modellers.

5. 3. GPSS/H

GPSS/H (version 2.0) is a block-oriented discrete-event simulation language and is available for a wide assortment of platforms. It is known for its execution speed, robustness, and its rich assortment of control statements, blocks, and compiler directives. Typically, block-oriented languages such as this are used to model a production system using little abstraction; transactions move through the simulation representing parts (or groups of parts) moving through the actual system. However, the power of block-oriented languages lies in the generality of the blocks, which allows any desirable degree of abstraction. Thus, in a JIT environment, a transaction might be used to represent a control entity at each machine, and others might represent the AGVs which interface with the machines.

Several constructs in GPSS/H make it possible to model the JIT process. Low-level blocks which set/reset logic switches can be used for implementing the numerous checks which must be made before a part can be built. The QUEUE block provides a convenient vehicle for automatically gathering a number of statistics. A drawback to using generic queuing structures (e.g., user chains or the future events chain) as opposed to a vector or array representation, is the difficulty in performing efficient search and extract operations on the queues. This issue is addressed in the recommendation section of this paper.

The SPLIT block can be used to simplify the model design. For example, all customer orders are represented as transactions. These transactions then split when an order to the next lowest level must be made; the children reside in the order queue, and eventually split as necessary, sending their children to the next level. This tree structure ensures that the model code is analogous to the real-world situation - all low level orders are generated from the final customer orders. When an order has been filled, the transaction representing it enters the AGV queue, and follows the order through the AGV code to delivery, completing the analogy.

GPSS/H is a well-designed and finely-tuned example of the original school of thought in discrete-event simulation. Although there are alternative modelling methodologies, block-oriented languages such as GPSS/H should not be considered outmoded: the overhead in performing some

tasks can be offset by the ease of others. Furthermore, none of the block-oriented languages are standing still - adding functionality in terms of modelling or in graphical analysis - GPSS/H, SIMAN, and SLAM II all have animation options.

5. 4. PCMODEL/XP+

PCMODEL/XP+ (version 9.17) is a general-purpose simulation language especially designed to make use of the graphical capabilities found on IBM PC compatibles. There are approximately fifty-four instructions in the language used for general logic, variable manipulation, moving entities on the screen, input/output, and simulation control. While the operators are relatively low-level (i.e. more assembler-like than Modula-2 or Pascal) it has a number of powerful features such as those for moving items directly on the PC screen or the operators for using vectors and arrays. For example, it has an operator that finds the lowest value in a vector and returns the index position - useful for finding zeroed slots, scheduling and dispatching algorithms, etc. Since the operator set is kept simple, it is very flexible and it can be made to do almost anything - it is just a case of programming the solutions. We believe that individuals with programming experience will find it easy to create solutions, but less experienced people will be challenged.

The JIT logic dictates keeping track of many variables (e.g., quantities of parts in bins, backorder status, etc.) which can make good use of vectors, arrays, and queues. The vector and array manipulation is simple and easy to do in PCMODEL. Per se, PCMODEL does not have queues in the sense that GPSS/H, SLAM II, and SIMAN have them. Queues are created and maintained by the user using vectors or two dimensional arrays. An empty slot is found, an item placed in it, logic is written that scans the vector, and so forth. Loops are written that go through the sub-component list for each machine, picking parts, swapping bins, indicating when a bin is full, and when parts should be ordered.

The material-handling logic is separate from the machine logic and there are not any built-in mechanisms for sending messages or entities between separate logic sections (called Routes) in PCMODEL. Two different routes can communicate via a two-step procedure. First, an entity in a route can be suspended and wait for an event to occur. Logic in a different route can cause an event to be triggered - thus waking up all suspended entities. The freshly awakened entities can then check one or more variables (mailboxes) and determine if something should be done. The mailbox technique is used to request AGV servicing, to place work orders at other machines for pulling, and for indicating when a bin of parts has arrived.

PCMODEL can be used to program almost any model. The degree of maintainability, readability, and re-usability of the programs is dictated by the care, attention, and foresight of the programmer. If good programming practice is followed (e.g., using appropriately designed data elements, naming conventions, comments, etc.), PCMODEL can do the job easily and quickly.

5. 5. PROMOD

PC based PROMOD (version 4.0) is a specialized simulation tool for quickly specifying and building models of production facilities. It is not a comprehensive language in the same sense as the other systems; there are not general variables, programming structures, etc. It is composed of a number of pre-defined structures for part routing, distributions, AGVs, conveyors, and animation.

The JIT implementation in PROMOD is perhaps one of the most cleanest and most obvious of the tools evaluated. This is possible because of a number of options in the routing table. It is possible to automatically delay parts and processing at one machine until another machine asks them to be made by using the JOIN function. In essence, there are the major routing steps from machine to machine and a minor routing abstraction that breaks down the JIT process into a number of sub-operations at each major operation. For example, for each major operation, an ORDER representing demand is JOINed with the bin quantity - resulting in a trigger to JOIN the necessary sub-components from the pre-process inventory queue. Then the bin quantity is JOINed with a bin and shipped off. This is an illustration of part of the basic control concept. A second function, UNQUE, is used in conjunction with JOIN to permit a single queue to have more than one type of part in it, yet allow similar parts to leave together - not necessarily in a first-in first-out fashion.

While it is possible to quickly construct a JIT model for a specific set of parts and part routings, it is necessary to build up the minor abstraction routing at each machine for each part being made there. Everything is explicitly named in PROMOD - for example, it is not possible to have the model data driven using parameters - ie. have multiple parts utilize the same abstraction logic, maintain their identity, obtain unique characteristics from arrays, etc.

PROMOD might appear quite simple and limited on the surface because it is a table-oriented, high-level system and is not a procedural or block-oriented language, but it has a large number of concepts that can be utilized together to achieve relatively simple solutions to non-trivial problems. This requires creativity on the part of the modeller, and

requires thorough documentation within the model to ensure maintainability.

5. 6. SIMAN

SIMAN (version 3.5.1) is a simple and easy to learn block-oriented simulation language capable of discrete/continuous modelling which runs on a wide variety of platforms. Models in SIMAN are composed of two separate sections that assist and guide the modeller in separating executable model statements and data elements. This separation is very useful for keeping the run parameters together for experiments without re-compiling the model or modifying the model code.

SIMAN has many general purpose modelling constructs to control the flow of entities through the simulation model (e.g., semaphores SIGNAL and WAIT) and a number of special-purpose functions specific to manufacturing. For example, it has special conveyor and transporter constructs. Subroutines with passed arguments are not directly supported; however, SIMAN has a STATION structure that permits one set of executable model code to be used to represent similar processes.

In a fashion similar to the other block-oriented languages, it is possible to clone orders to control the JIT work demand flow. Bringing the subcomponent entities together is more difficult. If the bill of material uses only one of each subcomponent, the MATCH block can be easily used to control the multiple queues. However, when the bill calls for multiple items, the JIT pull logic must be manually programmed using counter loops, queue processing loops, etc. The JIT problem is very data intensive and SIMAN does not have general-purpose array structures, though it is possible to use the TABLE and PARAMETER constructs to represent data stores - bill of materials, flexible routings, etc.

The AGV logic can be modelled in SIMAN using a number of different techniques. All of the techniques require the modeller to build an abstraction of the real system using the primary SIMAN functions which may require imagination and creativity on the part of an absolute beginner.

A problem common to the block-oriented languages is complex logic representation: often the modeller must use FORTRAN subroutines for the procedural logic. While it is possible with SIMAN to construct logic required by JIT using the simple built-in primitives (e.g., ASSIGN and BRANCH), the lack of sophisticated procedural constructs can result in unwieldy code when more complicated logic must be represented. SIMAN has become a very popular language for discrete-event simulation and can be expected to improve as it matures.

5. 7. SLAM II w/MHEX

The SLAM II simulation language was introduced in 1981, and allows discrete/continuous modelling in both process and event orientations. In its process orientation, the language uses a network of "nodes" (blocks) to represent the processes which entities undergo. "Activities" are used to connect nodes, and hence define the paths of entities through the network. They control multiple branching through the use of probabilistic and conditional statements, providing a flexible means of coding complex logical relationships. SLAM II is available on many platforms, and has been available for the PC since 1984. The MHex (Material Handling) extension available on many workstation/mainframe machines, provides constructs for modelling several types of material- handling systems, one of which is AGVS.

Modelling concepts for JIT systems used in SLAM II (version 4.03) are similar to those found in other block-oriented languages. JIT pull logic is accomplished using the entity-cloning features of multiple-branch activities. All production system information is maintained in the table ARRAY(i,j), including static information such as bill of materials, and the dynamic counters for inventories, bins, and order lists. If a particular product is depleted and a new order is required, an activity sends one entity to fill the product order, and another one to produce a bin of product at the specified machine. This "order entity" increments the ARRAY element corresponding to the order list for the machine, and simply awaits availability of the machine to which its processing is assigned. The activity construct thus combines conditional logic with an entity-cloning mechanism to provide a natural construct for modelling the tree structure of the JIT logic.

Statistics are provided through automatic reports and/or user-specified tables. Each activity can have a number associated with it, allowing statistics on entity counts and utilization to be maintained. Similar reports on resources are provided. The AGV system defined using MHex constructs also provides automatic reports on AGV utilization, and segment/control point entries and utilization. User-specified statistics are generated using the COLCT node, which can collect one of five types of statistics. All statistics-gathering functions provide mean and standard deviation (where applicable), and the COLCT node can optionally provide histograms.

In summary, SLAM II supplemented with the MHex extension is a mature, flexible base for simulation and like all of the block-oriented languages deals effectively with the JIT problem through the use of future event chains and/or data structures. MHex supplies a number of functions which might require FORTRAN routines to capture the variety found in the real world, and this implies the need for an experienced modeller.

5. 8. WITNESS

WITNESS (version 5.0) is a specialized simulation platform that runs on several machines. WITNESS is a high-level application using SEE-WHY, which is an extensive simulation programming language. The package is visually interactive and the model is developed using menus. The menus are very flexible and it is possible to build logic components such as IF-THEN-ELSE. WITNESS supports a wide variety of functions and data structures that can be used together to model almost any manufacturing situation.

The WITNESS system has built-in constructs for parts, tracks, vehicles, and machines. A PULL rule exists that blocks an upstream machine until the downstream machine indicates production is required. A machine can wait for a number of PULLs and similarly, a machine can PULL a number of parts at the same time. A major routing takes the bins (empty and full) between the real machines being modelled and when the bin arrives at a machine, a minor routing takes over. The minor routing emulates the splitting of bins, counting of parts, and ordering of replacement bins by using a number of "internal" machines.

The approach to modelling JIT with WITNESS is similar to other high-level packages: a number of phantom machines and parts can be used to mimic the real inventory control and decompose the problem. The degree of indirection in the modelling activity will depend on the complexity of the situation, and the creativity and problem solving skill of the modeller. Although the programming interface is menu driven and interactive, the tool retains power and flexibility.

6. RECOMMENDATIONS

Figure 2. illustrates one possible interpretation of the problem definition, and indicates the decision logic involved in one of the machines in the model. It is neither too complex nor trivial. It is a good starting point for discussing the methodological requirements of simulating JIT processes.

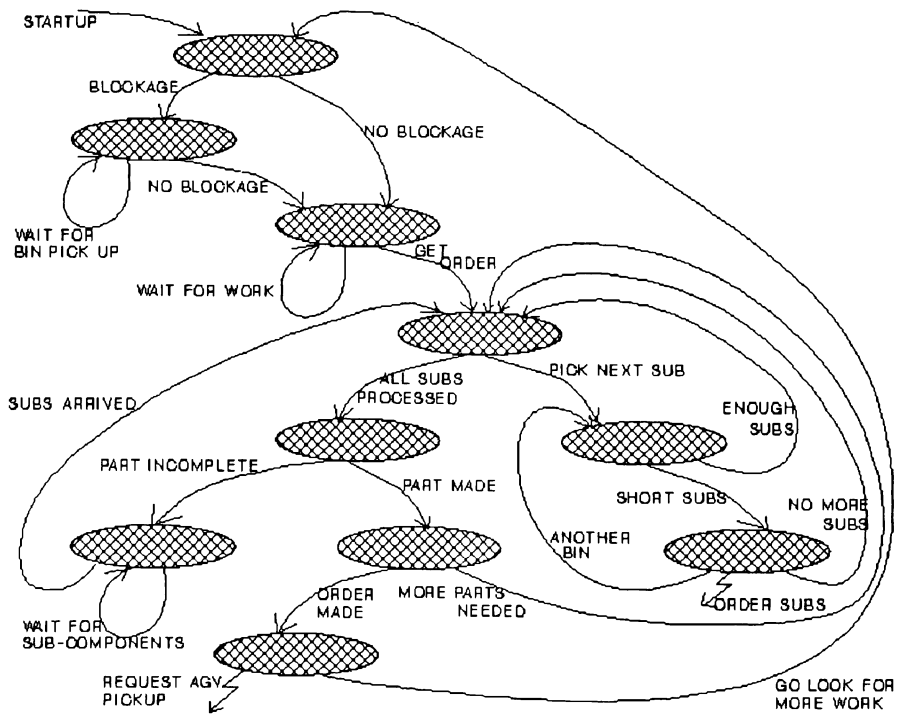

Figure 2: Finite State Description Of A Possible Interpretation

This model implies the need for a number of modelling primitives or concepts - some quite common and others less so. With almost any simulation language, one can assume that there are ways to wait for things to happen or to 'wake up' periodically to review the situation. Similarily, one can assume that meta structures exist for queues or that queues can be easily modelled with vectors, arrays, or lists and that it is relatively straight forward to determine the contents of the queue. A third common capability relates to local values per entity being modelled and global variables that can be used for gathering statistics. While the eight tools provide different solutions and concepts for each of these assumptions, none are so convoluted that the specific implementation is cause for concern.

The problems arise in dealing with multiple queues (or vectors) simultaneously and controlling part-pulling and consumption. In very trivial problems of one or two machines, the overhead and annoyances are minor. However, when many machines or resources must be modelled with many parts being produced, the inconveniences are significant. Similarly, maintaining the required statistics on orders, part consumption, Kanban usage patterns, machine starvation, etc. is not pleasant. For example, it was common for complete solutions to the problem definition to be approximately 1200 to 1600 program statements in length (declarations, executable model including statistical constructs, and report generation).

Based on our experiences, there are a number of methodological enhancements that could be added to simulation tools to make this type of manufacturing modelling more concise. The three major extensions are:

- the ability to trigger events based on inventory levels
- the ability to manipulate inventory levels based on bill of material and part production
- the ability to report on the relationships between resources, inventory, and the material handling system

These three specific manufacturing modelling extensions can be described via the following abstract modelling constructs:

1) the ability to have logic triggered when a queue reaches a level
   - monitoring parameters:
     - identifier of queue to be monitored
     - label/routine to activate when level reached
     - one or more levels to be monitored for
     - direction of approach - positive or negative
2) the ability to remove a number of items from a number of queues in a synchronized fashion
   - queue extraction parameters:
     - the total order quantity used to factor the extract request
     - the queue identifiers
     - quantity per queue (minimum, maximum) to be multiplied by the order quantity
   - control options:
     - only pull items with a matching attribute value and leave others
     - abort and reset the queues to the state prior to the synchronized pull
     - retry the pull - completely or just unsatisfied requests
     - wait and to retry the pull only after queues have been incremented
     - check the pull before actual attempt and see if sufficient items exist
   - names of different routines to activate:
     - (optional per queue) if the queue has insufficient items in it - this call would occur as a queue is processed with optional return actions: to swap to a different queue and continue, continue with this queue and assume it is all fixed up, continue with the pull and get all of the ones we can, or to abort the pull immediately
     - (optional per queue) if the queue to swap to also does not have sufficient parts
     - if at the end of the synchronized pull, there are unsatisfied extracts
     - if the synchronized extract was successful
3) the ability to provide statistics on the way the synchronized extracts were utilized - aggregate totals and time-based values
   - per queue set:
     - number of times the different type of actions were taken and the time between actions
     - number of times extract was performed and the time between extracts
   - per queue:
     - time between queue arrivals
     - number of times the different type of actions were taken and the time between actions

7. CONCLUSION

This benchmark project has attempted to illustrate how the JIT type of manufacturing simulation problem can be

solved using a number of simulation tools. Based on the verbal reports we obtained, each vendor invested approximately the same level of resources to obtain a solution. Some of the solutions are easier to enhance and maintain than others, and some of the supplied code was rather indirect - however, all of the eight tools were able to model the problem. It is believed that these issues could be addressed by consistent software engineering practices - documentation, naming conventions, etc.

When we started this benchmark, we suspected that there would not be a single clear winner - each tool has strengths and weaknesses depending on the context of who is doing the work and the type of work required. Each of the tools we included has been available for at least two years and is well-established. We hope the short descriptions of each approach and language will help other simulation practitioners in choosing appropriate software for modelling JIT problems. We also hope that the various vendors have learned from the exercise and can improve the situation at the source by enhancing their packages or providing tutorials that address this type of problem.

Three major methodological recommendations have been made regarding the modelling of JIT concepts. We believe that the three modelling constructs would greatly simplify all of the solutions and clarify the code. While the constructs have evolved from our JIT modelling experience, the multiple queue capabilities would benefit many other modelling situations.

This summary has been necessarily subjective and interpretive by nature. Distribution of the quantitative analysis and full benchmark results has been restricted to the vendors.

ACKNOWLEDGEMENTS

All of the companies that participated in our benchmark must be gratefully acknowledged for their unselfish and enthusiastic support of the study - AutoSimulations, Inc. - 801-298-1398 (AUTOMOD), Insight International Limited - 416-896-0515 (GENETIK), ISTEL Incorporated - 216-292-2668 (WITNESS), Pritsker Corporation - 317-463-5557 (SLAM II w/MHEX), Production Modeling Corporation - 801- 226-6036 (PROMOD), Simulation Software Systems Inc.- 408-436-8300 (PCMODEL), Systems Modeling Corp. - 412-741-3727 (SIMAN), and Wolverine Software Corporation - 703-750-3910 (GPSS/H).

REFERENCES

Armstrong F.B. and S. Sumner (1988). The project approach to simulation language comparison. *Proceedings of the 1988 Winter Simulation Conference* (M.A. Abrams, P.L. Haigh, and J.C. Comfort eds.). IEEE, San Diego, California, 636-645.

Dyck H., Johnson R.A., and J. Varzandeh (1988). Transforming a traditional manufacturing system into a JUST-IN-TIME system with KANBAN. *Proceedings of the 1988 Winter Simulation Conference* (M.A. Abrams, P.L. Haigh, and J.C. Comfort eds.). IEEE, San Diego, California, 616-623.

Law A.M. and S.W. Haider (1989). Selecting Simulation Software for Manufacturing Applications: Practical Guidelines & Software Survey. *Industrial Engineering*, **31-5**, 33-46.

McKay K.N., Buzacott J.A., and D.A. Phillips (1989). Cell-Sim - A Simulation Tool For Hybrid Environments. (to appear), *Third ORSA/TIMS Special Interest Conference On Flexible Manufacturing Systems: Operations Research Models And Applications*, Cambridge, Massassuchetts.

Moll W.H. (1988). JIT, truck docks, and simulation. *Proceedings of the 1988 Winter Simulation Conference* (M.A. Abrams, P.L. Haigh, and J.C. Comfort eds.). IEEE, San Diego, California, 719-721.

Moore J.B. and K.N. McKay (1986). Experience with MAP/1 in a Manufacturing Simulation, *Proceedings of CCICS'86*, Montreal, Canada, 61.1-61.5.

Moore J.B., McKay K.N., Kostelski D., and J.A. Buzacott (1986). Using Simulation to Plan Storage Space Needs With Just-In-Time Manufacturing. *Proceedings of SIM-2*, Chicago, Illinois, 95-106.

Sarker B.R. and R.D. Harris (1988). The effect of imbalance in a just-in-time production system: A Simulation Study. *International Journal of Production Research*, **26-1**, 1-18

Schroer B.J., Black J.T. and S.X. Zhang (1985). Just-In-Time (JIT), with Kanban, manufacturing system simulation on a microcomputer. *Simulation*, **45-2**, 62-70

AUTHORS' BIOGRAPHIES

Kenneth N. McKay is Associate Director of the WATMIMS Research Group and is a Doctoral candidate in the Department of Management Sciences at the University of Waterloo. He received a MASc in Management Sciences in 1987 and a BMATH in 1978 from the University of Water-

loo. His research interests include the Job Shop Scheduling task in the real world, JIT manufacturing methods, and a number of research activities relating to simulation methodology - design flexibility, project management, and software engineering. Before joining the research group, he was a software specialist in industry for eight years, working in software design evaluation and system architecture.

Kenneth N. McKay
Department of Management Sciences
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1
519-888-4519

Michael Rooks is affiliated with the WATMIMS Research Group and is pursuing graduate studies in Natural Systems Simulation. He has a B.Eng (Manufacturing) from McMaster University, and an extensive background in simulation development and analysis in the automotive industry. His research interests include simulation methodology in combined (continuous and discrete) applications, scenario generation and simulated system optimization, macro level modelling, and JIT manufacturing concepts in simulation languages.

Michael Rooks
Department of Systems Design
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1