A FLEXIBLE MANUFACTURING SYSTEM DESIGN AND EVALUATION PACKAGE

C. González Fernández-Vallejo
M. L. Reynoso Pineda
P. B. García de Castro
N. Julián Rigau

Instituto de Automática Industrial
La Poveda, Arganda del Rey
28500 MADRID SPAIN

ABSTRACT

A Flexible Manufacturing System design and evaluation package has been designed upon request of the leading spanish manufacturer of the field. The package consists on three programs, the component library manager, the plan design program and the simulator program. The library manager. contains both shape and chacteristics of the components. The plan design program permits the location of components extracted from a library, its connection through transportation elements, the drawing of a layout and the storing of its map for possible future modification. The simulator program consists on a process description and a simulation. The simulation is made through a specially designed program developed in C langua-e.

1 INTRODUCTION

A FMS is a very expensive piece of hardware whose performance is usually required by contract to be of a given amount. The penalties in case of breach of contract are terrible and a small firm making a mistake might even go broken. Because of this, a great deal of effort is invested in techniques that attempt to measure the throughput of a FMS before it is constructed and many of them are based on simulation. As a matter of fact, no other good technique exists presently other than simulation although many efforts are being done in other directions. Petri Nets offer mathematical instruments to study parallelism and timed petri nets (1) offer the promise of the same thing for the future. However, at this moment the techniques timed petri nets use are essentially similar to simulation.

1.1 Current Tendencies

The efforts developed in the design of tools for the simulation of FMS's could be clasified according to these criteria:

- The programming techniques used in the simulation process.
- The degree of integration.
- The number and complexity of the analysis tools additionally provided.

Among the programming techniques used in discrete-event simulation we find event scheduling (Simscript), activity scanning (GPSS), process interaction (Simula, GPSS, Simscript), message-based simulation (2), etc. Each of these techniques is more suitable for a specific class of problem. For example, event-scheduling is best suited for situations where the system and its actions are well defined, while process interaction is better for modeling systems with many parts that must work together (3).

Depending on the degree of integration we have specific programming languajes (SIMAN, GPSS, SLAM, MAST, MAP/1, SIMSCRIPT, etc), modular systems that generate specific programming languages code and interactive, complete environments that allow the user to model FMS's without the need of knowledge about programming or any previous relation to computers (SIMFACTORY, SIMULATIONCRAFT, (4)) Depending on the kind of tools provided by the system there are inteligent interfaces for the input of data (natural language recognition (5), assisted drawing tools, detection of inconsistencies, etc.) and tools that analize the output data of the simulation process. Among these ones there are analizers that detect malfunctions in the FMS's like bottlenecks, overloads, etc., and suggest changes for improving the performance (5,6). More advanced tools pretend to satisfy objectives set by the user in the scope of cost/performace values; the user provides alternative designs of the FMS and the tool tries to meet the goals by applying the rules stored in a knowledge-base (6).

The future tendencies in this area aim at the integration of this systems with CAD/CAM tools and with production control and planning software as well as making them more intelligent in order to obtain optimal configurations given the set of elements that constitute the FMS.

1.2 Our Approach To FMS Design Through Simulation

In this work, we describe the system we are developing for the computer aided simulation of FMS's. It is to permit an easy evaluation of the performance of the system by a human operator, and also it is to be able to perform some automatic optimation. The goal is twofold, either to evaluate the design (The manufacturer's problem) or to evaluate the manufacturing process (The user's problem). The first goal is used before the FMS is actually built: several designs are tested so that one may be selected. The second solves the problem of the planner. While the FMS is in production, the manufacturing plan can modify the overall performance of the cell in ways that cannot be foreseen, thus a mumber of plans can be tested and one of them selected.

The simulation system with which we are dealing, consists on three parts. The first one is a system that maintains a detailed library with the elements that the engineer uses to design the FMS. The degree of detail is important as far as design and budgeting are concerned; the second one is a design system that allows the user to draw a realistic layout of the FMS generating a logical representation of it for simulation purposes. Most of the simulation systems actually working lack this facility (4,5,6) which we think is very important for the engineer that designs and budgets FMS's in close relation with the customer; the third part is the simulation tool, consisting on both a manufacturing definition facility by means of which the designer can specify sequences of operations, batch sizes, control strategies etc. and the simulator which takes elements from the other systems and generates statistical information about the performance of the FMS.

2 THE ELEMENT LIBRARY

The first stage in the manufacturing of a library is to analize the elements the library is to keep. So, in the case of FMSs it must be studied the elements that participate in it in order to make abstractions of the characteristics that are important of simulation.

The objetive of this element library is to save all the graphical, numerical and alfanumerical characteristics of some of the elements that constitute an FMS (the "fixed" and "transport" elements). These characteristics are inherent to the elements, independently of the particular system in which they will be used. So when the layout has to be done the user only has to set their parameters and put them together. As this program is a modular one it is posible to modify an element without disturbing the whole system and it also permits the definition of several elements in parallel.

We can model the FMS as a device that transforms the material with which it works. This material is transformed by its succesive pass through several machines. This flow of material from one station to another is done by means of transportation elements.So it looks natural to conclude that there are, at least, four major groups involved in simulation of FMS's. We will call each one of this groups a class.

We will use, in order to define the elements that constitute a FMS the sign '{}' to denote a list that may or may not be empty and '()' to denote a n-tuple.

So, for us an element is made up as

```
ELEMENT = (    CLASS,    name,    model,{GENERAL
        CHARACTERISTIC}, {GENERAL RESTRICTION},
        {VARIABLE        ATTRIBUTE},{OPERATION},
        {BUFFER}, {FAILURE}, {DESIGN}, {I/O})
```

Where CLASS can receive any of the following values:

- "fixed" includes machines and operators.
- "transport" includes conveyors, cranes, trucks, robots, etc..
- "pallet" corresponds to the flowing entities of the system. These elements correspond to pallets that transport the parts to be processed or the parts themself.
- "consumable" correspond to the tools needed for the processing of the parts; it also include elements needed by the machines to work as it is lubricant, electric power, water, etc.. This elements are important during simulation because they impose not only phisical but economical limitations to the system.
- "Source/Sink" Correspond to the I/O elements, that create pieces in initial state or make them disappear when they are finished.
- "Consumable store" is an element which keeps consumable elements as directed in the process definition language.
- "Product store" is an element that can keep the pieces for some time while waiting.

--- GENERAL CHARACTERISTIC is of the form (name, VALUE), where
VALUE = number or string

This parameter correspond to the data that is fixed to the element and its value never changes. It is the same for all simulations.

--- GENERAL RESTRICCION is (name, minimun value, maximun value, increment)

This parameter sets limitations to the input/output of the element. It is not able to accept parts that do not have the value of their respective general restriction with the same name among these minimun and maximun values.

--- VARIABLE ATRIBUTE is also (name, minimun value, maximun value, increment)

This atributes will have different values during the simulation and will determine the state of the element. It may also be defined as a numerical or alfanumerical set. In case of being a set its definition will be :
(name, {value})

--- OPERATION = { action }

Action is the name of one of the operations an element is able to realize. For example drilling.

--- BUFFER = (name, capacity, {name of consumable})

--- FAILURE = (name, { probability density, minimum time between failures, maximun time between failures, minimun length of the failure, maximun duration of the failure })

A Failure determines the probability of failure in the interval given. The duration of the failure is selected randomly. In form of a table the user will give the failure statistics. The given values are range (time interval between failures or mantenance), probability (of a failure in that range) and duration (of the failure or mantenance).

Each failure should have an associate table.

--- DESIGN consists on a set of pictures, representing the

- "physical area" corresponds to the physical space ocupied by the element.
- "vital area" corresponds to the physical space needed by the element for its correct function.
- "access area" corresponds to the physical space that an element can access; for example a robot.
- "fancy area" corresponds to the drawings the designer may need to make his design more legible for him. This data is a help for the designer and is not needed for the simulation.

Each element may have a scaled physical drawing. When the form of an element, as could be a transport element, has a form that depends of the FMS layout its picture is drawn directly on it. These data will be saved in other library. The input/output points are drawn at this stage.

--- I/O = { POINT }

POINT = (TYPE, x-coordinate, y-coordinate)

TYPE ="input of pallets", "output of pallets", "input/output of pallets", "input of consumable", "output of consumible", "input/output of consumable"

The preceding lines determine the contents of information we need to represent an element. A program has been developed to carry out this task. The program consists on two parts, the first one to help in the making of the design and the second to help with the alphanumeric characteristics.

3 THE DESIGN PROGRAM

The second part of the project consists on a program to perform the following tasks

- Help the designer in the physical layout of the FMSs.
- Verify the design once it is made and insure against physical errors, i.e. if some machine lacks some system to feed things in or out to it.
- Make the logical map of the FMS in which abstraction is made of the physical characteristics in order to develope a data structure that might be read by the system in order to do simulation.

In order to achieve the previously stated goals, the first task the program makes is to help in the realization of the layout. Access is provided to element libraries so that the machines already defined can be located. After, the connection elements are put. Since we want the produced map to be realistic, the linear elements (conveyor belts) are only partially defined in the library, and the final data about them (width) is only defined at location time. This data, however, is only needed for map purposes. The points where two belts merge, or one forks into other two are transportation elements and should be taken exactly as defined in the library. After a belt is plotted its length is computed and stored, as well as when it is modified. This datum is important for simulation. Area transportation elements (robots) are located as they are defined in the library. Whenever an element is placed, the program prompts for a name, which is assigned to it an determines the element completely.

The verification takes into consideration that every fixed element has to have accessible every one of its input and output points, that is, given one input point of a fixed element, a conveyor belt is to end in it, or it should be located within the useful area of a robot. The output points of machines are to be either connected to input of belts, or located within the useful area of robots. Also, at least one input and one output elements are to be present, although there can be many more.

The logical structure of the FMS is a graph in which each node represents a machine and is connected to nodes representing the elements to which its input and output points are connected. Each node carries a reference to the library and element therof to which it is related. The conveyor belts are assimilated to fixed elements that do not transform the state of the pieces they process. The time of transition between its input and its output is a function of its length and the speed of the belt, a datum that can be changed dynamically at simulation time.

4 SIMULATION

After some FMS design has been succesfully entered some more steps are necessary before a simulation can be performed. First of all, the manufacturing is to be described. Later the simulator can be activated, passing to it some data on simulation.

4.1 Manufacturing Definition

A Language has been defined to perform the task definition, which mainly consists on the piece manufacturing way; Input/output element production program, general simulation program information and initial FMS setup. Each of these parts can reside in a different file, which is later read by the FMS simulator.

4.1.1 Piece Definition

The definition of a piece consists on a set of clauses, starting by one defining the piece name and, optatively, length and ending in an END Clause. In the middle there are three types of clauses: PROCESS, ASSEMBLE IN and ABSORB.

A Process clause consists on
1 A machine determination (Either generic or specific)
2 Definition of the input and output points.
2 (Optatively) Internal consumable store composition.
3 List of operations and time they take, including, optatively, the expense of consumable elements made.

The ASSEMBLE IN clause consists only in a list of machines to which the piece is to head. It must be the last clause of the piece program.

The ABSORB clause consist on
1 A machine determination (Either generic or specific)
2 Definition of the input and output points.
3 Time the operation takes, including, optatively, the expense of consumable elements made.
4 List of pieces to be absorbed, including the input point at which they enter the machine.

4.1.2 INPUT/OUTPUT Element Activity Definition

This definition consists on two parts, a first one to define batches in terms of pieces and a second one to define I/O activity in terms of batches.

A Batch definition consists on
1 Batch name definition
2 A set of clauses, each of them specifying a piece name, as it appears on a piece program and a quantity, which might be a range. In this case, at some moment, the value of the batch number of pieces is to be selected randomly.

A Source definition consists on
1 Source name specification, and generation mode specification (Can be seuquential or random).
2 Pallet name and attribute, if the FMS is to work with pallets.
3 list of Batch names and intervals between two consecutive parts generation.

A Sink definition consist on
1 Sink name and time of operation (The time it takes the sink to process the part. During this time the piece is unable to accept input).
2 List of parts names.

4.1.3 General Simulation Data

Some data pertinent to simulation is included in this part. These data include

- GENERAL CONTROL STRATEGY parameter, which informs the simulator program on the strategy use for piece addressing, when several machines can be used for the next step. Four options are considered: go to the closest machine (If possible), alternate all of them, select one of them randomly, go to the least used of all.

- LOCAL CONTROL STRATEGY. It consists on programs for some (or all) of the multiple output transportation elements. One of these programs consists on a series of clauses, each of them stating for a given piece in a given state the destination point, if some element (determined by its name) is free or occupied.

- PIECE/CONSUMABLE TRANSPORTATION parameters. They state whether the pieces (or consumable elements) are transported directly or in pallets.

- CONSUMABLE TRANSPORTATION STRATEGIE. This parameter defines whether a consumable element is substituted when its useful life ends or when it is clear is has not life enough to last through the operation.

- LINE SPEED.

4.1.4 Initial FMS setup.

This part permits to specify the number and position of the pallets at starting time, the buffer size and access type (Direct, FIFO, LIFO, Random) and its average access time.

4.2 Data For Simulation

The design of the FMS and the manufacturing definition provide data that is needed for the simulation, but not enough. More data are needed in order to carry out the simulation in the easiest of the ways. The data we are going to discuss here are the FMS internal representation, the piece programs and the I/O element and transportation element programs and the minimal trees.

4.2.1 FMS Representation

The system will be represented by means of a series of linked structures, containing, some of them links to more structures.

Each element of a FMS is represented by a structure, whose address quite represents the element. Within this structure, there are pointers to

- I/O points list elements
- Piece movements program (Only multiple output tranportation elements)
- Piece generation/consumption program (Only I/O elements)
- Minimal trees

The I/O point list consists on a linked list of elements, each containing the number of the point, its type, pointer to the element to which this point is linked and the number of the I/O point.

4.2.2 FMS Component Program Representation

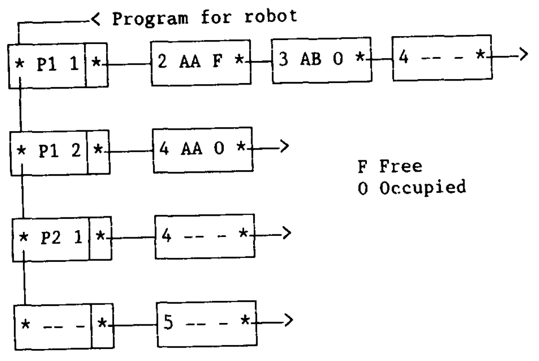
We have already mentioned the program to describe specific and special behavior of multiple output transportation elements. These piece movement programs are linked lists, each of its elements making reference to the movement of a piece. These references consist on a state and a piece name.

We represent the following program:

```
TRANSPORT name
   SENDS p1 STATE 1
         IF AA FREE VIA 2
         IF AB NOT  FREE VIA 3
         ELSE VIA 4
   SENDS P1 STATE 2
         IF AA NOT FREE VIA 4

   SENDS P2 STATE 1 VIA 4
   REST VIA 5
```

where the meaning of the keywords is obvious, in the Figure:



The I/O elements also have programs. These programs are also a lists of linked lists, themselves linked to the elements to which they make reference. It is a straightforward representation of the batch definition, where lists of piece names and batch sizes are enumerated.

4.2.3 Minimal Tree Representation

Our simulation program is to have some intelligence, and most specially the capacity to send pieces directly to their destination at any moment, and the knowlege of the minimal path to this destination. Devices contrived to this effect are the minimal trees. They are the description of the unique minimal connection between the machines performing two consecutive steps of the piece program (Realize there may be several origins and several destinations). The minimal trees are trees with multiple roots and multiple leaves, and its nodes contain information about the possible destinations to which they lead. There is a minimal tree for each state transition from one state another. Therefore as soon as a piece emerges from a machine, the minimal tree can be used to determine the possible destinations so that one may be selected. Any path other than the minimal has to be given explicitly by means of the program of a transportation element.

4.3 Simulation Program

Once the Data Structure is clear, the Simulation Program design and implementation becomes an easy thing. We decided originally to carry out the implementation in C because of portability and because it gave us a powerful structure capacity as well as dynamic memory allocation. The program is structured as a module generating calls and other modules, which are called by the first. The overall organization is conceived much in the way an operating system passes requests to the drivers serving the periferals. All the drivers receive the same data, but each reacts in a different way to it. All of them share a common input and output data structure, although some fields might be unused. The called modules thus receive the same data, and depending on the module type perform one action or another.

4.3.1 Input data structure

We have said that the driver modules, representing each one element type are called receiving each the same data structure. This structure has the following fields:

-Subject	Pointer to the data structure that represents the machine performing the action.
-Command	The action that is to be performed.
-Piece	Pointer to a mobile element that might be the piece or the pallet containing the piece.
-Input	An input point. Usually the piece input point.
-Output	An output point. Usually the piece output point.

-list	A pointer to a list of consumable products required.

The commands that may be passed to the driver modules can be:

- Initialize
- List available consumables.
- Produce consumables.
- Status.
- Listing.
- Wake-up.
- Block output point.
- Unblock output point.
- Get piece.
- Get consumables.

4.3.2 Output data structure

The modules produce all of them an output data structure in which the fields are:

-Piece	Pointer to a mobile element or pallet. Produced at the output point.
-Output	Output point.
-Block	Pointer to a list of output points blocked as result of the command.
-Ublock	Pointer to a list of output points unblocked as result of the command.
-list	Pointer to a list of consumable products produced.
-Status	Result of the command, when a status is required from an element.
-Time	Time the next call is to be done to the element.

4.3.3 The Simulator Program (SP)

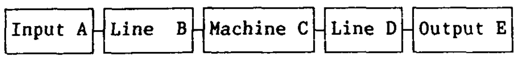
The general structure of the Simulator is easy, since most of the complexity has been shifted to the data structure. The core of the program is a module that keeps an ordered list of calls and the moments at which the calls are to be done. Initially the list is empty. Essentially, the algorithm the SP implements can be written down as follows.

1 Establish dialog with all elements passing them the command "INITIALIZE". Some of them, after looking its program, will include in their answer the time for the next call. that will be included in the list of call.

2 Get the first entry in the list and call the corresponding element. Analyze the answer. Realize that analyzing the answer might (usually will) require to enter new commands into the list.

3 Repeat step 2 until the list is empty.

As an example, let us do a short example in a table form. The cell to which it corresponds is



T	Action	Message	List (After)
0	Call A	Initialize	Empty
	A answers	Wake me at 20	(A,20)
20	Wake up A		Empty
	A answers	Put out P &	
		Wake me at 25	(A,25)
	Call B	Get Piece P	
	B answers	Block input &	
		Wake me at 28	(A,25)-
			(B,28)
	Call A	Block input	
25	Wake Up A		
	A Answers	Nothing	(B,28)
28	Wake Up B		Empty
	B answers	Unblock Input &	
		Wake me at 40	(B,40)
	Call A	Unblock input	
	A answers	Put out P &	
		Wake me at 38	(A,38)-
			(B,40)
	Call B	Get Piece P	
	B answers	...	...
..	...	...	...

5 CONCLUSIONS

The design of FMS is a very rewarding enterprise because of the possible consequences of bad results. Our approach, consisting in aiding a human designer with an integrated set of tools developed in common procedural language provides in our opinion the highest cost/performance relation.

6 REFERENCES

1. Proc. Int. Whorkshop "Timed Petri Nets". IEEE Comp. Soc. Press. 1985
2. Rajive L., K. Mani Chandy, A "Message-Based Approach to Discrete- Event Simulation". IEEE Transactions on Software Engineering, VOL SE-13, NO 6, JUNE 1987
3. Ricardo F. Garzia and Mario R. Garzia, "Discrete-Event simulation". IEEE Spectrum December 1986
4. M. Montazeri, L.F. Geldera and L. N. Van Wassenhove. "A Modular Simulator for Design, Planning and Control of Flexible Manufacturing Systems". Then International Journal of Advanced Manufacturing Technology, 3(1), 15-32, 1988
5. Donnie R. Ford and Bernal J. Schroer. " An expert manufacturing simulation system". SIMULATION. 48:5 May 1987
6. Joseph M. Mellichamp and Ahmed F.A. Wahab, "An expert system for FMS design". SIMULATION. 48:5 May 1987