Automated Generation and Evaluation of Control Programs for Discrete Manufacturing Processes

Bruce H. Krogh, Reg Willson, and Dhiraj Pathak
Department of Electrical and Computer Engineering
and the Robotics Institute
Carnegie Mellon University

ABSTRACT

This paper describes a set of software tools for specifying the control and process dynamics for discrete manufacturing systems. Specifications are entered and modified in a relational database using a formalism that naturally reflects the functional components of a discrete control system. A Petri net model of the control and process logic is extracted automatically from the database and various analysis tools are provided to investigate the consistency and viability of the specifications. The control logic is then translated into source code for the on-line control computer. The specification formalism and generated control software are illustrated using a simple die-stamping example and a real-time process simulator for program verification is described.

INTRODUCTION

The writing and debugging of computer programs for on-line control accounts for a major component of the cost to implement automated manufacturing systems. The objective of the research described in this paper is to develop software tools to automate the generation and verification of computer programs for the on-line control of discrete manufacturing processes from a high-level description of the system. With this software the manufacturing engineer can specify the control logic for each operation in the manufacturing process, analyze the control specification for completeness and logical consistency, and then generate computer programs for on-line control computers.

Figure 1 illustrates the basic components of the software and laboratory setup for generating and evaluating real-time control programs. The functional description of the manufacturing process and the process control logic are entered into a relational database on a workstation. The sequencing logic in the functional specification is analyzed for consistency and potential problems are identified for the user. The logical analysis and diagnosis software is based on results from the theory of Petri nets [1], [2], and their application to discrete manufacturing processes [3]. When the control specification has been verified, a computer program is generated for the on-line control computer. This program is ported from the workstation to the computer which controls the process. To test the control programs in the laboratory we have implemented a real-time process simulator on a personal computer. This simulator accepts actuator commands from the control computer and generates sensor signals in real-time so that the control program is executed as if the actual process were being controlled.

This paper is organized as follows. In the following section we briefly describe Petri nets and our methodology for modeling discrete manufacturing systems. We then describe the specification language for the relational database which contains the functional description of the process and sequencing control logic. In subsequent sections we describe the software for logic analysis and generation of the control program, and the real-time process simulator for testing the on-line control program. In the concluding section we discuss directions for future research and extensions of the present development software.

PETRI NET MODELS OF MANUFACTURING SYSTEMS

We consider applications such as discrete parts manufacture and automated assembly for which the overall manufacturing process can be decomposed into a set of distinct *operations*. An operation is a single stage of the process which is executed independently from the other parts of the system. Operations can be performed concurrently, but precedence relations among the operations can impose a partial ordering on the operation sequence. Each operation requires a set of *resources*, such as raw materials, fixtures, and robots. As the process evolves in time, the resources cycle through sequences of discrete states, some of which are detected by sensors. The states of the resources are used by the control computer to make operation sequencing decisions.

As an example, consider the metal stamping system illustrated in figure 2 in which a robot services a stamping machine transferring, material between the machine fixture and totes on a conveyor system. The resources in this simplified model include the stamping machine, the robot, the conveyor system, totes and the material in the totes. The control computer must coordinate the activities of the stamping machine, robot and conveyor by issuing the appropriate commands based on sensor signals indicating the states of the various resources. Discrete states of the robot might include a ready state, and two states corresponding to the two activities of moving raw material from the tote into the fixture and moving the stamped material from the fixture back to the tote. Signals from the robot controller would indicate the robot status to the system control computer.

To model analytically the discrete decision and control structure of a manufacturing system several researchers have investigated the use of Petri nets which were originally conceived to model concurrency in computer systems [4], [5], [6], [7], [8], [9]. Figure 3 illustrates the standard graphical representation of a Petri net in which *transitions* are represented by vertical bars, *places* are represented by circles, and *tokens* are represented by dots which reside in the places. The distribution of tokens in the Petri net places is called the *marking*, indicating the current state of the system. In figure 3, transition **t1** is said to be *enabled* since both of its input places (p1 and p6) contain tokens. Enabled transitions can *fire*, in which case a token is removed from each of its input places and a token is added to each of its output places. If transition **t1** in figure 3 fires, the tokens are removed from places **p1** and **p6** and a token is placed in place **p2**, the only output place for transition **t1**.

Petri nets are attractive for modeling discrete manufacturing systems because qualitative and quantitative properties of the Petri net graphs correspond to properties of interest in manufacturing systems [4], [5], [7], [8]. Petri net models for manufacturing systems normally represent the control logic and the process states in a single graph (see [3], and the references therein). Petri net models of this type are often used for simulation and performance evaluation of alternative control schemes and system configurations for a manufacturing process [6], [10], [11]. Such models are also used in so-called "token-player" control schemes where the control computer maintains a Petri net model of the system to determine control actions, using sensor inputs to update the marking of the model [12], [13], [14].

0-8186-0888-9/88/0000/0092/$01.00 © 1988 IEEE

With the traditional approach to building Petri net models of manufacturing systems, it is difficult, however, to distinguish the function of the control computer from the state transitions inherent to the process. To facilitate the use of Petri nets for control program evaluation and generation, we have developed an alternative approach to building Petri net models which separates the process model and control logic into separate graphs. Our overall Petri net framework is illustrated in figure 4 which shows that the basic components of the feedback control loop are represented by separate Petri net models. In addition to separating the process and control logic models, we also represent the sensor and actuator components separately so that alternative implementations of the system can be evaluated. Figure 4 includes simplified Petri net models for portions of the stamping process example. The arrows connecting the process, sensor, control computer and actuator blocks indicate the causal relationships among the Petri net graphs. In this extended modeling framework, transitions in one block can be conditioned on the states (markings) of places in preceding blocks. For example, transitions in sensor states are conditioned on the presence of tokens in resource state places in the process model.

The Petri net models described above provide the analytical basis for our logic verification software described below in section 4. We do not use Petri nets as the representation language for building the system model, in contrast to other researchers [9], [15]. In the following section we describe a rule-based specification language which allows the manufacturing engineer to define the elements and structure of the state-transition models without actually developing a Petri net model beforehand. The computer then performs the task of extracting a Petri net model for analysis from the rule-based specification.

A FUNCTIONAL SPECIFICATION DATABASE

In this section we describe a specification language for describing the process and control logic for manufacturing systems in terms discrete state variables representing the physical resources, sensors and actuators in the system. Specification statements are stored in a relational database, which facilitates the entry and modification of the specification and extraction of the equivalent Petri net model for analysis. From this high-level specification of the manufacturing system, the software described in the following sections extracts a Petri net model for the analysis of the control logic and generates the on-line control programs, thereby eliminating the time-consuming and error-prone process of manually encoding and debugging the control logic for the control computer.

The starting point for developing the system specification is the identification of the discrete state variables needed to characterize *process resources* and the discrete values that each variable can assume under all possible operating conditions. Following this, rules describing state transitions within the process resources are added to the database. State transition rules are defined by a set of preconditions in terms of the values of one or more state variables, and a set of postcondition values for the state variables when the transition occurs. The relational database enables the manufacturing engineer to specify the resource state transition rules as they occur to him or her, rather than requiring that the entire process specification be developed before entering it into the database.

The *actuator* and *sensor* components of the system are specified in a similar manner, with transition rule preconditions using appropriate state variables. Preconditions for rules governing sensor state changes can involve only process state variables, while those governing actuator state changes can involve only controller state variables. This reflects the causality relation between the process and the sensor states and between the control computer and the actuator states, indicated in figure 4. We note that by separating the sensor, and actuator specifications from those of the process, it is easy for the system designer to modify the structure of the sensing and actuation in the system specifications to evaluate alternative control strategies.

The final step in developing the system specification is to describe the *controller* in terms of state variables and state transition rules. The controller state variables correspond to the internal state information required to implement the desired sequencing logic, with the state transition rules depending on the values of the controller and sensor state variables. In contrast to the state variables in the process, sensor and actuator specifications, the controller state variables are entirely defined by the engineer, and do not necessarily correspond to physical entities or their attributes.

The user enters the system specifications into a relational database through a file editing procedure. Internally the relational database is defined by *fields*, the basic data units of the database; and *relations*, which are tables containing associated fields. For our application there are five fields: *variable_name*, *variable_state variable_type*, *variable_description*, and *rule_label*. Each *variable_name* identifies one of the state variables in the system which makes transitions among its associated *variable_state*'s according to the rules identified by the *rule_label*'s. The four *variable_types* are ACTUATOR, PROCESS, SENSOR, and CONTROLLER, corresponding to the basic decomposition of the manufacturing system. The four relations (and the associated fields) defined in the specification database are:

- *system_variables* (variable_name, variable_state, variable_type);
- *variable_states* (variable_name, variable_state);
- *rule_preconditions* (rule_label, variable_name, variable_state); and
- *rule_postconditions* (rule_label, variable_name, variable_state).

In the next section we describe how the system specifications are used to create a Petri net model of the controlled manufacturing process for the purposes of analysis.

VERIFICATION OF PROCESS CONTROL LOGIC

The software for analyzing and diagnosing potential problems in the process and controller specification is based on analytical tools from Petri nets and graph theory [1], [16]. As illustrated in figure 5, to perform the analysis the present logic verification software first builds from the specification database a Petri net model of the process, sensors, actuators and controller. The software then evaluates qualitative properties of the Petri net model and its set of reachable markings to identify problems or inconsistencies in the control logic specifications.

To construct a Petri net model from the system specifications, the software first assigns Petri net places to each of the discrete values possible for each of the discrete state variables. The state transition rules in the system specifications are then translated into Petri net arcs and transitions. In constructing the Petri net model of the system, each discrete state variable comprises a state machine subgraph within the overall Petri net. State transitions in one subgraph may be conditioned on the states of other subgraphs (*unsynchronized transitions*) or they may be coincident with transitions in other subgraphs (*synchronized transitions*). For unsynchronized transitions, subgraphs are interconnected by assigning the conditioning place (state) in the one subgraph as both an input and output place for the conditioned transition in the other subgraph. This "self-loop" mechanism is used to model the dependencies of state transitions in one part of the system on states of other components of the system.

For synchronized state transitions between subgraphs, the synchronized transitions in each of the subgraphs are simply merged into a single Petri net transition. The resulting Petri net graph is represented in the computer by an incidence matrix which indicates the locations of directed arcs connecting places and transitions. The sensor, actuator, process and controller state variables all correspond to separate subgraphs in the Petri net model of the system.

Several properties of the system specification are evaluated using the Petri net model of the system. The *connectivity* of each Petri net subgraph corresponds to the ability to move from state to state within the state space for the corresponding system component. If, for example, the subgraph for a particular resource state variable is not connected, there is a potential logical inconsistency in the specification since this indicates that starting from one state, the state variable can never enter some other states defined for it. The user is notified of such problems in the specification when they are detected.

Further properties of the system specification are related to the set of reachable markings for the Petri net model and the transitions which can occur among the markings. To analyze these properties it is necessary for the manufacturing engineer to specify an initial state for each of the system's state variables. These states correspond to the initial marking of the Petri net and are entered during the interactive analysis of the system specifications. Given an initial marking for a Petri net, the *reachability set* is defined as the set of reachable markings generated by all valid firing sequences starting from the given initial marking. The *reachability graph* is a directed graph in which the nodes correspond to the markings in the reachability set and the graph arcs indicate the possible transitions between markings.

Given a initial marking of the Petri net, the analysis software generates a reachability graph which embodies a complete finite state representation of the system. In general, the set of reachable markings is very large and direct graph theoretic analysis of the reachability graph will often be computationally infeasible. In addition, such an analysis would provide results describing the behavior of the overall state of the system, obscuring the behavior of individual system components. To circumvent these problems, the analysis software generates *reduced reachability graphs* for each of the state variables in the system. In a reduced reachability graph, the overall reachability graph is collapsed around a submarking of the system corresponding to the places for a specific state variable. To collapse the reachability graph, the software simply coalesces adjacent vertices for which the marking of the given state variable is the same. Not only is the resulting graph far more compact, it also captures the behavior of a specific variable of interest rather than the behavior of the entire system.

Reduced reachability graphs are analyzed using graph theoretic techniques to determine *transient states* and *cycles* of states for the each of the system's state variables. Transient states, reached only once from the initial state during the operation of the system, correspond to start-up conditions in the process. Cycles correspond to resource activity cycles in the manufacturing process [17], that is, repeating sequences of states which occur in the normal operation of the system. The identification of transient states and cycles helps the user validate the model by explicitly representing aspects of the system behavior that are only implicit in the rule-based specification. For example, manufacturing processes normally involve repeated cycles of operations. The absence of a cycle for a resource state variable may mean there is a flaw or oversight in the specification of the transition rules for the process or control logic.

The existence of global deadlocks in the manufacturing process is manifest as *terminal nodes* in the reachability graph. Terminal nodes correspond to system states from which the system cannot exit for the given control specifications. This is normally an undesirable condition which must be rectified by the system designer so that the manufacturing process continues to operate smoothly under all foreseeable contingencies. Local deadlocks in the manufacturing process appear as terminal nodes in the reduced reachability graphs for selected state variables. Local deadlock that does not coincide with a global deadlock in the manufacturing system is referred to as *livelock*. In livelock the overall state of the system continues to change while parts of the system are in fact deadlocked. Livelock is not readily apparent in the full reachability graph for the system in which there may be no terminal nodes. This condition, however, is manifested as a terminal node in the reduced reachability graph for the the deadlocked resource.

The separation of the system specification and model into actuator, process, sensor and controller elements permits both open loop analysis of the process and closed loop analysis of the controlled system. Open loop analysis is performed by disconnecting the controller state variables from the actuator state variables and performing the analysis procedures described above. In open loop analysis the desirability of terminal nodes, transient states and cyclic sets of states is often the opposite of the closed loop expectations. Open loop analysis is generally valuable in helping validate the process model.

Problems identified by the analysis software can be interactively analyzed by the user and appropriate modifications can be made in the system specification database. The modeling methodology and analysis techniques provide a rich set of verification tests. When the process and control logic have been correctly specified, the database is ready to be used to generate the computer program for on-line control.

AUTOMATED PROGRAM GENERATION

The objective of the program generation software is to encode the control logic specified in the controller database into a real-time computer program. The program generation software automates the process of going from a specification to an implementation, taking into account the requirements of the control computer and the programming language being used.

While the controller representation discussed in section 3 is useful for generating Petri net models for analysis, we have developed an alternative representation of the control portion of the system model which is more compact and intuitive. This representation, illustrated in figure 6, allows the manufacturing engineer to formulate the control logic in terms of *operations*. Each operation represents some control function which the designer deems necessary for the proper functioning and coordination of the manufacturing hardware. The representation of the control logic in the database is independent of the particular computer to be used for on-line control.

An operation is defined by specifying some enabling *conditions* and some *actions* which are to be carried out when the operation is executed. The generated control program must check the enabling conditions for each operation and perform the associated actions if the conditions are found to be true.

The conditions for executing an operation can be a set of any of the following primitive conditions:

physical condition (pc):
: A value can be specified for a sensor indicating a physical condition in the system. The pc can be specified to be TRUE either when the sensor state is equal to the specified value, or when the the sensor state is different from the specified value.

sequence condition (sc):
: Special variables called sequence variables can be used to enforce an ordering in the execution of the operations. A sequence variable can be incremented by a RECORD action to record the execution of an operation. When used as a condition, the sequence variable must be positive for the operation to be enabled. When an operation conditioned on a sequence variable is executed, the sequence variable is decremented. Sequence variables of the form `op_n` are automatically incremented when operation number `n` is executed.

: There are two modes for using sequence variables as sequence conditions: *global* and *local*. When a sequence variable is used in global mode to condition several operations, only the first of those operations to be enabled is executed. In local mode, all operations using the same sequence variable will be executed when their other enabling conditions are satisfied.

counter condition (cc):
A counter condition can be specified to be TRUE either when a counter variable is equal to a specified value or different from the specified value. Counter variables can be incremented and reset by operation actions.

In addition to the actions mentioned above for modifying sequence and counter variables, the following operation actions are defined:

physical action:
The controller initiates a physical action in the process by writing a specified value to an actuator port. In the representation physical actions can be specified to occur before a delay (**pabd**) or after a delay (**paad**).

delay: A real-time delay of a specified number of clock ticks can be assigned to an operation with physical actions occurring before and/or after the delay as described above.

As an example of the controller representation, we consider the specification for two operations belonging to the control logic for a conveyor-based part replacement system. In this system, a workpiece to be processed is held in place by two stops called the *front* and *rear* stops. The front and rear stops are lowered when actuator ports **STOP_1** and **STOP_2**, respectively, are set to 1. When these actuator ports are set to 0 the stops are raised. The workpiece rests on the conveyor. The conveyor starts running when actuator port **AC3** is set to 1 and stops when **AC3** is set to 0. The sensor port **LS3** indicates the presence of a piece between the stops: when it is 1 a piece sits between the stops, when it is 0 the piece is not present between the stops.

The two operations considered in this example are described as follows. Suppose both stops are raised and there is a part present between the stops. In the first operation the front stop is lowered and the conveyor moves forward so that the part is displaced from its position between the stops. In the second operation the front stop is raised back up, the rear stop is lowered, and the conveyor moves forward so that a new part occupies the position between the stops. The control logic for these two operations is represented as follows:

```
op_name: lower STOP_1, start CONV
op_number: 1
sc: start_rep  G
pc: LS1  0  T
delay: 10
pabd: STOP_1  1
paad: AC3  1

op_name: stop CONV, switch stops
op_number: 2
sc: op_1  G
pc: LS3  0  T
delay: 10
pabd: AC3 0
pabd: STOP_1 0
pabd: STOP_2 1
paad: AC3  1
```

In the representation above, the first operation (**op_number 1**) is enabled when the sequence variable **start_rep** is positive and the sensor **LS1** has value 0. Sequence variable **start_rep** is incremented in some other operation in the conveyor control logic. Execution of operation 1 begins with the lowering of the front stop. Then, following a delay of 10 clock ticks the conveyor motion is initiated. The execution of this operation automatically increments the sequence variable **op_1** which is used as a sequence condition for the second operation.

In addition to the sequence condition **op_1**, the second operation (**op_number 2**) has the physical condition **pc: LS3 0**, indicating the workpiece has moved. This operation stops the conveyor, raises the front stop and lowers the rear stop. After a delay of 10 clock ticks the conveyor motion is again initiated. Another operation stops the conveyor and completes the cycle for indexing workpieces through the system.

To enter the control logic representation, the designer builds a datafile containing the operation definitions which is loaded into the controller database. he control logic representation in the database is transformed by the program generation software into C source code for the on-line control computer. Each operation in the controller database is translated into a conditional statement and placed within a *while* loop construct which results in a repeated polling of all the operations. This provides an efficient monitoring of the sensor states and allows concurrent control actions within each polling cycle. The enabling conditions for each operation are scanned in turn, and the specified actions are executed whenever the operation conditions are satisfied.

The generated control program uses a function *read_sensor* to query sensors ports. This function is specific to the particular hardware configuration being employed. In evaluating each set of operation conditions, the control program checks the physical conditions (sensor states) last, and only when the other conditions are true. This minimizes the time committed to sensor queries in each scan cycle.

The real-time clock in the control computer is used to realize delay actions in the operations. The scan of the operations is not halted for a delay action. Rather, an operation is disabled for the duration of a delay action by setting a *delay flag*, and the operation scan cycle continues.. After the specified number of clock ticks, the remaining operation actions for the delayed operation are executed and the delay flag is reset. During a delay, the conditions for the delayed operation are not evaluated. Thus, once an operation has been initiated, its enabling conditions are not evaluated again until all actions before and after the operation delay have been executed.

The following C code was generated by the program generation software for the two operations in the conveyor example discussed above:

```
if( (flag_0) &&( start_rep > 0 )
        && ( read_sensor(LS1) == 0))
{
        flag_0 = 0;
        start_timer( 0, 10);
        write_actuator( STOP_1, 1);
}
        if (end_delay[0]) {
        end_delay[0] = 0;
        flag_0 = 1;
        write_actuator( AC3, 1);
        start_rep--;
        op_1_1++;
}
/*----------------------------------*/
if( (flag_1) &&( op_1_1 > 0 )
        && ( read_sensor(LS3) == 0))
{
        flag_1 = 0;
        start_timer( 1, 10);
        write_actuator( AC3, 0);
        write_actuator( STOP_1, 0);
        write_actuator( STOP_2, 0);
}
        if (end_delay[1]) {
        end_delay[1] = 0;
        flag_1 = 1;
        write_actuator( AC3, 1);
        op_1_1--;
        op_2_1++;
}
```

In this example the boolean variables **flag_0** and **flag_1** are used to disable the conditional statements for the operations during delay actions. The flags are reset when the delay is over. The function **start_timer** is used to enable the delay clocks for the required duration. The sensor **LS1** is queried by the **read_sensor** function and the actuator **AC3** is written to by the function **write_actuator**.

REAL-TIME VERIFICATION

Final verification and evaluation of the control program requires that it be executed in real time with physical actuator and sensor signals. One alternative is to control the real system. Indeed, this is often the only alternative for testing control programs in manufacturing systems. There are obvious drawbacks, however, to using the actual manufacturing system as the test-bed for debugging the control program. One drawback is that the system must actually be implemented before the control program can be tested. This precludes testing alternative control schemes during the planning and development phases for a manufacturing system. Furthermore, the process itself may be too critical, and mistakes too costly, to allow untested control programs to be executed with the real system. Horror stories abound in the manufacturing community of control program "bugs" which were not discovered until the actual process was being controlled, leading to catastrophic results.

To permit the testing of the computer-generated control programs under realistic conditions, while avoiding the time-consuming and costly development of a physical system to be controlled, we have implemented a real-time process simulator. As illustrated in figure 1, the control computer is connected to the real-time simulator by physical lines which carry signals identical to the physical actuator and sensor signals. The control program can then be put through its paces to make sure that there are no critical timing problems or logical inconsistencies which were not detected by the logic analysis software.

Our real-time process simulator is based on the discrete event simulation kernel in FACTOR, a commercial software package for on-line scheduling of factories [18]. The requirements for real-time simulation differ, however, from standard discrete event simulation in two significant ways. First, real-time simulation of process delays must result in actual delays. For example, in the real-time simulation of the stamping process discuss above, if the control computer issues a signal to initiate a motion by the robot and it takes three seconds for the robot to perform the motion, then the process simulation must wait three seconds before sending the sensor signal to the control computer indicating the robot is ready again. Thus, "simulation time" becomes equivalent to "real time". The second way in which real-time simulation differs from standard discrete event simulation is that external inputs (control signals) must generate entries in the simulator's event calendar.

To implement these real-time features we have developed a *timing process* within FACTOR which interacts with the simulation event calendar and data structures. The timing process is a C procedure which handles the physical I/O for actuator/sensor signals, and introduces physical delays to synchronize the FACTOR simulation clock with the real-time clock in the simulation computer. The real-time process simulator allows us to configure realistic models of manufacturing process to be controlled by the computer-generated control programs. Standard features of FACTOR enable us to easily build and modify simulation models, and collect and tabulate data from the control experiments.

FUTURE RESEARCH

This paper describes the current development of software and analysis tools for the evaluation and generation of control programs for discrete manufacturing processes. The objective of this work has been to provide practical experience and insight into the design of controllers for an important class of systems. This work parallels and complements our theoretical work on the analysis and synthesis of controls for general discrete event systems.

There are several directions for future work. The representations of the process and control logic need to be expanded to include more aspects of real systems, including non-binary signals and more sophisticated timing mechanisms. We are also investigating the representation and generation of distributed control logic in which several computers control different portions of the manufacturing process. One possibility is to have the computer automatically decompose the system specifications into an appropriate distribution for multiple control computers.

We are also considering ways to enhance the automatic program generation so that the expertise of a human programmer is emulated in the generated code. Currently, the operations are scanned in an arbitrary order determined only by the order in which they are encountered in the database. A more sophisticated approach would be to group operations according to their respective conditions and actions so as to economize on the overall scan time required. It would also be useful in many applications to be able to specify priorities and multiple scan rates for operations so that operation conditions are evaluated only when necessary.

In most applications it is essential that the representation and design tools provide techniques for representing systems hierarchically. We are investigating the use of aggregation and "macros" to facilitate the specification of large-scale systems. The reduced reachability graphs discussed above may provide an analytical tool for understanding and evaluating such a representation.

ACKNOWLEDGMENTS

This research was supported in part by the Robotics Institute, FACTROL, Inc., the AT&T Foundation, Bell-Northern Research Ltd., and the National Science Foundation under research grant number DMC-8451493.

References

1. J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.

2. W. Reisig, *Petri Nets: An Introduction*, Springer-Verlag, New York, Monographs in Theoretical Computer Science, Vol. 4, 1985.

3. M. Kamath and N. Vishwanadham, "Applications of Petri Net Based Models in the Modeling and Analysis of Flexible Manufacturing Systems", *IEEE International Conf on Robotics and Automation*, Apr 1986, pp. 312-317.

4. D. Dubois and K.E. Stecke, "Using Petri Nets to Represent Production Processes", *Proceedings of the 22nd IEEE Conference on Decision and Control*, December 1983, pp. 1062-1067.

5. Y. Narahari, and N. Vishwanadham, "A Petri Net Approach to the Modelling and Analysis of Flexible Manufacturing Systems", *Annals of Operations Research*, 1984, pp. 449-472.

6. P. Alanche, *et al*, "PSI: A Petri net based simulator for flexible manufacturing systems", *Advances in Petri Nets 1984*, Springer Verlag, 1985, pp. 1-14.

7. C.L. Beck and B.H. Krogh, "Models for Simulation and Discrete Control of Manufacturing Systems", *Proceedings of the 1986 IEEE Conf on Robotics and Automation*, April 1986, pp. 305-310.

8. B.H. Krogh and C.L. Beck, "Synthesis of Place/Transition Nets for Simulation and Control of Manufacturing Systems", *Proceedings 4th IFAC/IFORS Symposium Large Scale Systems*, Aug 1986, to appear

9. J. Martinez, P. Muro, and M. Silva, "Modeling, Validation, and Software implementation of production systems using high level Petri nets", *Proceedings of the 1987 IEEE Conf. on Robotics and Automation*, Raleigh, NC, March 1987, pp. 1180-1185.

10. G. Balbo, et al, "Generalized stochastic Petri nets for the performance evaluation of FMS", *Proceedings 1987 IEEE Conf. on Robotics and Automation*, Raleigh, NC, March 1987, pp. 1013-1018.

11. G. Bruno and M. Morisio, "Petri-net based simulation of manufacturing cells", *Proceedings 1987 IEEE Conf. on Robotics and Automation*, Raleigh, NC, March 1987, pp. 1174-1179.

12. T. Murata, N. Komoda, and K. Matsumoto, "A Petri-net Based Factory Automation Controller for Flexible and Maintainable Control Specifications", *Proceedings of the IECON'84*, IEEE, International Conference on Industrial Electronics, Control and Instrumentation, October 1984, pp. 362-366.

13. M. Courvoisier, R. Valette, J.M. Bigou, and P. Esteban, "A Programmable Logic Controller Based on a High Level Specification Tool", *Proceedings of the IECON'83*, IEEE, International Conference on Industrial Electronics, Control and Instrumentation, 1983.

14. D.H. Crockett and A.A. Desrochers, "Implementation of a Petri net controller for a machining workstation", *Proceedings 1987 IEEE Conf. on Robotics and Automation*, Raleigh, NC, March 1987, pp. 1861-1867.

15. G. Bruno and G. Marchetto, "Process-translatable Petri nets for the rapid prototyping of process control systems", *IEEE Trans. on Software Engineering*,Vol. SE-12,No. 2,Feb 1986, pp. 346-357.

16. F.S. Roberts, *Discrete Mathematical Models with applications to social, biological and environmental problems*, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1976.

17. C.L. Beck, "Modeling and Simulation of Flexible Control Structures for Automated Manufacturing Systems", Tech. report, Robotics Institute, Carnegie-Mellon University, 1985.

18. FACTROL, Inc., *FACTOR Implementation Guide*, 3.0 ed., West Lafayette, IN, 1987.

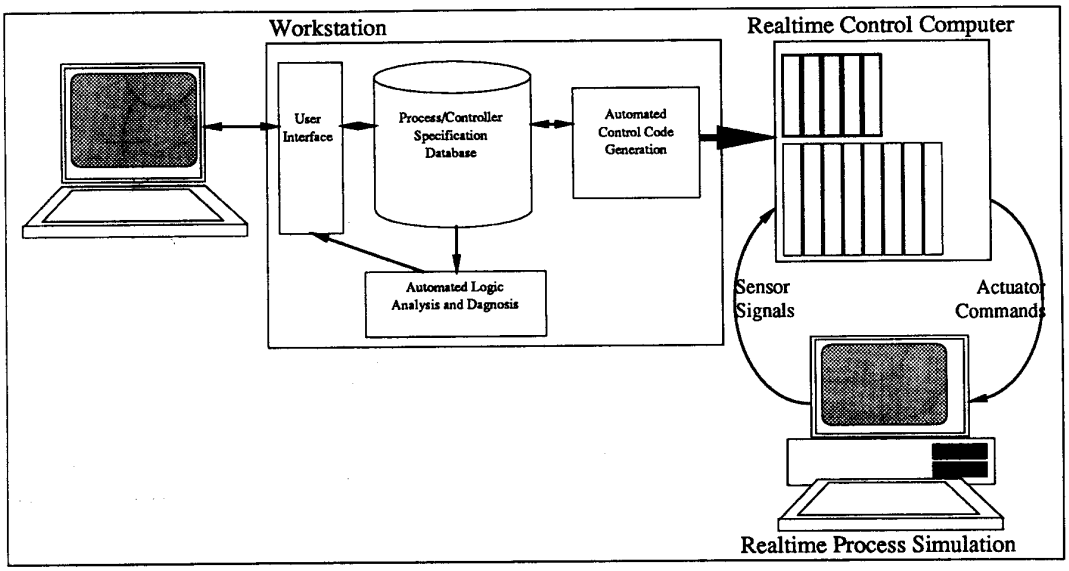

Figure 1: Development system for generation and verification of real-time control programs.

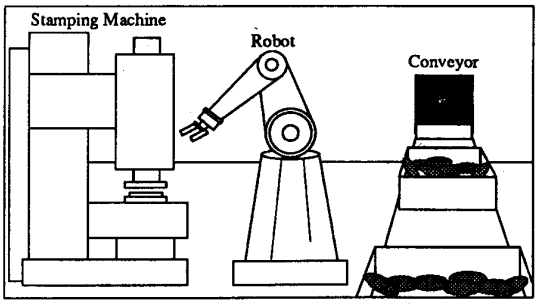

Figure 2: A discrete manufacturing process.

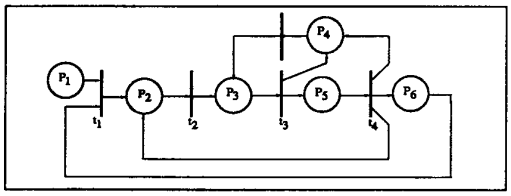

Figure 3: An ordinary marked Petri net.

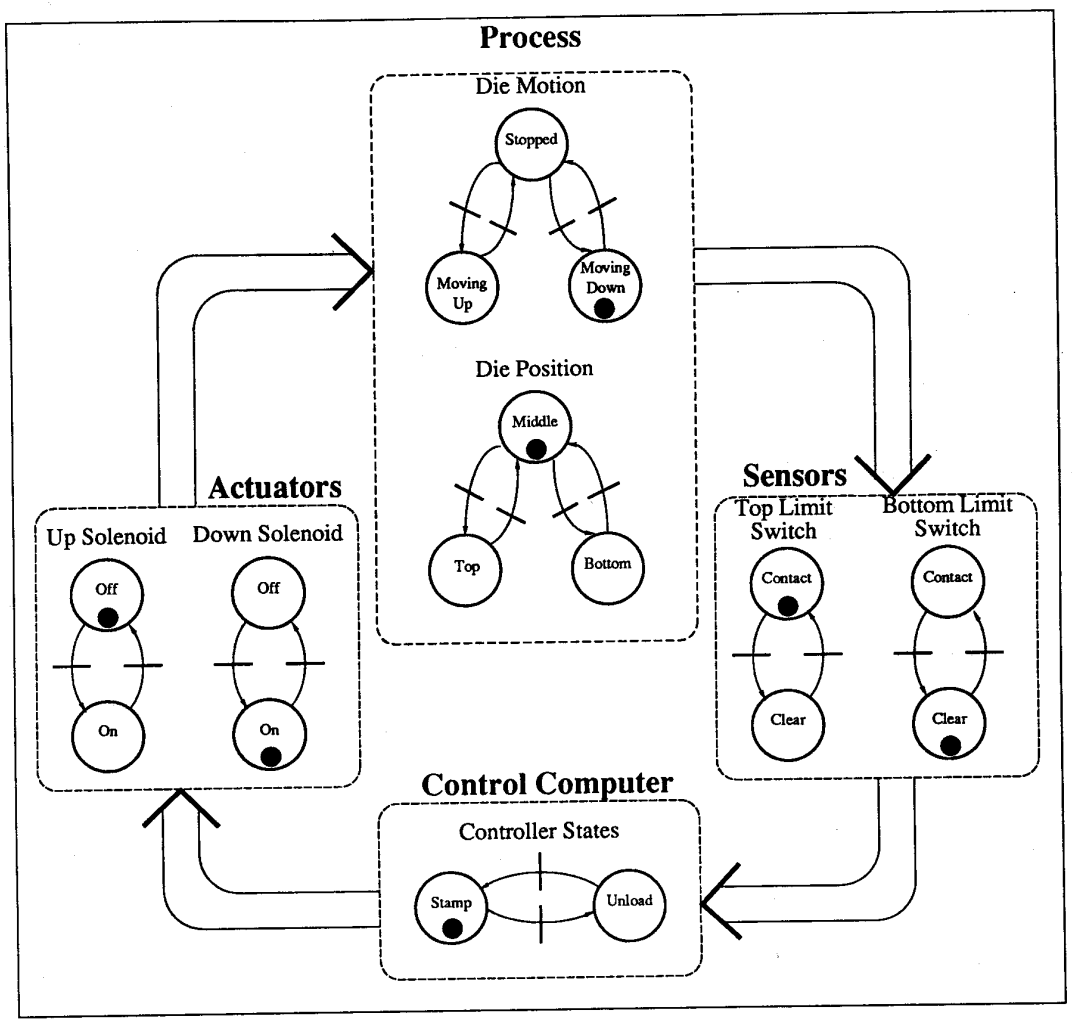

Figure 4: Decomposition of Petri net models for feedback loop

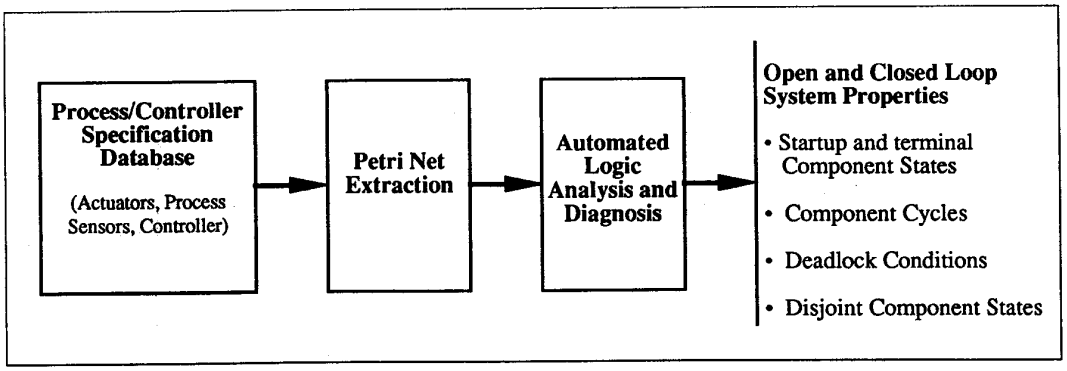

Figure 5: Software for Petri net-based verification of process and control logic specifications.

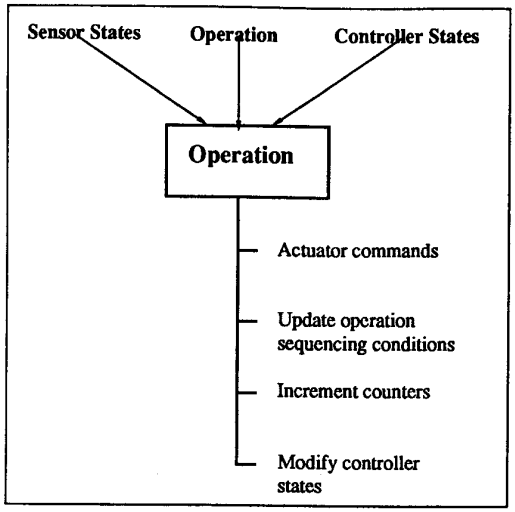

Figure 6: Operation specification for controller.