IEEE TRANSACTIONS ON

KNOWLEDGE AND DATA ENGINEERING

A publication of the IEEE Computer Society

MARCH / APRIL 1999 VOLUME 11 NUMBER 2 ITKEEH (ISSN 1041-4347)

REGULAR PAPERS

CORRESPONDENCE

Dr. Naphtali Rishe
Florida International University
School of Computer Science
Southwest 8th St. and 107th Avenue
University Park
Miami, FL 33199

IEEE COMPUTER SOCIETY

http://computer.org • tkde@computer.org

ELSEVIER

The Journal of Systems and Software 45 (1999) 61–78

The Journal of Systems and Software

Architecture-driven modeling of real-time concurrent systems with applications in FMS [1]

Yi Deng *, Chia-Rung Yang

School of Computer Science, Florida International University, ECS 359, University park, Miami, FL 33029, USA

Received 14 March 1997; received in revised form 23 June 1997; accepted 22 July 1997

Abstract

Petri nets have become increasingly popular for flexible manufacturing systems (FMS) modeling and control because they accurately capture the concurrent, non-deterministic and time-dependent properties of the systems. While offering many advantages, conventional Petri net models suffer from some serious problems that limit their usability as design models for complex FMS. Central to these problems is the lack of an engineering support for incremental design, refinement, and analysis of large-scale systems. In this paper, we present an architecture-driven approach for the modeling and design of FMS that effectively addresses the problems while leveraging the strengths of Petri nets. The approach has two major aspects: The first is a Net-based and Object-based Architectural Model (NOAM) that introduces a well-founded architectural framework into the Petri nets notation and lays a foundation to support formal design. The second is a modeling method based on NOAM that uses architecture decomposition and refinement as the basis to reduce design complexity, to provide smooth transition from informal to formal design, and to support incremental refinement and analysis. A case study using NOAM for FMS modeling is provided to show the applicability of our approach.

1. Introduction

Flexible manufacturing systems (FMS) are aimed to provide a means to achieve better quality, lower cost, and smaller lead time in manufacturing. An FMS is a typical real-time concurrent system composed of a number of computer-controlled machine tools, automated material handling and storage systems that operate as an integrated system under the control of host computer(s) (Greenwood, 1988). The growing demand for higher performance and flexibility in these systems and the interlocking factors of concurrency, deadline-driven activities, and real-time decision making pose a significant challenge in FMS design, especially in terms of control and scheduling. Moreover, uncertainty in product demand knowledge, finite manufacturing capacity, random machine failures and repair rates further make the system behavior more dynamic and hard to predict (Looveren et al., 1986). Given the complexity, an ad hoc way for FMS design is clearly inadequate, and an engineering methodology to address the complexity and dynamics in FMS modeling, design and analysis is essential (Elmaraghy and Ravi, 1992).

Petri nets are a popular mathematical-based, graphical modeling tool for concurrent and distributed systems (Genrich and Lautenbach, 1981; Ghezzi et al., 1991; Jensen, 1992; Palaudetto and Raymond, 1993). In addition to its rigor and analytic capability (e.g., for deadlock detection, performance evaluation, real-time schedulability analysis), Petri net models are executable, and thus can be used as a system prototype for simulation. Furthermore, Petri nets are capable of describing both software and hardware, system and environment, at different levels of abstraction. Because of these strengths, Petri nets are increasingly used as a modeling tool for FMS (D'souza and Khator, 1994; Solot and Vliet, 1994).

While offering many advantages, ordinary Petri net models suffer from some basic problems that limit their usability and application as a design model for complex FMS:

* Corresponding author. Tel.: +1 305 348 3748; fax: +1 305 348 3549; e-mail: deng@cs.fiu.edu

[1] This work was supported in part by the National Science Foundation under Grant No. CCR-9308473, by the Air Force Office of Scientific Research under Grant No. F49620-96-1-0221, and by NASA under Grant No. NAGW-4080. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied by the above named Agencies.

0164-1212/99/$ – see front matter
PII: S0164-1212(98)10068-7

First, Petri net-based models tend to become too large even for a modest-sized problem (Murata, 1989). This makes it extremely hard to build, understand, change and analyze nets-based models for practical complex problems.

Second, the primary concern of Petri nets, like many other formalisms, is behavior modeling and analysis. Most of Petri net-based models do not provide explicit or adequate support to modeling, refinement, and analysis of system architecture – the organization of the system into components and their interactions at a sufficient high-level of abstraction (Luckham et al., 1995a, b), a central concern in complex system design. This creates a significant semantic and syntactical gap between an actual FMS design and its Petri net model. For a typical FMS, the configuration of various hardware components, e.g., machine tools, automatically-guided vehicles (AGV), buffer stations, are normally quite stable. The challenge is the "soft" aspect of the design, i.e., the control and scheduling of the components, where repeated changes are to be made, different design alternatives need to be explored. Without a clearly defined interface and connections between the components, any change to the control/scheduling design will likely have a global impact, which requires reworking the entire model, and voids the results of analysis done earlier.

Last but not the least, conventional Petri nets lack support for incremental design. Constructing a complex Petri net model is largely an ad hoc rather than an engineering practice. No established methodology or procedure is available to support systematic design and refinement.

In this paper, we present a formal approach for modeling and design of FMS that effectively addresses the above problems while leveraging the strengths of Petri nets. We discuss this approach from two aspects:

The first is a Net-based Object-based Architectural Model (NOAM) (Deng et al., 1996, 1997; Du and Deng, 1995). Building on the theory of Petri nets, NOAM provides a framework that incorporates well-founded architectural design concepts and structure into the abstract notation of Petri nets. It provides a formal foundation to support incremental design, refinement, and analysis of complex real-time systems like FMS. Compared to conventional Petri nets, a NOAM model of FMS design can be easily extended, refined and changed. Furthermore, NOAM supports "plug-and-play" in the sense that different subsystem models can be plugged-into a high-level design to evaluate the alternatives of design without having to change the overall design structure.

Second, based on the NOAM framework, we further present a modeling method that defines a systematic and incremental process to construct, analyze and refine complex FMS designs in terms of one or more cycles. In each cycle, it guides system designer to establish a clear focus for each step of the design while minimizing distractions from other design issues. To ease the burden of constructing a complex formal design from scratch, we present a mechanism that provides a smooth and incremental transition from an informal to a formal system model. The informal phase emphasizes the global control and communication structure of the system design, and the formal phase rigorously defines the semantics of the design.

The rest of the paper is organized as follows. In Section 2, we briefly summarize the application of Petri nets in FMS and compare our work with existing structured or object-based Petri net models. In Section 3, we introduce the basic modeling framework of NOAM and provide more background information of this research. A simple but typical FMS is described in Section 4, which is used as a case study to illustrate and demonstrate our modeling approach and its advantages. In Section 5, we present a modeling methodology for NOAM, and use the method to incrementally construct a formal design of the FMS described in Section 4. Finally, we conclude this paper in Section 6 and point out future research directions.

2. Related work

In this section, we first briefly summarize existing work using Petri nets for FMS modeling. We then evaluate existing efforts to address the problems raised in the introduction, and compare them to our work.

Petri nets have been applied to the specification, verification, performance analysis, real-time control and simulation of FMS. Net-based models have also been used to obtain production rates, throughput, delays, capacity, resource utilization, reliability measures and deadlock avoidance for FMS. The details of these applications can be found in surveys by D'souza and Khator (1994) and by Shukla and Chen (1997). Some typical uses of Petri nets in FMS modeling are listed below (Brussel et al., 1993; Camurri et al., 1993; Huang and Chang, 1992; Knapp and Wang, 1992; Lin and Lee, 1995; Martinez et al., 1986; Meng et al., 1995; Qadri and Robbi, 1994; Solot and Vliet, 1994; Zuberek, 1995; Zhou et al., 1992; Zhou et al., 1993, 1995):

- Nets can be used as a graphical modeling tool to visualize complex control structure and systems behavior.
- The nets notation precisely captures the precedence relations and structural interactions of stochastic, concurrent and asynchronous events, which can be used to support the development of operating strategies for work scheduling and job sequencing.
- The executibility of Petri net models enables them to be used as simulation tools to evaluate control and scheduling policies.

- Real-time Petri net models can be used to design and implement real-time control systems.
- Their analytical capability supports deadlock detection, performance evaluation, and verification of real-time schedulability.
- Conflicts and buffer sizes can be modeled easily and efficiently.
- Net models can help to identify production bottleneck, and to assess the capacity and utilization of equipment.

The focus of this paper is different from the above studies. While those studies deal with the issue of how to use Petri nets to address specific modeling and analysis problems in FMS, the goal of our effort is to develop an engineering practice to systematically and cost-effectively apply Petri net theory in complex FMS modeling, design and analysis.

The problems associated with Petri nets are well-documented, and have been the subject of many investigations. Some early techniques to resolve the problems include high-level Petri nets (Genrich and Lautenbach, 1981; Jensen, 1992), which introduce variables into tokens, and enable a more compact system model, and hierarchical Petri nets (Jensen, 1992), which describe systems in terms of hierarchically structured subnets. Although the hierarchical net models introduce hierarchical structure into the net notation, the subnets do not directly correspond to specific design concepts, and there is no mechanism to ensure that the subnets are self-contained modules with clearly defined interfaces. Consequently, any change to a subnet (or supernet) may affect any other part of a system model.

One promising way to handle complexity is to introduce the concept and structure of object into the formal notation. The modular and encapsulated nature of object, and its support for hierarchical modeling provide a way to manage complex model building. In addition, the object model offers a natural paradigm for distributed systems because objects naturally model the autonomous and cooperating entities.

Based on the object concept, several structured Petri net models are proposed. Among them, PROTOB (Baldassari and Bruno, 1991) is proposed for process control systems modeling, and is based on an extended data flow model and Petri nets, which specifies systems as inter-related objects with I/O ports as their interface of communication. In PROTOB, the object (module) inner control structure and communication parts are clearly separated and described using different representations. The former is represented using PROT net, an object-based extension of Petri nets, and the latter is defined by input/output places and the links between them, which simulates transmission lines that define the route of messages (tokens). The types of input and output places connected by a link must be the same because the link is not allowed to have any procedure to adapt to change message type. It is unclear how the formal semantics of Petri nets is preserved under the different representations, and how to describe real-time properties since no formal notation of time is given in the description of the model.

The *Cooperative Objects* language (Bastide et al., 1993) is designed to facilitate the specification of objects with concurrent behavior. In Cooperative Objects, a system component (object) is modeled by two tightly coupled subnets, one for describing the interface and behavior of the component, and the other for the implementation of the behavior. The objects are connected together based on client–server relationship, where the interface specifies the services provided to, and the implementation specifies the services (if any) required of its environment. This model does not have a formal definition of time or time-dependent properties in its representation. Furthermore, since part of the inter-object communication is described in the implementation of the object, any change to the implementation may affect the control structure of the entire model. This setup is inconsistent with the architectural principles described in (Luckham et al., 1995a, b), and affects the stability of the design.

In *OPNets* (Lee and Park, 1993), a system component (object) is modeled by a high-level Petri net with input/output gates and queues as its interface to its environment. Objects are encapsulated to ensure modularity. High-level components can be constructed by using lower-level objects as building blocks. Communication between objects are established by connecting the interfaces of related objects with gates that form a subset of the transitions. An application of OPNets in FMS modeling and analysis is presented in Wang, 1996, where they are used to represent part of an object-oriented Petri net cell control model (OPNCC). Among the above models, OPNets is most close to NOAM. They differ in two aspects: First, as indicated by the authors (Lee and Park, 1993), OPNets don't contain real-time notation in their definition, which is necessary to describe real-time constraints and time-dependent priorities. Second, for a OPNet model to be semantically complete and executable, a system must be decomposed to the lowest level where system components are primitive sequential objects. NOAM, on the other hand, provides an explicit notion of module specification (MS) and *module design* (*MD*) (see Section 3), which enable us to model a system at arbitrary level of abstraction.

Another structured net model is Communicating Time Petri Nets (CmTPN) (Bucci and Vicario, 1995). Unlike the above models, CmTPN has a well defined real-time semantics, and supports reachability-based compositional verification that helps to reduce the complexity of analysis. In CmTPN, self-contained time Petri nets (modules) communicate with environment via *reading* and *writing places* (*ports*). A reading place and a

writing place connect to each other through one-to-one channels.

In addition to the above models, there are also some other Petri nets-based models employing the object concept. Examples are OBJSA nets (Battiston et al., 1988) and POT (Engelfriet et al., 1990). However, these models do not provide a comprehensive structuring mechanism like the models mentioned above. Although most of these models provide certain support to describe system structure, architectural representation and design are neither a primary concern nor explicitly addressed. This problem is consistent with the nature of the object model. That is, the object model is desirable for constructing, manipulating and reusing components of systems. However, what is missing from this model is a "global system view" – the plan of how the components fit together to form a coherent system (Luckham et al., 1995a, b).

3. A net-based and object-based architectural model

In this section, we present the NOAM framework and notation. Since the main focus of this paper is modeling methodology rather than formal semantics, the use of mathematical notation is avoided in our discussion whenever possible. The formal definition of the NOAM representation can be found in (Du and Deng, 1995).

3.1. The modeling framework of NOAM

The objective of NOAM is to serve as the foundation for a formal engineering methodology for real-time distributed system design. To achieve this goal, NOAM possesses certain basic properties or features. In this section, we discuss what these properties are, present the basic framework for NOAM, and show how the framework supports the properties. We focus on the concepts associated with the NOAM framework, while deferring the discussion of specific notation and its semantics to later sections.

As a design representation (as opposed to a behavioral model), NOAM should accurately describe the organization of the system it models. In addition to system behavior, NOAM also describes how the components of the system work together to achieve such a behavior. This is exactly the issue of architecture. For a real-time distributed system like FMS, some basic issues include decomposition of the system into components, the global control structure, the communication and synchronization of the components, and the time-dependent system behavior embodied by such an organization.

Complex system design is an evolutionary process. A good design model should be flexible enough to allow frequent changes to the design and to support the exploration of various design alternatives, while minimizing the impact of such changes to the overall design. This is especially important to FMS modeling, where the physical configuration and the functional behavior of its hardware components, e.g., machine tools and AGVs, are quite stable and static. The central issues of design are the coordination, control and scheduling of these components. For a complex FMS, it is necessary to experiment with different alternatives of control and scheduling policies against the same hardware configuration. It is therefore highly desirable to be able to "plug-in" the specifications of various control modules to a FMS model without having to make major changes or re-construct the entire system model each time.

The modeling and design of a complex FMS must be a systematic process. An ad hoc way of modeling is not only difficult and hard to apply in a large scale, but also prone to errors and hard to achieve a system structure that is extensible and maintainable.

Last but not the least, a good FMS model must support analysis and verification of important system properties *as the design evolves*, which may take place in forms of either formal verifications or simulation and prototyping. This requires that a methodology that progressively produces complete, analyzable and executable design models of the FMS at different levels of detail, and a proper balance between expressiveness and analyzability.

In the following, we discuss the framework of NOAM, and show how NOAM supports the above features. The basic structure of a NOAM model is illustrated by Fig. 1. A NOAM model is layered specification of a FMS organized under two basic levels of abstractions: Module Specification (MS) and Module Design (MD). An MD describes a design of a subsystem in terms of the connection and interaction of its components, e.g. CNC machines, AGV or Cell Controller, where each component is represented by an MS, which describes the communication interface (in terms of *input* and *output communication ports*) and the required behavior of the component, hence the name of MS. A component may represent the entire FMS, a subsystem (such as cell controller, AGV, CNC machine and load/unload station, etc.), or a primitive object (such as operator and transporter). It is specified in Fig. 1, for instance, an interaction between MS2 and MS3 must be synchronized with MS1. The focus of MD is the structure and control of the system, while the focus of MS is the specification of the component.

The concept of MS is different from the concept of object interface in an object-oriented model, and different from the existing object-based Petri nets models. First, all connections between a component and its environment are specified in the interface. This is in contrast to the Cooperative Objects model (Bastide et al.,

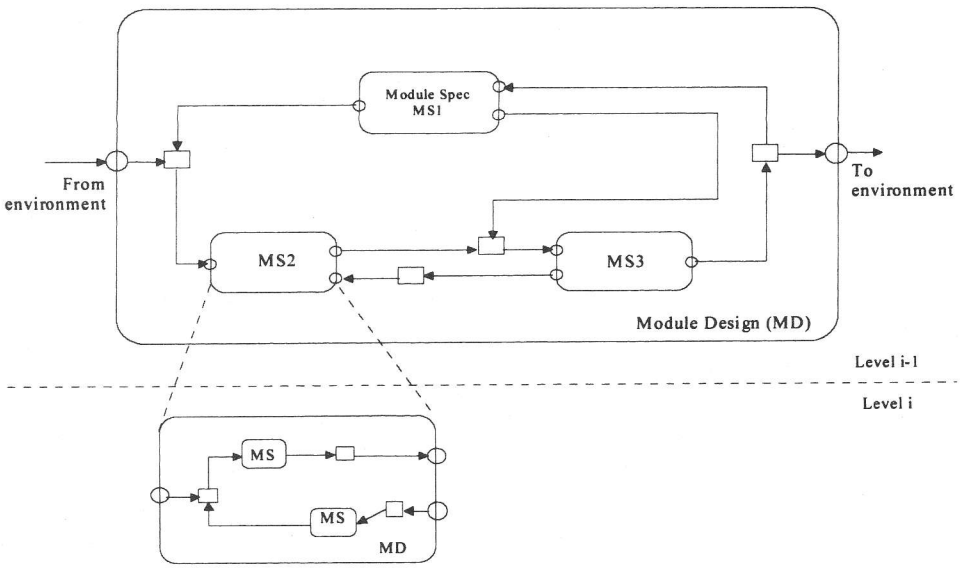

Fig. 1. The basic structure of NOAM.

1993), for example, where outgoing connections are defined in the implementation of the object. Consequently in NOAM, inter-component dependency at level i-1 of the design is determined solely based on the MSs, and independent of designs at the next and more detailed level. This is essential in achieving design stability (D'souza and Khator, 1994). Second, in addition to the signature information about the operations of the component, which defines an object-oriented interface, an MS also contains the semantics of these operations including the changes they cause to the state of the component, inter-dependency and temporal ordering among the operations, synchronization among inputs, timing constraint of each operation. Therefore, an interface specification in NOAM fully specifies the required behavior of the corresponding component. Any level of a NOAM specification is self-contained, analyzable, and executable. Consequently, a complex system can be designed, specified, and analyzed at a high-level and then gradually refined (or automatically transformed) into a more detailed and realistic design.

The NOAM model has several desirable properties: First, a module, whether an MS or MD, is a self-contained entity, which interacts with other components *only* through its *communication ports*. The internal details of the module are encapsulated and hidden from others. This feature significantly reduces the burden of complex FMS modeling as it effectively draws a boundary between different components. Thus, the model of each component or subsystem can be constructed incrementally. Any change to the internal structure of a module will not affect other parts of the design so long as its interface defined by its communication ports remains the same.

Second, the concepts of MS and MD provide an abstraction to support incremental design and exploration of alternatives of design. To see why, consider a NOAM design at level i-1 as illustrated in Fig. 1, one can incrementally replace the specification of a subsystem (MS) with an MD that confirms with the interface of the MS and represents the design of the subsystem at the next and more detailed level if so desired, and do so in such a way that no other part of the design needs to be redone. Furthermore, we can "plug-in" different MDs in place of the same MS that represent the different design alternatives to the same subsystem to readily compare and evaluate the different designs without having to reconstruct the entire system model every time.

Third, the correspondence between MS and MD helps to enhance design traceability that are crucial to ensure design correctness and consistency for complex systems like FMS. MS represents the desired behavior of a system component, while MD describes a design that is consistent with the behavior. By systematically replacing MSs with matching MDs, we can consistently trace and analyze how a complex system requirement is satisfied by progressively detailed designs. By evaluating whether the design of a subsystem (MD) is consistent with the corresponding MS, we can determine whether the subsystem design is correct or consistent with its requirements.

Together, these features lay a foundation for a systematic approach for complex FMS design that is lacking in the ordinary Petri nets. Furthermore, as will be discussed in the later sections, the NOAM framework is integrated with a formal notation of time based on Time Petri nets (TPN) (Merlin and Faber, 1976) for modeling and analyzing the temporal and time-depen-

dent behavior inherent to FMS, which is missing in other structured or object-based Petri net models like OPNets (Lee and Park, 1993).

3.2. Overview of NOAM notation

Fig. 2 shows the MS of an abstract component $M1$. The outer box with round corner identifies the boundary of the component. The circles with different fillings drawn on the boundary are called *communication ports*. The ports are a special class of Petri net places, and they define the component interface, from which messages or signals arrive at or leave the component. A port with an inside cross (e.g., g_1) is called an *input port*. An input port can receive messages (tokens) of a specified type. The message may represent an invocation to an operation defined on $M1$ or a signal. An input port is defined as an n-tuple:

$<$ *port_name*, *input_parameter_list* $>$, where

port_name is the name of the associated operation and *input_parameter_list* (omitted in this example) specifies the format of the input token. A port with an inside triangle (e.g. g_5) is called an *output port*, from which a communication (via token) to another component is specified. It is defined as a quadruple:

$<$ *port_name*, *dest_object_id*, *dest_port_name*, *argument_list* $>$, where

port_name is the name of the port; *dest_object_id* and *dest_port_name* are the names of destination component and the port on the component, respectively; and *argument_list* specifies the format of the token to be sent to the destination component. The token may represent a piece of data, a command or a signal.

Inside the MS is Time Petri Net (Berthomieu and Diaz, 1991; Merlin and Faber, 1976) specifying the behavior of the component. Informally speaking, the net is a bipartite graph consisting of two disjoint sets of nodes:

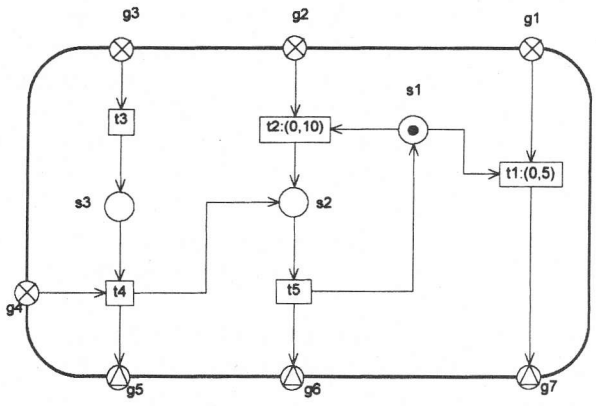

Fig. 2. An example of module specification (MS).

places, denoted graphically by circles, and *transitions*, denoted by boxes. A labeled and directed arc connects a transition to a place or vice versa. Given a transition t, a place p is called an input (or output) place of t if there is an arc from p to t (or from t to p). The set of input (or output) places of t is denoted by $I(t)$ (or $O(t)$). The label on an arc from p to t (or from t to p) is denoted by $L(p,t)$ (or $L(t,p)$).

The places represent passive elements of a net, such as states or conditions. A place may hold one or more *tokens*. (The maximum number is defined by the capacity of the place, which has a default value of ∞ if not specified.) The assignment of tokens at a particular place is called the *marking* of the place. The assignment of tokens to all places in the net (at a given time instant) is the *marking* of the net (at that instant), which defines the state of the net (hence the component it specifies).

The transitions model dynamic elements, e.g., (timed) actions or events. A transition t may be associated with a logical expression $e(t)$ serving as the *guard* for the corresponding event or action. By default, $e(t) = TRUE$. As shown in Fig. 2, t may be associated with a time interval $\tau(t) = (t_{\min}, t_{\max})$, where $t_{\min}, t_{\max} \in Q$, where Q is the set of non-negative rational numbers, and $t_{\min} \leqslant t_{\max}$. The time interval defines a duration during which t may *fire* (i.e., the event or action associated with t may occur) from the time it is *enabled*. If not specified, the default time interval is $(0, \infty)$. To model deadline-driven behavior, NOAM uses *strong time semantics*, i.e., an enabled transition must fire within the time bound by $(t_{\min}, t_{\max})$ if it is not disabled by the firing of another transition.

The labeled arcs L represent a flow relation used to model control/data flow between the places and transitions. When a transition t fires, tokens specified by $L(p,t)$ for each $p \in I(t)$ are removed from p and tokens specified by $L(t, p)$ for each $p \in O(t)$ are deposited in p, thus changing the net marking and, consequently, the state of the system.

Using the above notation, the essential behavior of a component in a real-time concurrent system can be precisely and conveniently specified. For example, the MS in Fig. 2 specifies the following properties, among others:

- *High-level abstraction*. The entire operation associated with port g_1 is specified by a single transition t_1. The required time duration for the operation is (0,5), i.e., it must be completed in at most five time units from the time the corresponding message is received.
- *Mutual exclusion*. Operations (associated with ports) g_1 and g_2 are mutually exclusive; the winner must obtain the token in place s_1.
- *Concurrency*. Since operations associated with g_1 and g_2 do not have mutual dependency, they may be executed in parallel.

- *Synchronization*. The processing of messages (tokens) arriving at gates g_3 and g_4 must be synchronized at transition t_4, which may mean that the execution of g_3 requires a reading at t_4 of sensor data coming from g_4, for example.

In addition, each MS should have an initial marking, representing the initial state of the object when it is created. In the example, the initial state is defined by the token in place s_1.

While the MS specifies the behavior of a subsystem, the MD describes a design that is consistent with the behavior specified by the MS. It is in the MD where the subsystem specified by the MS is decomposed into interconnected sub-components, communication and synchronization with its environment is expressed, and control flow of the operations defined in the MS is detailed. Fig. 3 shows an imaginary module design for the MS in Fig. 3. The round-box icons (called MS icons) marked *M*11.*MS* and *M*12.*MS*, which represent the MSs of the sub-modules created during the decomposition. With this structure, a designer will be able to treat a sub-module as a black box and focus on the architecture of the system. Formal specification for the sub-modules can be made subsequently.

From Fig. 3, it can be seen that the actions or operations represented by transitions t_1 and t_2 in Fig. 3 are now consolidated into the sub-component *M*11 (notice transitions t_{11} and t_{12} mark the beginning and end of action t_1, respectively). The time interval (0,10) originally associated with action t_1 is now allocated to newly identified module M11. In addition, the interface of the MD remains the same as the interface specified by the MS to enforce consistency of design. Since MS and MD represent different aspects of the same subsystem, consistency between the two levels of specification must be maintained to support traceability. NOAM provides basic rules for this purpose. For example, as shown in Figs. 2 and 3, all places, including ports, that appear in the MS must appear in the MD as they should have the same interface, and same model of module state. They should also have the same initial marking (initial state).

4. A simple but typical FMS

In this chapter, a simple but typical FMS is described, which is based on a real-world manufacturing automation project documented by a machine tool builder's casebook. This example FMS will be used throughout this paper to illustrate our approach of FMS modeling.

The FMS model, as shown in Fig. 4, is composed of one cell controller, seven computer numerical control (CNC) machines, two load/unload (L/UL) stations, one work-in-process (WIP) rack, 20 load/unload buffer stations, one rail guided vehicle (RGV), and two automated guided vehicles (AGVs).

Parts enter and exit the system at the L/UL area and move to/from machines on AGV. Pallet-fixture combos are used to carry the parts while the parts are moved by the AGV or are waiting for processing in buffers. There are buffer stations in front of the L/UL area and each CNC machine, which hold incoming and outgoing parts waiting for AGV to pick up or for being processed by the CNC machine or operator. An operator is needed at the L/UL area to inspect the parts and to perform the load/unload operations.

Machines 1–4 represent four identical Milwaukee-Matic 800's (MM8), Machine 5 and Machine 6 are two identical Horizontal Head Changers (HHC), and Machine 7 is a Special Finish Boring Station (SFBS). Parts

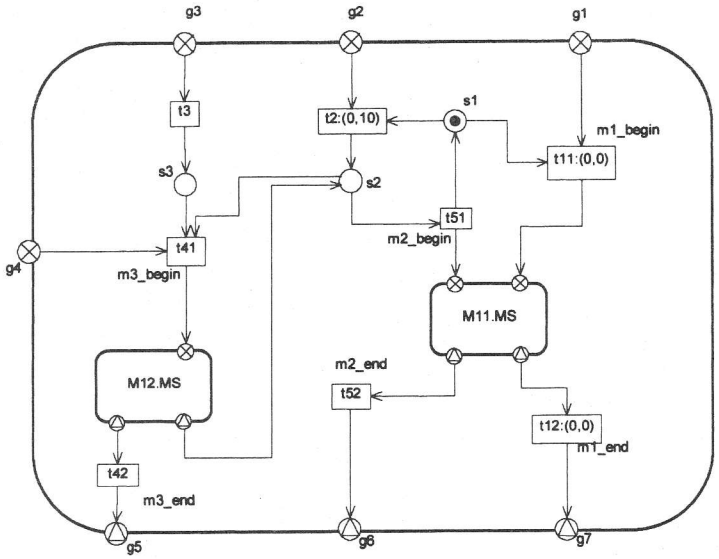

Fig. 3. An example of a module design (MD).

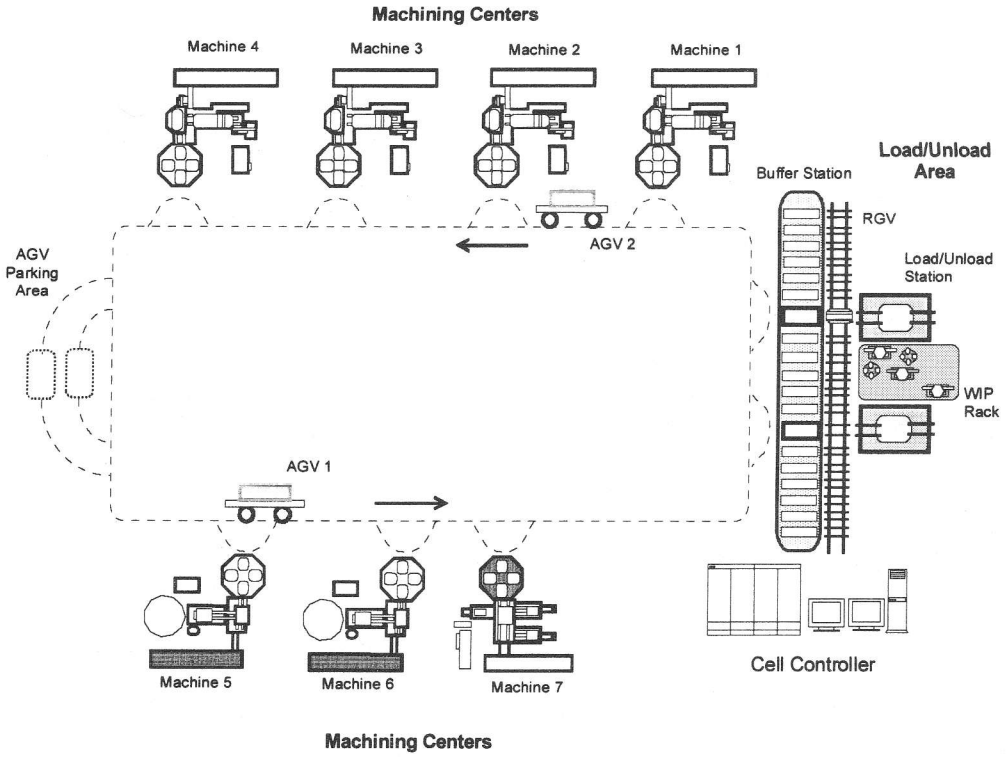

Fig. 4. A simple but typical FMS.

can be processed by any identical machines for the same operations. There are buffer stations in front of each CNC machine. A machine buffer has space for four workpieces, where only three of them can be occupied at the same time, and one is always reserved for the active workpiece.

In the load/unload area, the part with the highest priority is loaded with a pallet-fixture combo by operator on the L/UL station. A loaded workpiece is then moved to an empty buffer station and awaits an AGV coming to pick it up. The WIP rack acts as a buffer to temporarily store the unloaded workpieces. In other words, after a workpiece was unloaded by the operator, the part will be moved to the WIP rack if this part is not ready for the next operation. Meanwhile the unoccupied pallet-fixture combo will be moved to an empty buffer station. Parts, stored in the WIP rack, are waiting for the next available pallet-fixture combo in order to have the next operation. The buffer stations are used to store the workpieces (loaded with a pallet-fixture combo) and the pallet-fixture combos. Two of the buffer stations are for the active stand, where AGV can pick up or drop off the workpiece. There is one RGV running on a bi-directional rail. It provides transportation and shuttle for workpieces and pallet-fixture combo between load/unload stations and buffer stations.

The idle AGV is directed by the cell controller to pick up a workpiece with the highest priority and brings it to its destination. AGVs have to inform the cell controller and wait for next instruction after completion of a job. If there is no incoming job waiting for an AGV, the AGV will return to its parking area.

This system is controlled by a cell controller, which is the most important component in a FMS. Cell controller, acting as a commander, coordinates all the movements of individual objects based on certain scheduling policy.

For the sake of clarity, we limit the number of raw parts to four types. As shown in Table 1, there are four types (A, B, C, and D) of parts, each part may need more than one operation. There are four types (W, X, Y and Z) of pallet-fixture combos, where parts have to be loaded with corresponding types of pallet-fixture combo, specified in Table 1, for the specific operation. For example, the first operation of type A part is loading with type W pallet-fixture combo. This operation may take 300–400 time units. The second operation is machining by one of the four identical CNC machines, and this may take 2100 time units. After the second operation, type A part will be back to load/unload station, and unloaded and reloaded with type X pallet-fixture combo by the operator. The next operation (operation 5) is machining by a CNC machine M5 or M6. Operations 6 and 7 will have the part unloaded and reloaded with type Y pallet-fixture combo, and operation 8 will have the part machined by CNC machine M7. Finally, type A part will be unloaded by an operator, and this is the final operation.

Table 1
Parts routing and processing time information

Operation #	Processing time (second)	Pallet combo type	Operation
Part type: A			
1	300–400	W	Load
2	2100		M1/M2/M3/M4
3	100–200		Unload
4	300–400	X	Reload
5	1500		M5/M6
6	100–200		Unload
7	300–400	Y	Reload
8	1200		M7
9	240–360		Unload
Part type: B			
1	300–400	W	Load
2	2400		M1/M2/M3/M4
3	100–200		Unload
4	300–400	X	Reload
5	1800		M5/M6
6	100–200		Unload
7	300–400	Y	Reload
8	2100		M1/M2/M3/M4
9	100–200		Unload
10	300–400	Z	Reload
11	1500		M7
12	240–360		Unload
Part type: C			
1	300–400	W	Load
2	2100		M1/M2/M3/M4
3	100–200		Unload
4	300–400	X	Reload
5	2100		M1/M2/M3/M4
6	100–200		Unload
7	300–400	Y	Reload
8	1500		M5/M6
9	100–200		Unload
10	300–400	Z	Reload
11	1200		M7
12	240–360		Unload
Part type: D			
1	300–400	W	Load
2	1500		M5/M6
3	100–200		Unload
4	300–400	X	Reload
5	1800		M1/M2/M3/M4
6	100–200		Unload
7	300–400	Y	Reload
8	1500		M1/M2/M3/M4
9	100–200		Unload
10	300–400	Z	Reload
11	1200		M5/M6
12	240–360		Unload

5. Architecture-driven FMS modeling based on NOAM

We present in this section a systematic method of using NOAM for FMS design driven by the decomposition and refinement of system architecture. The example FMS described earlier is used as a case study to demonstrate the applicability of our approach. The method supports an incremental process to construct a complex FMS model from informal to formal, from high-level to detailed representation, thus significantly reducing the complexity of modeling. It is shown that, by basing our method on the decomposition and refinement of system architecture, our approach closely assembles practical design, and can be easily integrated into a practical system design process. At each stage, our method creates an FMS design that is complete, executable and analyzable. It is also shown that the NOAM interface structure effectively encapsulates the internal details in component design, thus helping to isolate the effect of changes local to the component and supports the exploration of design alternatives.

5.1. Modeling methodology

Mathematical-based models are normally hard to understand and use in large-scale system design in part because of the lack of association between the abstract notation and practical design concepts, and the lack of guidance as how to apply the formal representation in the design process. The modeling methodology outlined below addresses the problems by providing an incremental process for constructing complex formal design, and a smooth transition from informal to formal design. By introducing the concept and framework of system structure in NOAM, we lay a foundation for a systematic approach to FMS design that takes advantage of the rigor and analyzability of formal notation. In the meantime, we provide a way to control the complexity by applying the principle of *separation of concerns* to help designer to establish a clear focus at each step of the design and to minimize distraction from other design issues. The NOAM design method consists of the following steps:

1. *Define system interface.* This step defines the interface of interaction between the system (note: the term *system* is used to denote an entire system, a subsystem, or a system module depending on the stage of design) and its environment in terms of *input/output ports*. As part of the specification, the type and format of data that can be channeled through each port are defined in terms of token format. Once the system interface is defined, it serves as the sole communication channel to/from the system. Informal description about the ports can be used at this step.
2. *Design a system architecture that matches with the interface.* This step corresponds to top-level architectural design consistent with the system interface defined in the last step. It decomposes the system into sub-systems (or components) and the interactions between them. For each sub-system, its interface is defined. The system architecture is specified in terms of time-dependent connections and synchronization between the sub-systems' interfaces described using

Time Petri nets notation. Informal notation maybe used at this step.

3. *Construct MS for every component created in the architecture.* While the primary concern of Step 2 was the interaction and synchronization between the sub-systems, this step is to define the behavior of the subsystems consistent with their interface specification. In other words, a module specification (MS) for each sub-system is constructed. Again, informal notation may be used. Upon the completion of this step, a complete (but may be informal) architectural model of the system is produced.
4. *Substitute informal representation with formal notation.* At this step, any informal description used in Steps 1–3 is transformed into a formal one based on the formal NOAM notation. Upon the completion of this step, we now have a complete, executable and analyzable formal model that represents the top-level design of the system.
5. *Conduct model analysis or evaluation (optional).* Formal verification, simulation and/or prototyping can be performed against the architectural model produced in Step 4, if desired.
6. *Create sub-architectures.* At this step, the module specification (MS) of each sub-system is examined. If a sub-system is too large for direct implementation, use the corresponding MS as the basis (which describes the interface and required behavior for the sub-system), and repeat Steps 1–5 to develop a sub-architecture for the sub-system. If necessary, more than one sub-architectures that confirms to the same interface and behavioral requirements can be developed, and plugged in the position of the MS to evaluate different design alternatives (this maybe necessary for certain critical components, e.g., Cell Controller of an FMS).

In the above procedure, the informal steps concern about the framework and communication structure of system design, and allows designer to focus on the global control structure. The formal modeling step provides a transition to translate such a design into the formal definition of NOAM.

5.2. Systematic modeling of FMS: a case study

In this section, we use the example FMS described in Section 4 to illustrate how to apply the NOAM modeling method to support incremental design of FMS.

5.2.1. Define top-level system interface

At this first step of design, we consider the entire FMS as a single component. As the part process plans and raw parts are sent to this system, the system outputs the process reports and finished parts after a certain time. We model the FMS as an object, which has an interface with four input/output ports. Fig. 5 shows this interface and Table 2 details the descriptions for the interface.

5.2.2. Informal model of top-level architecture

We first decompose the FMS into four critical components: *Cell Controller*, *AGVs*, *CNC* Machines and *L/UL system*. Cell Controller describes the cell controller; AGVs represent all of the automated guided vehicles; L/UL system represents all the components in load/unload area, which may include load/unload stations, WIP rack, RGV, load/unload buffer stations and operators; and CNC Machines include all the CNC machines. The relationship between the components is shown in Fig. 6. The FMS is controlled by the Cell Controller which coordinates all the movements of individual components. AGVs, CNC Machines and L/UL system receive operating instructions from Cell Controller and report their current status to Cell Controller. L/UL Area and CNC Machines must coordinate with AGVs for the transportation of workpieces.

The block diagram of Fig. 6 can be translated into an informal version of NOAM design incorporating the interface specification of Fig. 5. The resulting top-level

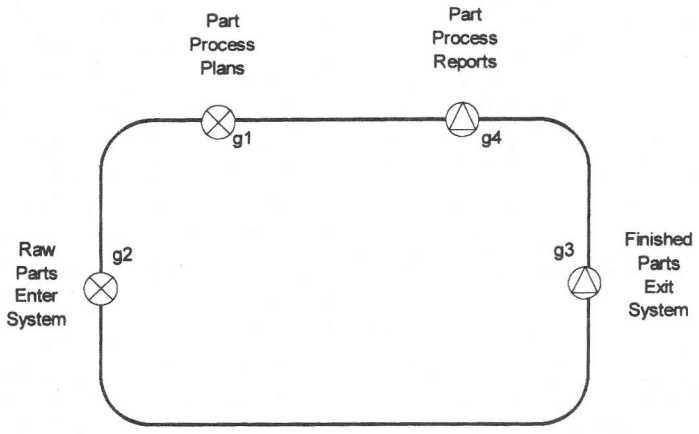

Fig. 5. The FMS interface.

Table 2
The interface descriptions of the FMS

Ports	Type	Interpretations
g_1	Input	New process plans received
g_2	Input	Raw parts enter the system
g_3	Output	Finished parts exit the system
g_4	Output	Current process reports are sent to the supervisor

module design (MD) is shown in Fig. 7. In this design, the Cell Controller sends process instruction to each of the other components (e.g. L/UL Area, AGVs and CNC Machines), which in turn report their processing status to Cell Controller. There are interfaces between AGVs and L/UL Area, and between AGVs and CNC Machines. These interfaces model the transfer of parts between different components.

An additional task of this step is to adequately specify the interfaces for each newly identified component or sub-system (i.e., Cell Controller, L/UL Area, AGVs and CNC Machines), which are discussed in the next subsection.

5.2.3. Module specification of the components

Having decided the overall system structure and communication pattern, we now proceed to module specifications (MS) of the newly identified components. Because of space constraint, only the MSs for Cell Controller and CNC machines are provided here.

A. Module specification of Cell Controller.

The communication interface of Cell Controller described by input/output ports are specified in the top-level design. A more detailed description of these ports are provided in Table 3.

Based on this interface specification, a module specification of the Cell Controller that describes its functions and behavior can be constructed. Because the main purpose of this paper is to investigate modeling framework and methodology, we will not provide detailed discussion about how to construct the time Petri net inside the MS describing the state transition and behavior of the component, which is achieved using standard Petri net techniques. The MS is shown in Fig. 8. The meanings of the transitions (representing actions or events) and places (representing states or conditions) in the MS are informally described in Tables 4 and 5, respectively.

In the specification of the Cell Controller, inhibitor arcs are used to describe the priorities of the transitions, which share the same resources. For example, transition t_7 can be enabled and fired only if there is no token in place p_3, because t_8 has a higher priority than t_7. Moreover, t_6 can be enabled and fired only if there is no token in places p_2 and p_3, this indicates t_6 has lower priority than t_7 and t_8. We also observe that raw parts can be loaded if there is no WIP waiting to be loaded/unloaded and parts in WIP rack can be loaded if there is no WIP in the L/UL buffer station. This scheme controls the quantity of WIP and provides decision information on load/unload scheduling.

B. Module specification of CNC Machines.

As another example of module specification, the MS for CNC Machines is similarly constructed based on its interface definition and the required functions. Here we simply list the result of the specification. Table 6 describes the CNC machines' communication ports, Tables 7 and 8 describe the transitions and places of the MS.

The MS shown in Fig. 9 describes a brief work cycle of the CNC machines. A workpiece is delivered to machining center by an AGV (g_2) and moved (t_4) to machine buffer station (p_4). If the machine is idle (p_1) and a machining instruction is received (g_7), a workpiece is moved to machining position (t_1) and waits to be serviced (p_2). A processed workpiece is then moved to machine buffer (t_3) and waits to be delivered back to L/UL area by AGV (p_6).

5.2.4. Formalizing top-level design

Up to this point, we have incrementally constructed the top-level architecture of the FMS. The design is informal in the sense that the communication ports,

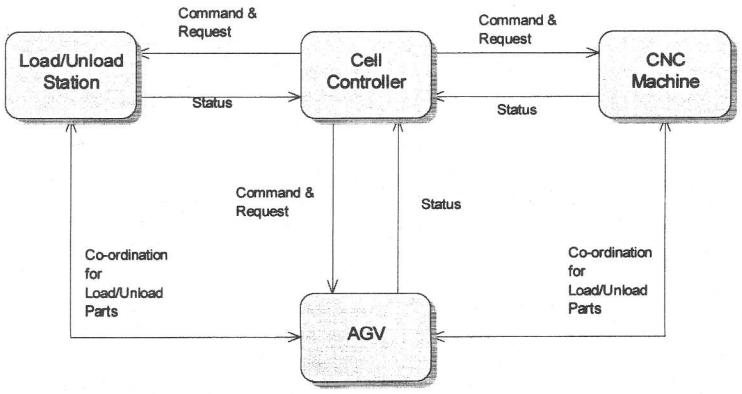

Fig. 6. Relationship of sub-systems in the FMS.

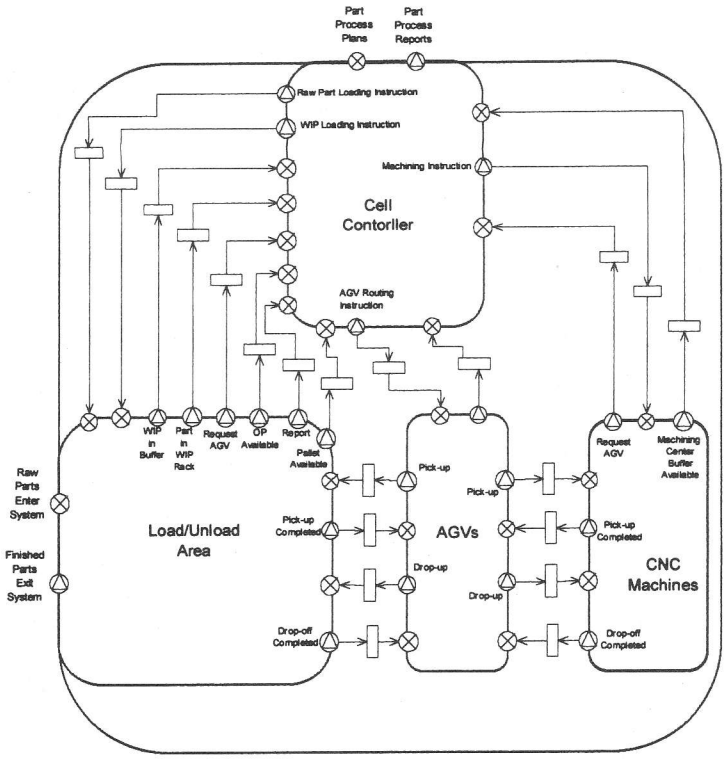

Fig. 7. Top-level module design of the FMS.

places, and transitions are described using the natural language instead of the formal notation. This informal design process is important, however, because it allows a designer to focus on the conceptualization of design without having to worry about the details of formal Petri net notation. This can also be viewed as a part of the incremental formalization process, in which we focus on structural formalization of the design, while delaying the semantic formalization, so as to reduce the burden of considering too many things at the same time, thus reducing the complexity of design at each stage.

Since the informal design already has a relatively formal structure, and provides a complete context for the model elements such as ports, places and transitions, it also makes the translation of these informally described elements much easier task. In this section, we use

Table 3
Communication ports for the Cell Controller

Ports	Interpretations
g_1	New process plans received
g_2	Send the load instruction to L/UL_Area to direct operator to perform the loading of a raw part
g_3	Send the load instruction to L/UL_Area to direct operator to perform the loading of an in-process workpiece
g_4	Send the unload instruction to L/UL_Area to direct operator to perform the unloading of an in-process workpiece
g_5	Send current completed process report
g_6	Receive a process report from L/UL_Area
g_7	Receive a message indicating that a pallet-fixture combo becomes available
g_8	Receive a message indicating that an operator becomes available
g_9	Send the routing instruction to AGV
g_{10}	Send the process instruction (NC programs etc.) to CNC_Machines
g_{11}	Receive a message indicating that an empty buffer space becomes available in machining center
g_{12}	Receive a message indicating that a WIP in the L/UL buffer station waits to be delivered
g_{13}	Receive a message indicating that an AGV becomes available
g_{14}	Receive a message indicating that a WIP in the machining center waits to be delivered
g_{15}	Receive a message indicating that a WIP in L/UL buffer station waits for unloading
g_{16}	Receive a message indicating that a part is moved to WIP rack

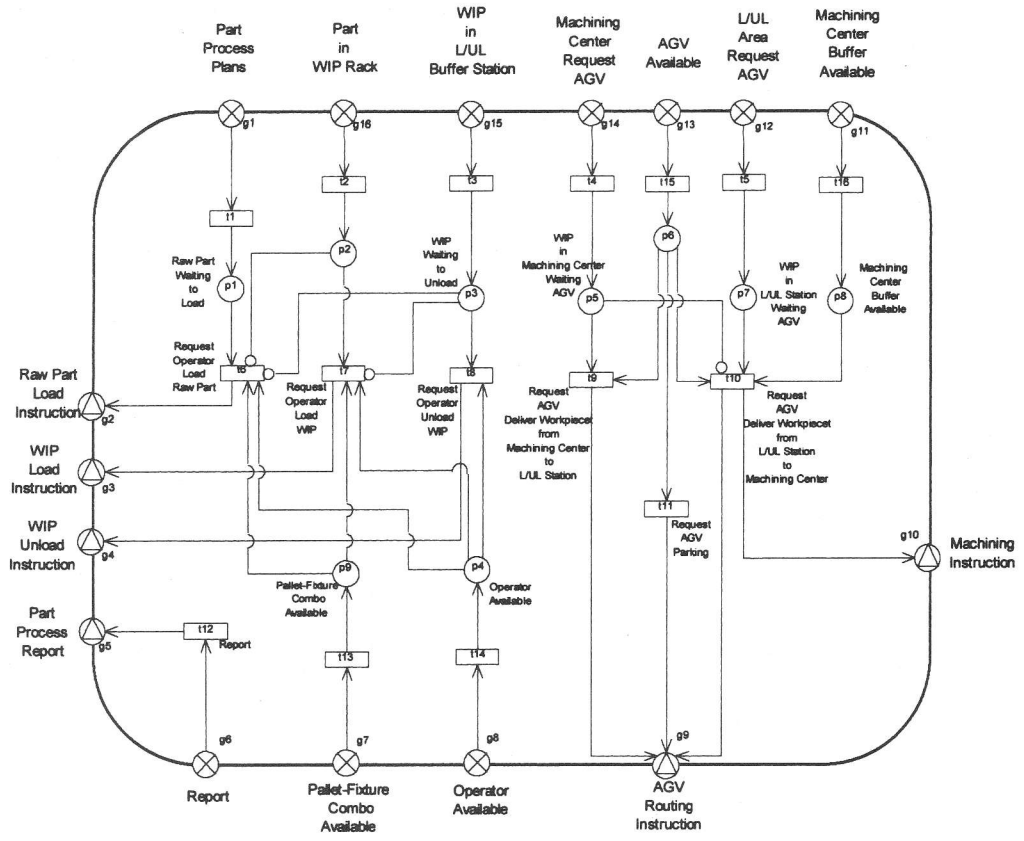

Fig. 8. Module specification of Cell Controller.

the CNC Machines as an example to show the result of formalization.

Fig. 10 denotes the same CNC Machines specification as shown in Fig. 9. The difference is that the informal description of the ports, places, transitions, and arcs are replaced with corresponding formal description as follows:

Formal definition of communication ports:

g_1: $\langle \text{Drop}_{\text{Ok}}, \text{AGVs}, \text{Machine}_{\text{Drop}_{\text{Ok}}}, (\text{AGV}_{\text{ID}}) \rangle$
g_2: $\langle \text{Drop-off}, (\text{AGV}_{\text{ID}}, \text{Part}_{\text{ID}}, \text{Pallet}_{\text{Type}}) \rangle$
g_3: $\langle \text{Pick-up}, (\text{AGV}_{\text{ID}}, \text{Part}_{\text{ID}}) \rangle$
g_4: $\langle \text{Pick}_{\text{Ok}}, \text{AGVs}, \text{Machine}_{\text{Pick}_{\text{Ok}}}, (\text{AGV}_{\text{ID}}, Part_{\text{ID}}, Operation\#, Pallet_{\text{Type}}) \rangle$
g_5: $\langle \text{Buffer}_{\text{Available}}, \text{Cell}_{\text{Controller}}, \text{MB}_{\text{Available}}, (Machine_{\text{ID}}) \rangle$
g_6: $\langle \text{Req}_{\text{AGV}}, \text{Cell}_{\text{Controller}}, \text{M}_{\text{Req}_{\text{AGV}}}, (Part_{\text{ID}}, Machine_{\text{ID}}, Operation\#) \rangle$
g_7: $\langle \text{M}_{\text{Instruction}}, (\text{Part}_{\text{ID}}, \text{Machine}_{\text{ID}}, \text{Operation\#}) \rangle$

Take input port g_2 as an example. The definition $g_2 = \langle \text{Drop-off}, (AGV_{ID}, \text{Part}_{\text{ID}}, \text{Pallet}_{\text{Type}}) \rangle$ means that the type of operation represented by the port is Drop-off, the input data required by the operation has the structure (*AGV_ID*, *Part_ID*, Pallet_Type). Similarly, output port g_1 represents an outgoing flow that acknowledges that the Drop-off operation has been successfully completed. Notice that there is no need to explicitly specify in the formal definition whether the port is an input or output port, which is implicitly determined by the direction of its adjacent arcs.

Formal definition of transitions:

t_1: $\tau(t_1) = (60, 90)$;
$e(t_1)$ $= (g_7.\text{Machine\_ID} = = p_1.\text{Machine\_ID})$ $(g_7.\text{Part\_ID} = = p_4.\text{Part\_ID})$;
t_2: $\tau(t_2) = (1200, 2400)$;
t_3: $\tau(t_3) = (60, 90)$;
t_4: $\tau(t_4) = (60, 90)$;
t_5: $\tau(t_5) = (60, 90)$;
$e(t_5)$ $= (g_3.\text{Part\_ID} == p_6.\text{Part\_ID})$.

The definition of t_1 indicates, for instance, that the action (load workpiece into machine) must take place within 60–90 time units from the moment the operation is enabled; and the condition (guard) for the action to take place is that the requested machine is available (g_7.Machine_ID = = p_1.Machine_ID) and the part to be processed is waiting in the buffer (g_7.Part_ID = = p_4.Part_ID). Combined, the definition specifies not only the timing but also the logical

Table 4
Transitions of the Cell_Controller

Transitions	Interpretations
t_1–t_5	Calculate the priorities of incoming jobs based on a dispatching rule
t_6	Direct operator to load a raw part with the highest priority
t_7	Direct operator to load a WIP with the highest priority
t_8	Direct operator to unload a WIP with the highest priority
t_9	Request AGV to deliver a workpiece with the highest priority from machining center to L/UL buffer station
t_{10}	Request AGV to deliver a workpiece with the highest priority from L/UL buffer station to machining center
t_{11}	Request AGV to return to its parking area
t_{12}	Create process report

Table 5
Places of the Cell Controller

Places	Interpretations
p_1	Raw parts wait for loading
p_2	Parts in WIP rack waiting for loading
p_3	Workpieces in L/UL buffer station waiting for unloading
p_4	Operators available to perform the load/unload operation
p_5	Parts in machining center waiting to be delivered
p_6	AGVs available
p_7	Parts in L/UL buffer station wait to be delivered
p_8	Empty buffer spaces available in the machining center
p_9	Pallet-fixture combos available

Table 6
Communication ports of the CNC_Machines

Ports	Interpretations
g_1	Inform AGV that the transport of workpiece is completed
g_2	Receive a message indicating that AGV has arrived to drop off part
g_3	Receive a message indicating that AGV has arrived to pick up part
g_4	Inform AGV that the transport of workpiece is completed
g_5	Inform Cell Controller that an empty buffer space becomes available
g_6	Inform Cell Controller that a workpiece waits to be transported
g_7	Receive workpiece machining instruction

Table 7
Transitions in the MS of the CNC_Machines

Transitions	Interpretations
t_1	Part is moved into the machining position from machine buffer
t_2	Part is processed by CNC machine
t_3	Processed part is moved to buffer from the machining position
t_4	Part is moved from AGV to buffer station
t_5	Part is moved from buffer station to AGV

requirements for the action. Furthermore, the definition of arcs, shown below, precisely specifies the input and output flow for every operation defined by the transitions.

Table 8
Places in the MS of the CNC_Machines

Places	Interpretations
p_1	CNC machines available for next job
p_2	Parts ready to be processed by CNC machine
p_3	The processed parts waiting to be moved to a buffer station
p_4	Parts waiting to be processed
p_5	Transporter is idle
p_6	Parts waiting to be transported to the L/UL area

Formal definition of arcs:

a_1: (AGV_ID, Part_ID)
a_2: (AGV_ID)
a_3: (Part_ID, Pallet_Type)
a_4: ε
a_5: (Part_ID, Pallet_Type)
a_6: (Part_ID, Machine_ID, Operation#)
a_7: (Machine_ID)
a_8: (Part_ID, Machine_ID, Operation#, Pallet_Type)
a_9: ε
a_{10}: (Part_ID, Machine_ID, Operation#, Pallet_Type)
a_{11}: (Part_ID, Machine_ID, Operation#, Pallet_Type)
a_{12}: (Part_ID, Machine_ID, Operation#, Pallet_Type)
a_{13}: ε
a_{14} (Machine_ID)
a_{15}: (Part_ID, Machine_ID, Operation#)
a_{16}: (Part_ID, Operation#, Pallet_Type)
a_{17}: (Part_ID, Operation#, Pallet_Type)
a_{18}: ε
a_{19}: (AGV_ID, Part_ID)
a_{20}: (AGV_ID, Part_ID, Operation#, Pallet_Type)
a_{21}: (Machine_ID)

From the above discussion, it can be seen that generating formal specification of the design is reasonably straightforward given the informal design. A major reason for this is due to the fact that the global control structure, the data/control flow, and the context for each operation has been explicitly defined and specified. Given

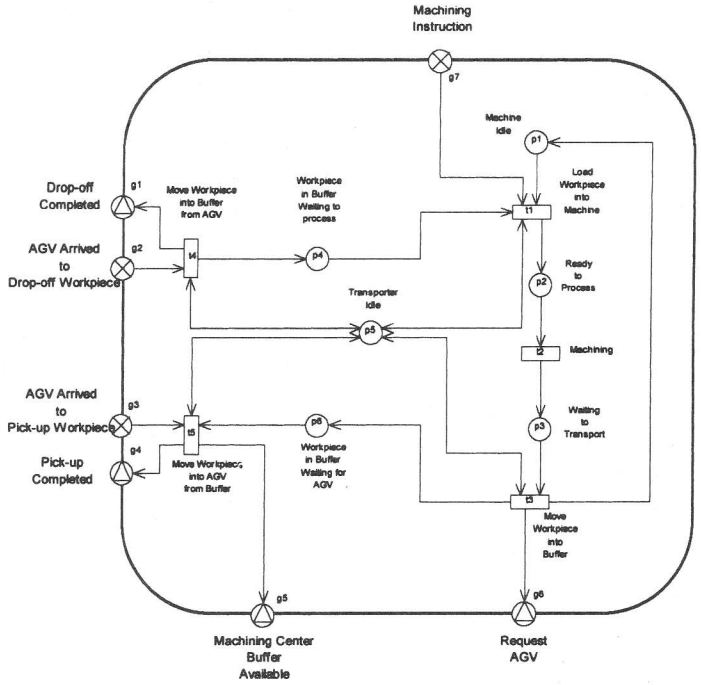

Fig. 9. Module specification of the CNC Machines.

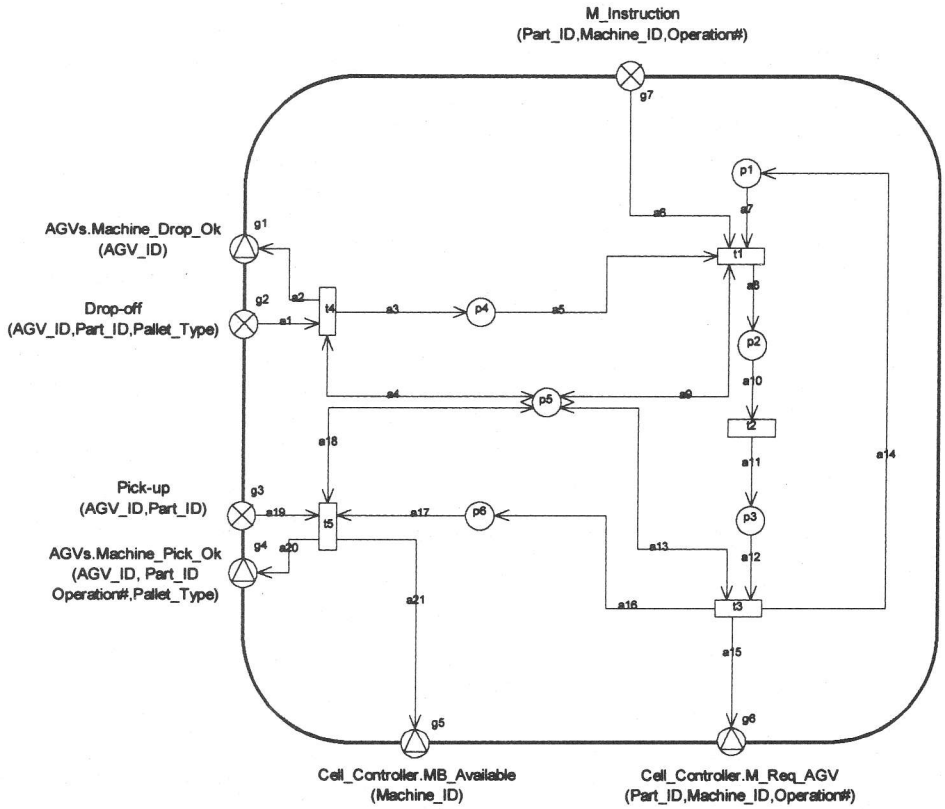

Fig. 10. Formal module specification (MS) of CNC Machines.

this information, a designer can fully concentrate on enforcing the preciseness and consistency of the design.

Upon the completion of this formalization step, we have a complete, both in terms of structure and semantics, architectural model of the FMS regardless of whether further decomposition or refinement is needed. This model is executable, and thus can be used as a system prototype for experimental evaluation. It is formally verifiable using any existing Petri net verification techniques. This gives us an opportunity and tools to incrementally evaluate each stage of the design process. This feature not only reduces the complexity of analysis, but also allows us to promptly detect and correct design faults as they occur, thus preventing them from propagating to later design or implementation stages where correction of the same errors are much more difficult and costly.

5.3. Discussion

We have shown how to incrementally proceed to achieve a formal design model of FMS based on NOAM. Moreover, we provide an easy way to modify, refine and decompose the design, or to explore different alternatives of design.

For instance, the design of CNC Machines shown in Fig. 10 may be too simple to describe the behavior of Transporter, which transports workpiece between AGV and machine buffer station, and between machine and machine buffer station. Fig. 11 shows a refinement of CNC Machines, where we refine the specification of the Transporter and model the dispatching rule using First-Come-First-Served (FCFS) policy. The refinement will not affect or alter the other components in the same level or the upper level because the interface of CNC Machines remains the same. This is a very important advantage in complex system modeling.

In order to meet different requirements, we may change the associated action or logical expression of a transition. For example, a transition can associate a different program to calculate the priorities of jobs based on a different scheduling rule. We may also use a function to calculate the workpiece machining time or AGV traveling time more precisely. Obviously, these modifications can be easily done, and, more importantly, they don't require any changes to our module specification or the system structure.

Another advantage of our approach is that it provides a systematic approach for subsystem decomposition. For example, we can decompose the L/UL Area into three sub-components, L/UL Buffer Station, RGV and L/UL Station, if a more detailed and precise design is needed. A corresponding module design (MD) of L/UL Area then needs to be created, which, of course, has to satisfy the behavior specified by the MS. Because the interface of the L/UL Area defines a clear boundary of the subsystem, the relationship of L/UL Area with other components will not be affected. It should be noted that

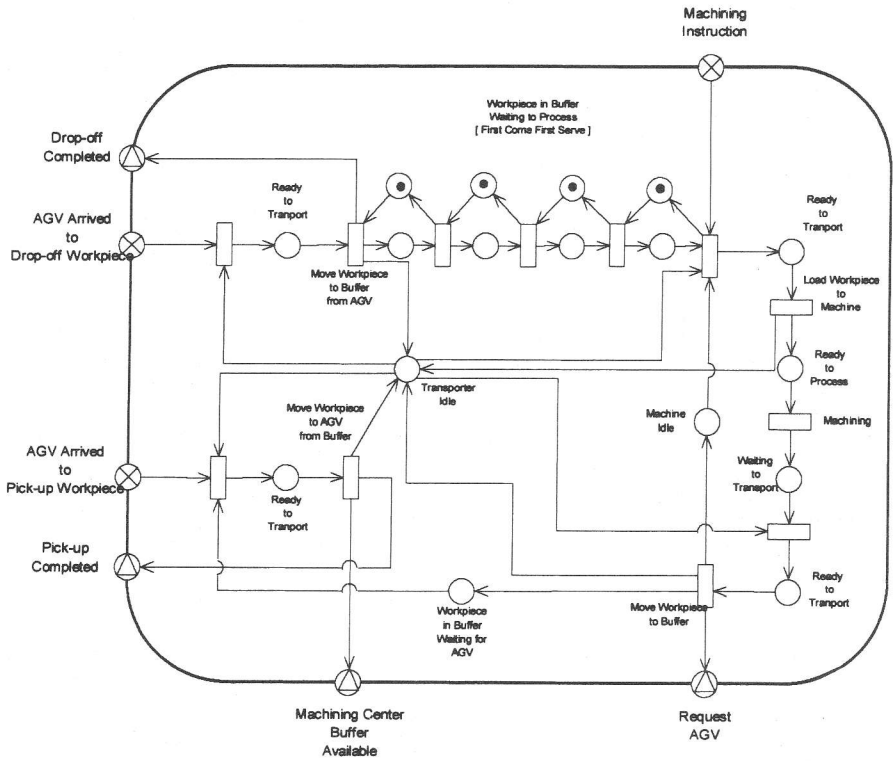

Fig. 11. Refinement of CNC_Machines.

the design of a component interface is crucial. With a careful analysis, a good interface design will minimize the effect of modification and decomposition in the overall design.

6. Conclusion

We have presented an architecture-driven formal approach based on Petri nets for the modeling and design of (FMS), which effectively addresses the problems of Petri nets as a design model of complex real-time concurrent systems. Through the case study, we have demonstrated that our approach, centered around the NOAM, provides a systematic way to incrementally construct, refine and analyze complex FMS models. It uses architectural decomposition as the basis to reduce design complexity, to provide smooth transition from informal to formal design, and to explore design alternatives. A NOAM model of FMS is much more compact, is consistent with practical system design both syntactically and semantically, and has a structure that is more readable, changeable and extensible.

Based on the framework of NOAM, we are investigating two open issues. The first is analysis techniques that allow us to compositionally verify time-critical system properties, such as real-time satisfiability and schedulability. Because NOAM allows more complex connection and synchronization structure among concurrent modules, it introduces additional complexity for compositional analysis. Second, what NOAM (and other Petri net-based formalisms) describes is the operational model of a real-time system. It does not explicitly describe the required real-time properties or constraints that the modeled system must satisfy. (One may argue that the model itself represents the requirements that the final operational system must satisfy. These requirements, however, are buried in the operational structure, and are extremely difficult to identify.) We are working to extend NOAM by integrating its Petri net-based operational notation with a logic-based descriptive notation, so that the resulting model provides a foundation to enhance and ensure the traceability of system design against real-time constraints in the design and refinement process.

References

Baldassari, M., Bruno, G., 1991. PROTOB: An object-oriented methodology for developing discrete event dynamic systems. Computer Language 16 (1), 39–63.

Bastide, R., Sibertin-Blanc, C., Palanque, P., 1993. Cooperative Objects: A concurrent Petri net-based object-oriented language. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pp. 286–291.

Battiston, E., De Cindio, F., Mauri, G., 1988. OBJSA nets: A class of high-level Petri nets having objects as domains. In: Rozemberg G. (Ed.), Advances in Petri nets '88, vol. 340, Lecture Notes on Computer Science, Springer, 1988, pp. 20–43.

Berthomieu, Diaz, M., 1991. Modeling and verification of time dependent systems using time Petri nets. IEEE Transactions on Software Engineering 17 (3), 259–273.

Brussel, H.V., Peng, Y., Vallckenaers, P., 1993. Modeling flexible manufacturing systems based on Petri nets. Annals of the CIRP 42 (1), 479–484.

Bucci, G., Vicario, E., 1995. Compositional validation of time-critical systems using communicating time Petri nets. IEEE Transactions on Software Engineering 21 (12), 969–991.

Camurri, A., Franchi, P., Gandolfo, F., Zaccaria, R., 1993. Petri net based process scheduling: a model of the control system of flexible manufacturing systems. Journal of intelligent and Robotic Systems 8, 99–123.

Deng, Y., Du, W., Attie, P.C., Evangelist, M., 1996. A formalism for architectural modeling of distributed real-time systems. Proceedings of 8th International Conference on Software Engineering and Knowledge Engineering. Lake Tahoe, Nevada, pp. 408–417.

Deng, Y., Lu, S., Evangelist, M., 1997. A formalism for Architectural modeling and prototyping of distributed real-time systems. Proceedings of the 30th Hawaii International Conference on Systems Science, Maui, Hawaii.

Du, W., Deng, Y., 1995. Formal notation of NOAM. Technical Report. School of Computer Science, Florida International University, Miami, FL.

D'souza, K.A., Khator, S.K., 1994. A survey of Petri net applications in modeling controls for automated manufacturing systems. Computers in Industry 24, 5–16.

Elmaraghy, H.A., Ravi, T., 1992. Modern tools for the design, modeling and evaluation of flexible manufacturing systems. Robotics & Computer-Integrated Manufacturing 9 (4), 335–340.

Engelfriet, J., Leih, G., Rozenberg, G., 1990. Net-based description of parallel object-based systems, or POTs and POPs. In: de Bakker, J.W., de Roever, W.P., Rozenberg, G. (Eds.), Foundations of Object-Oriented Languages, vol. 489, Lecture Notes on Computer Science, Spinger, Berlin, pp. 229–273.

Greenwood, N.R., 1988. Implementing Flexible Manufacturing Systems. Halsted Press, New York.

Genrich, H.J., Lautenbach, K., 1981. System modelling with high-level Petri nets. Journal of Theoretical Compute Science 13, 109–136.

Ghezzi, C., Mandrioli, D., Morasca, S., Pezze, M., 1991. A unified high-level Petri net formalism for time-critical systems. IEEE Trans. of Software Engineering 17 (2), 160–171.

Huang, H.P., Chang, P.C., 1992. Specification, modeling and control of a flexible manufacturing cell. International Journal of Production Research 30 (11), 2515–2543.

Jensen, K., 1992. Coloured Petri Nets, vol. 1. Springer, Berlin.

Knapp, G.M., Wang, H.P., 1992. Modeling of automated storage/retrieval systems using Petri nets. Journal of Manufacturing Systems 11 (1), 20–29.

Lee, Y.K., Park, S.J., 1993. OPNets: an object-oriented high-level Petri net model for real-time system modeling. Journal of Systems Software 20 (1), 69–86.

Lin, J.T., Lee, C.C., 1995. A CTPN-based scheduler for a flexible manufacturing cell. Journal of the Chinese Institute of Engineers 18 (5), 655–672.

Looveren, A.J., Gelders, L.F., Wassenhove, V., 1986. A review of FMS planning models. In: Andrew K. (Ed.), Modelling and Design of Flexible Manufacturing Systems. pp. 3–31.

Luckham, D.C., Kenny, J.J., Augustin, L.M., Vera, J., Bryan, D., Mann, W., 1995a. Specification and analysis of system architecture using Rapide. IEEE Transactions on Software Engineering 21 (4), 336–355.

Luckham, D.C., Vera, J., Meldal, S., 1995. Three concepts of system architecture. Technical Report. Stanford University.

Martinez, J., Alla, H., Silva, M., 1986. Petri nets for the specification of FMSs. In: Andrew K. (Ed.), Modelling and Design of Flexible Manufacturing Systems. pp. 389–406.

Meng, J., Soh, Y.C., Wang, Y., 1995. A TCPN model and deadlock avoidance for FMS jobshop scheduling and control system. IEEE International Workshop on Emerging Technologies and Factory Automation, Paris, France, pp. 521–532.

Merlin, P., Faber, D.J., 1976. Recoverability of communication protocols. IEEE Transactions on Communication COM-24 (9).

Murata, T., 1989. Petri nets: properties, analysis and applications. Proceedings of the IEEE 77 (4), 541–580.

Palaudetto, M., Raymond, S., 1993. A methodology based on objects and Petri nets for development of real-time software. Proceedings of the IEEE International Conference on systems. Man and Cybernetics 2, 701–710.

Qadri, F., Robbi, A., 1994. Timed Petri nets for flexible manufacturing cell design. IEEE International Conference on Systems, Man, and Cybernetics. Texas, pp. 1695–1699.

Shukla, C.S., Chen, F.F. The state-of-the-art in intelligent real-time FMS control: a comprehensive survey. Journal of Intelligent Manufacturing, to appear in 1997.

Solot, P., Vliet, M.V., 1994. Analytical models for FMS design optimization: A Survey. International Journal of Flexible Manufacturing Systems 6 (3), 209–233.

Wang, L.-C., 1996. The development of an object-oriented Petri net cell control model. International Journal on Advanced Manufacturing Technology 11, 59–69.

Zuberek, W.M., 1995. Schedules of flexible manufacturing cells and their timed colored Petri net models. IEEE International Conference on Systems, Man, and Cybernetics. New York, pp. 2142–2147.

Zhou, M.C., Dicesare, F., Rudolph, D.L., 1992. Design and implementation of a Petri Net based supervisor for a flexible manufacturing system. Automatica 28 (6), 1199–1208.

Zhou, M.C., McDermott, K., Patel, P.A., 1993. Petri net synthesis and analysis of a flexible manufacturing system cell. IEEE Transactions on System, Man, and Cybernetics 23 (2), 523–531.

Zhou, Q., Wang, M., Dutta, S.P., 1995. Generation of optimal control policy for flexible manufacturing cells: a Petri net approach. International Journal of Advanced Manufacturing technology 10, 59–65.

Yi Deng received the B.S. degree in Computer Science from the University of Science and Technology of China in 1983, the M.Sc., and Ph.D. degrees, both in Computer Science, from the University of Pittsburgh in 1990 and 1992, respectively. He has been on the faculty in the School of Computer Science, Florida International University – the State University of Florida in Miami, since 1992, where he is currently an Associate Professor, and directs the Advanced Distributed Systems Engineering Lab. His research interests include distributed systems, real-time systems, and software engineering, especially software architecture and formal methods. His research has been supported by NSF, USAF Rome Lab, AFOSR and NASA. Dr. Deng is an Associate Editor for the International Journal of Software Engineering and Knowledge Engineering, and co-chairs the Program Committee for the 1998 International Conference on Software Engineering and Knowledge Engineering.

Dr. Deng is a member of the Association for Computing Machinery and the IEEE Computer Society.

Chia-Rung Yang received the BA degree from the Tankang University, Taiwan, ROC, in 1991, and the M.Sc. degree from Florida International University in 1996, both in Computer Science. He is currently a Product Engineer at the Acer Latin America Inc., where he is responsible for new product development and testing, as well as transfering new products into mass production. His research interests include software engineering and system testing.