Integrated Architectural Modeling of Real-Time Concurrent Systems with Applications in FMS[1]

Yi Deng, Jiacun Wang and Rakesh Sinha
School of Computer Science
Florida International University
{deng, wangji, sinha}@cs.fiu.edu

ABSTRACT — A Real-time Architectural Specification (RAS) model and its application in the modeling of flexible manufacturing system (FMS) are presented. 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). 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. A formal engineering approach that helps handle the complexity and dynamics of FMS modeling, design and analysis is needed. RAS combines mature operational and descriptive formal methods, in particular Time Petri nets (TPN) and Real-Time Computational Tree Logic (RTCTL), to form an integrated system model for architectural specification and analysis of real-time concurrent systems such as FMS. The contribution of RAS is twofold: First, it provides a formal system to systematically maintain a strong correlation between (real-time) requirements and design and to verify the conformance of the design to the requirements, which helps enhance traceability and thus to help us to achieve high assurance in design. Second, it offers better scalability in modeling and analysis, which provides an effectively way to deal with complexity in the application of formal methods. These two features together make RAS a suitable model for the design of FMS.

Keywords: Flexible manufacturing systems, formal system design, real-time systems, time Petri nets, real-time computational tree logic.

---

1. This work was supported in part by the NSF under Grant No. HDR-9707076, by Air Force Office of Scientific Research under Grant No. F49620-96-1-0221, and by NASA under Grand No. NAGW-4080. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official polices or endorsements either expressed or implied by the above named Agencies.

1. Introduction

Flexible manufacturing systems (FMS) 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). 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. Given the complexity of the FMS design, an *ad hoc* method is clearly inadequate and a more rigorous approach addressing the complexity and dynamics in FMS modeling, design and analysis is needed [11].

In recent years, formal techniques such as Petri nets have been increasingly used in FMS modeling (see Section 2). As a popular modeling tool for concurrent and distributed systems, Petri nets provide a rigorous and operational way to describe and analyze system properties. In addition to its rigor and analytic capability, 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. These strengths make Petri nets a powerful modeling tool for FMS.

While offering many advantages, however, ordinary Petri net models suffer from some problems that limit their usability and application as a design model for complex FMS. Petri net-based models tend to become too large even for a modest-sized problem [21]. The primary concern of Petri nets, like many other formal techniques, is behavior modeling and analysis. They, however, lack of mechanisms to structure complex designs in such a way that both helps enforce design integrity and provide a systematic and incremental way for design and analysis. For example, in an FMS, 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. Most of Petri net-based models do not provide explicit or adequate support to this task.

In this paper, we go beyond conventional Petri net modeling and present an integrated formal method to support scalable and evolutionary design of real-time concurrent systems from the perspective of time-dependent architectural modeling and analysis. Our approach is based on two basic hypotheses: (1) The ability to systematically maintain a strong correlation between requirement and design at every design level is the basis to achieve quality design. An effective way to achieve such strong design traceability is to incorporate (real-time) requirement constraints as an integral part of design modeling. (2) Constraint-driven compositional analysis sensibly integrated as a part of design process is a powerful means to control complexity and cost in analysis. Based on these principles, we have developed the Real-time Architectural Specification (RAS) model [9]. RAS is built on top of Time Petri nets (TPN) [4] and Real-Time Computational Tree Logic (RTCTL) [12]. Petri nets are a well-known operational model best suited for modeling the control of distributed systems but cumbersome for specifying rules and constraints. By contrast, temporal logic, a popular descriptive formalism, is best suited for describing rules and constraints but not control and composition of systems. By integrating them into one coherent architectural model, RAS establishes two desirable features: First, it provides a formal system to systematically maintain a strong correlation between (real-time) requirements and design and to verify the conformance of the design to the requirements. Second, it offers better scalability in modeling and analysis and provides an effective way to deal with complexity in the application of formal methods. These two features together make RAS a suitable model for the design of FMS.

The rest of the paper is arranged as following: Related work is discussed in Section 2. An overview of RAS model, including an example and formal definition, is given in Section 3. In section 4, we illustrate the use of RAS to give an incremental modeling of an FMS. Finally, in Section 5, we briefly describe an incremental timing analysis technique based on the RAS model. The formal description of RAS model and the rules for the use of RAS model are given in Appendix I and II, respectively.

2. Related 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, and reliability measures and deadlock avoidance for FMS. The details of these applications can be found in surveys in [10, 24]. Some typical uses of Petri nets in FMS modeling can be found in [5, 7, 13, 14, 16, 20, 23, 26-28].

Those studies deal with the issue of how to use Petri nets to address specific modeling and analysis problems in FMS. We address a different issue. Our goal is to develop an engineering practice to systematically and cost-effectively apply Petri net theory in complex FMS modeling, design and analysis.

Related to our approach, several structural Petri net models are proposed both to provide a mechanism for system composition and to manage complexity in modeling. These include PROTOB [1], OBJSA nets [3], the Cooperative Objects Language [2], and OPNets [12]. An application of OPNets in FMS modeling and analysis is presented in [27], where they are used to represent part of an object-oriented Petri net cell control model. Each of these models provides certain object-based structure for system composition. However, none of these models has a formal semantics for modeling timing and timed behavior. A more recent model, Communicating Time Petri Nets (CmTPN) [6], has a formal semantics about time, and supports reachability-based compositional verification.

On the opposite side of Petri nets are various logic-based models. Logic provides a more abstract approach to the description and analysis of FMS. In temporal logic, various temporal operators are provided to describe and reason about how the truth-value of assertions varies over time. It has proven to be a very useful formalism for reactive systems [22]. RTCTL [12], a real time propositional branching time logic, has been a popular choice for describing real time systems like FMS. It allows us to express various desired types of behavior, including safety, liveness, and bounded-fairness.

As an operational model, Petri nets (with structural extensions) are well suited to model FMS as abstract programs, which can be formulated as the parallel composition of subsystems. However, it is cumbersome for describing system requirements. As a descriptive model, temporal logic is appropriate for specifying rules and constraints. However, it does not reflect the component/interaction view relevant for design level specification of FMS. So, when used alone, neither of these two methods is sufficient for architecture modeling of an FMS. In Mandrioli *et al.* [19], the advantage of using the TRIO logic and TPN together for system specification and verification are suggested and explored. Unlike our work on RAS, however, it is unclear to us what the concrete objective of the suggested integration would be. Moreover, a formal framework was yet to be developed to show how to integrate TRIO with Petri nets.

3. Overview of RAS Model

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 simultaneously maintaining the integrity of design and minimizing the affect of such changes to the overall design. The RAS model was motivated by these concerns. RAS models a distributed system as a multi-leveled composition of components and real-time constraints that the components and their compositions must satisfy at every design level. A strong correlation between system constraints and design is maintained at every level.

An RAS *model* consists of three basic elements: *operational component models*, *connections*, and *architectural constraints* (constraints in brief). The operational component models describe the real-time behavior and communication interface of the components or subsystems; the connections specify how the components interact with each other and, in turn, form an *operational system composition model*; and finally, the constraints define real-time system requirements imposed on the components and connections. All connections are defined using only communication interfaces, which gives us the flexibility to change the design of individual components without a need to void the analysis of the entire system.

Figure 1 shows an example of a two-level RAS model. The top-level design has three components – *A*, *B*, and *C*. Component *C* is further refined at the lower level design into the composition of components *E*, *F*, and *G*. The design at any level must satisfy the constraints specified at that level. The operational component model has two parts: (1) *communication ports* (denoted graphically by half circles), including *input ports* (e.g., *port*6) and *output ports* (e.g., *port*7), and (2) a TPN that describes the time-dependent, operational behavior of the component, that is, it defines the semantics associated with the ports. The communication between a component and its environment is solely through the ports. A connection represents a channel of interaction between components. It is modeled by a simple TPN and defines the direction of message flow and delay in the channel. For example, components *A* and *B* have a request-reply relationship that is modeled by the bi-directional channel. The basic communication model supported by RAS is asynchronous message passing; that is, the sending component does not suspend itself if there is a concurrent activity available. At any given design level, ports can also be divided into *external ports* (e.g. *port*1 in the high level design, *port*6 and *port*7 in the low level design) and *internal ports* (e.g. *port*3 and *port*4). An internal port is defined to communicate with other components, while an external port is used to describe the inputs and outputs from and to the environment of the system.

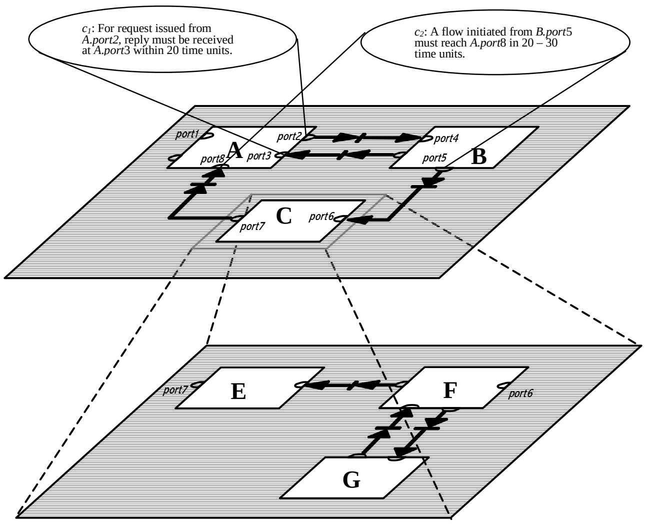

Figure 1 Framework of the RAS model

In addition to serving as a component's communication interface, the ports also provide the *linkage* between the operational design (components and connections) and the descriptive architectural constraints. The resource constraints of the design are modeled by Petri nets (marked places). The timing constraints are specified by RTCTL formulas defined over ports. Each port represents an atomic proposition, which is satisfied by an arriving token (an incoming message at the port). These ports constitute the alphabet (atomic propositions) of the RTCTL formulas that define the constraints. All constraints are specified using ports only, no internal information about the components is revealed. For example, constraint c_1 limits the time taken in a request-reply interaction with *B*, and constraint c_2 specifies the required synchronization between components *A*, *B*, and *C*. The set of constraints can be logically divided into three classes: *component constraints*, *environmental constraints*, and *path constraints*. A component constraint describes the timing properties of a component that its environment expects from it, an environmental constraint describes the timing properties that a component expects from its environment, and a path constraint describes the timing constraint for message transmission among components.

4. Incremental Modeling of an FMS

In this section, we illustrate the use of RAS to model a hypothetical FMS. This will further explain the RAS framework as well as illustrate the benefits of using the RAS framework.

4.1 Overview of the System

The manufacturing system is composed of three subsystems: processing, checking, and repairing subsystems. These three subsystems run concurrently. The processing subsystem is composed of 5 machines, indicated as M1-5. Two types of workpieces, indicated as W1 and W2, are processed and then assembled into a new one, indicated as W3. The processing flow is shown in Figure 2. That is, W1 and W2 are first processed by M1 and M2, which results in $W1^{(1)}$ and $W2^{(1)}$, respectively. Next, $W1^{(1)}$ is processed by either M3 or M5, which results in $W1^{(2)}$, then by either M4 or M6, which results in $W1^{(3)}$; W2 is processed by either M3 or M4, which results in $W2^{(3)}$, then by either M5 or M6, which results in $W2^{(3)}$. Finally, $W1^{(3)}$ and $W2^{(3)}$ are further assembled by M7 into W3. W3 is the product of the processing subsystem.

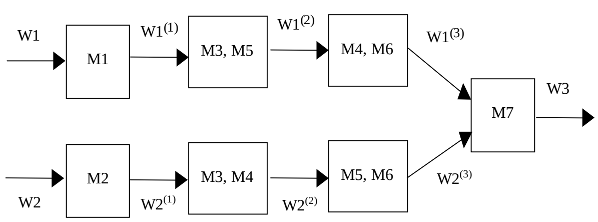

Figure 2. The processing flow of the processing subsystems.

After receiving a product from the processing subsystem, the checking subsystem examines whether the product satisfies quality control conditions. If it does, then the product will be sealed and output as the final product of the whole system. If it doesn't, it will be sent to the repairing subsystem for repair. When the repairing activity is finished, it will be returned to the checking subsystem and sealed.

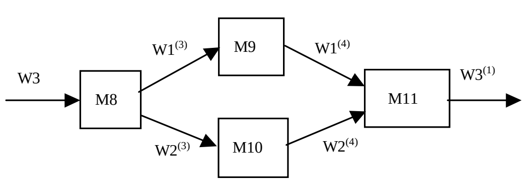

Figure 3. The processing flow of the repairing subsystems.

The repairing subsystem is composed of 4 machines, indicated by M8-11. An unqualified product W3 of the processing subsystem is first disassembled by M8 into two workpieces, $W1^{(3)}$ and $W2^{(3)}$. $W1^{(3)}$ and $W2^{(3)}$ are processed by M9 and M10, which results in $W1^{(4)}$ and $W2^{(4)}$, respectively. Then, $W1^{(4)}$ and $W2^{(4)}$ are assembled by M11 into $W3^{(1)}$, which is then returned to the checking subsystem.

In addition, there are two vehicles, vehicle 1 and vehicle 2, responsible for the transfer of products between processing and checking subsystems and between checking and repairing subsystems, respectively.

The requirements for this FMS design include:

(1) The processing time of the processing subsystem for each pair of workpieces is not longer than 25 time units.

(2) The processing time of the checking subsystem for each product of the processing subsystem is not longer than 12 time units.

(3) After the checking subsystem sends an unqualified product to the repairing subsystem, it must receive the repaired product within 20 time units.

All these requirements are applied in the case that no waiting time exists for any workpiece at any processing stage.

4.2 RAS Interface of the System

In order to capture the profile of the operational model of the manufacturing system, we first present the interaction interface of system's components (each component is corresponding to a subsystem, namely the processing subsystem is viewed as component PS, the checking subsystem as CS, and the repairing subsystem as RS), which is shown as in Figure 4. Table 1 and Table 2 show the meanings of all ports and connection transitions.

4.3 Architectural Constraints Imposed on the System

In this section, we give the RTCTL formulae for the three constraints of the FMS design. The first constraint, c_1, is

c_1: port1 $\rightarrow AF^{\leq 25}$ port2,

which is a component constraint and states that the processing time of the processing subsystem for each pair of workpieces is not longer than 25 time units. The second constraint, c_2, is

c_2: port3 $\rightarrow AF^{\leq 12}$ port4,

which is also a component constraint and states that the processing time of the checking subsystem for each product is not longer than 12 time units. The third constraint, c_3, is

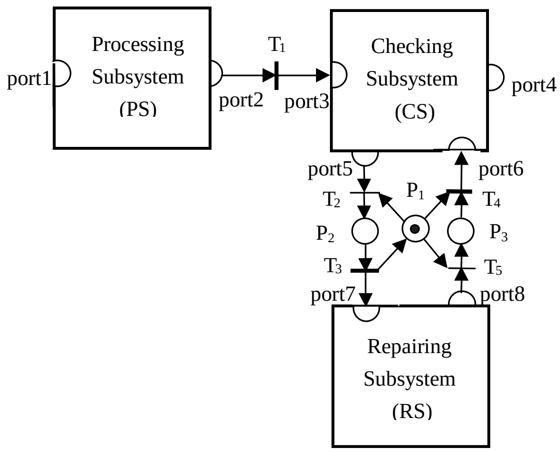

Figure 4. RAS interface of the FMS.

Table 1. Legends of ports in Figure 4.

PORT	TYPE	DESCRIPTION
port1	Input	A pair of workpieces are ready for processing
port2	Output	The processing of a product at the processing subsystem is finished
port3	Input	A product of the processing subsystem is ready for quality checking
port4	Output	A product of the whole system is available
port5	Output	An unqualified product of the processing subsystem is ready to be transferred to the repairing subsystem.
port6	Input	A repaired product is ready for processing by the checking subsystem
port7	Input	An unqualified product of the processing subsystem is ready for repairing
port8	Output	The repairing of an unqualified product is finished

Table 2. Legends of places and transitions in Figure 4.

P_Set	DESCRIPTION	
P_1	Vehicle 2 is available	
P_2	Vehicle 2 is transferring an unqualified product to the repairing system	
P_3	Vehicle 2 is transferring a repaired product back to the checking system	
T_Set	DESCRIPTION	TIME
T_1	Vehicle 1 transfers a product of the processing subsystem to the checking system	[1, 2]
T_2	Vehicle 2 begins transferring an unqualified product to the repairing system	- -
T_3	Vehicle 2 ends transferring an unqualified product to the repairing system	[1, 2]
T_4	Vehicle 2 begins transferring a repaired product back to the checking system	- -
T_5	Vehicle 2 ends transferring a repaired product back to the checking system	[1, 2]

$$c_3\text{: port5} \rightarrow AF^{\leq 20}\text{port6},$$

which is an environmental constraint and states that after the checking subsystem sends an unqualified product to the repairing subsystem, it must receive the repaired product within 20 time units.

4.4 Operational Model of Components (Subsystems)

The operational model of component CS (Checking Subsystem) is shown as in Figure 5 and the legends of places and transitions involved in this model are listed in Tables 3. Here, we use a thin bar to represent an immediate transition, and we use a thick bar to model timed transition. t32 and t33 form a random switch. We assume that the probabilities for these two transitions are 0.9 and 0.1, respectively. This information is necessary when we verify the system's constraints. Due to the space limitation, the operational models of components PS and RS are omitted here, which can be found in [9].

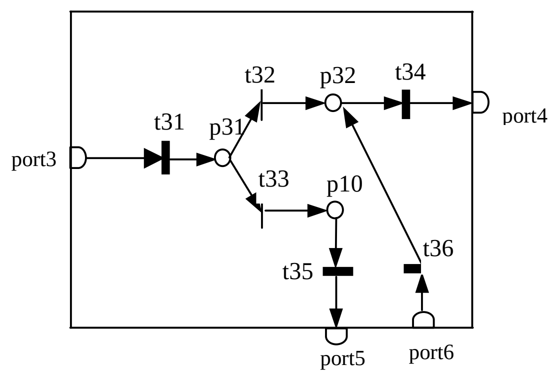

Figure 5. The operational model of the checking subsystem.

Table 3 Legends of places and transitions in Figure 5.

P_Set	DESCRIPTION	
p31	Checking result is available	
p32	Ready for sealing	
p33	Ready for being loaded onto the vehicle	
T_Set	DESCRITION	TIME
t31	Checking the product	[1, 3]
t32	The product passes the test	- -
t33	The product doesn't pass the test	- -
t34	The subsystem seals the product	[2, 4]
t35	The subsystem loads the product onto the vehicle	[1, 2]
t36	The subsystem unloads the product from the vehicle	[1, 2]

So far, we have presented the top-level RAS model of the manufacturing system. Next, we consider the refinement of the model.

4.5 Model Refinement

In this section, we illustrate the refinement of component of checking-subsystem (CS) into a sub-architecture which is composed of four components.

Figure 6 shows the RAS interface of the refined checking subsystem, which is composed of four components: Checking Center, Analysis Unit I, Analysis Unit II, and Sealing Unit. After receiving a product from the processing subsystem, the checking center does some checking operations and then sends the resulting data to the two analysis units for analysis (T11 and T13 fire). After the data analysis, the two analysis units return the analysis results to the checking center (T12 and T14 fires). The checking center then fuses the analysis results and determines if the product satisfies quality control conditions. If it does (T15 fires), then the product will be sealed and output as the final product of the whole system. If it doesn't (T16 fires), it will be sent to the repairing subsystem for repair. When the repairing activity is finished, it will be returned to the sealing unit and sealed. Notice that the connection transitions T15 and T16 forms a random switch, and they have the same meanings as t32 and t33 in Figure 5, respectively.

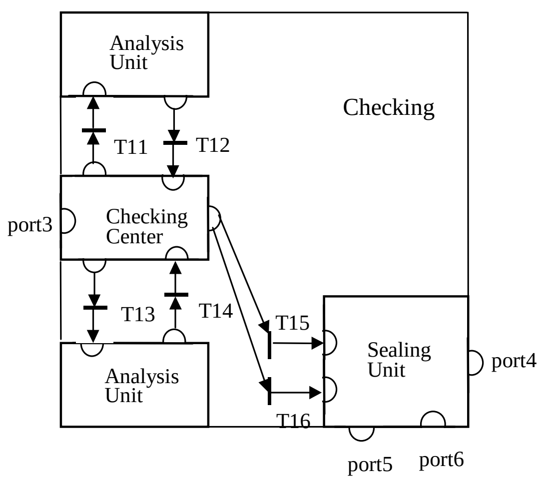

Figure 6. Refinement of the checking subsystem.

In order to be consistent, this refined model must satisfy all constraints that the old component design shown in Figure 6 satisfies. Otherwise, we cannot safely plug the new component into the system's RAS model. From the top-level RAS model we know that component CS is subject to a component constraint c_2 and has a requirement of environmental constraint c_3. Therefore, this refined design must satisfy c_2 whenever the environmental constraint c_3 is satisfied.

4.6 Summary

Based on the above discussion, we may find that compared with other formal methods, such as ordinary Petri nets, our model has the following advantages:

(1) The modular structure of RAS helps conquer the complexity of the design of FMS. If we want to change the design of a particular component of the FMS, then we only need to change the representation of this component and make sure that it satisfies all the necessary constraints, without a need to redesign the entire model.

(2) The linkage between design and requirements helps achieve a high-assurance design of FMS. When we redesign or refine a component of the manufacturing system, all the ports of the component will be inherited in the new design, and some constraints related to the component are also decomposed and added to the new design, which ensures the consistency of requirements between the sub-system and the system.

5. Constraint-Driven Compositional Analysis

The process of verification is driven by showing that the components and their composition satisfy their corresponding constraints at every design level. The modular nature of RAS and its emphasis on maintaining a strong correlation between design and requirements provide a natural support for incremental verification and for enforcing conformance of the design to the requirements. We have developed a verification technique that helps to achieve these benefits. We cut down the verification complexity by supporting horizontal as well as vertical composition-ability. Due to the space limitation, we only present the basic ideas of our verification algorithm and illustrate its working on the FMS described in Section 4. A detailed description of the algorithm is given in [9].

Our technique works by verifying designs independently at each level – starting from the very top level. At any given level, verification proceeds by analyzing the components against their corresponding component constraints one at a time and then composing these results to deduce system-wide properties at that level. Thus, the complexity of analysis is proportional to the size and number of components, rather than the size of the entire model. After a design has been verified at a given level, it is further refined into lower level design. The high level constraints are propagated to the lower level. Then the design at the lower level is checked against the lower level constraints, and the refinement process continues. At every level, we make sure that the constraints are consistent with the constraints at the parent level so that a verified lower-level design can be safely plugged into its parent level architecture without having to reverify the entire model.

We first describe the verification algorithm for a given level of design. Then we describe verification across design levels.

5.1. Incremental Verification at a Given Design Level

Our main technical tool is a set of reduction rules that lets us replace many types of RAS components by simple TPN of constant size, while preserving the ports and the time-dependent properties of the component as specified by the constraints. In [9], we introduced a set of *component-level reduction rules* to support such closure. It should be pointed out that unlike other reduction methods of TPN, such as given in [25], which work at individual transition level, our reduction rules are constraint-oriented and work at the component level. Consequently, our method works at a much coarser level than the reduction rules given in [25], and we need fewer applications of our rules to reduce the size of the model being analyzed. Our reduction rules assume that the underlying (untimed) net corresponding to the TPN of a RAS model is safe. ([25] makes a similar assumption.)

Our algorithm works in two stages. In the first stage, we order the components so that the timing behavior of any component depends only on the components which are lower in this order. This stage assumes that the underlying dependency-graph of components is acyclic. In the next stage, we consider components one at a time in the order computed in the previous stage. Each component is verified against its component constraints and then reduced to a simple TPN by one or more applications of reduction rules.

We ordered the components in such a way that the first component does not depend on any other components so it can be verified and reduced to a simple TPN. The second component depends only on the first one, which has already been reduced, and so on. Suppose that there are *m* components, which are linked together by inter-related connection structures to form the system or subsystem. Then our incremental verification method has three steps:

(1) For each component, list the reduction rule that can be applied to it; and

(2) Build the dependency graph on components and perform a topological sort. If the sorted order is CP_1, CP_2, …, CP_m, then we know that $\forall$ i > j, CP_j doesn't depend on CP_i.

(3) Reduce each component according to the sorted order, and verify constraints at the same time, until all constraints are checked.

We now illustrate the incremental verification on the FMS described in Section 4. The assumption that all three constraints are applied in the case that no waiting time exists for any workpiece at any processing stage ensures that we may use the net with only one input from port1 to verify the system constraints, which implies that the underlying untimed Petri net of the FMS is safe and our component-level reduction rules can be used. The FMS consists of three components: PS, CS and RS. There is one dependency edge: CS$\rightarrow$RS, so we can order these components as

PS, RS, CS.

Applying reduction rules to components PS and RS results in Figures 7(a) and (b). Suppose that applying simple reachability analysis [4] to the operational models of components PS and RS we conclude

$SI(t_{(1,2)}) = [11, 22]$, and

$SI(t_{(7,8)}) = [7, 13]$,

where $SI(t_i)$ represents the static firing time interval of transition t_i. Then, the constraint c_1: port1 $\rightarrow AF^{\leq 25}$ port2 is verified. Next, we consider the component CS. Figure 7(c) shows CS and its interaction with simplified PS and RS,

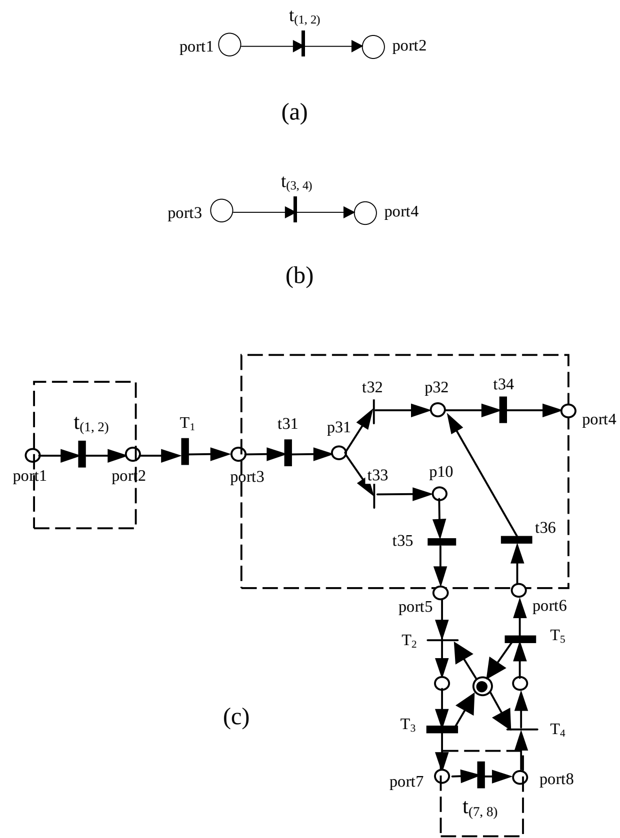

Figure 7 (a) Reduction of component PS.
(b) Reduction of component RS.
(c) The simple but equivalent model of the system.

and we use it to verify other constraints. Based on Figure 7(c) and the fact that the switch probabilities for transition t32 and t33 are 0.9 and 0.1, respectively, we obtain

$$D\_(\text{port5}, \text{port6}) = SI(T_3) + SI(t_{(7,8)}) + SI(T_5) = [9, 17],$$

and

$$\begin{aligned} D\_(\text{port3}, \text{port4}) &= 0.9 \times [\, SI(t_{31}) + SI(t_{34})] \\ &\quad + 0.1 \times [\, SI(t_{35}) + D\_(\text{port3}, \text{port4}) + SI(t_{36}) + SI(t_{34})] \\ &= [4.1, 9.1]. \end{aligned}$$

where $D\_(A, B)$ represents the message transferring time delay between A and B. Based on these values, we have verified c_3: port5 $\rightarrow AF^{\leq 20}$ port6 and c_2: port3 $\rightarrow AF^{\leq 12}$port4. Therefore, all constraints have been verified.

5.2. Constraints Decomposition and Incremental Verification across Design Levels

Our technique for constraint decomposition and incremental verification across design levels consists of three basic elements: 1) We automatically derive constraints for the lower-level design that is consistent with the higher-level constraints. 2) The lower-level design is verified using the technique discussed in the last section. 3) The sub-architecture is plugged into the parent-level architecture to form a more detailed architectural model.

In fact, when we refine an RAS component into a sub-architecture, the RAS model mandates that it inherit all the ports from the high-level component to ensure interface consistency. During the verification of the high-level design, we compute time delays for certain port pairs of (input port, output port). Correctness of the high-level design is contingent on the values of these time delays. So for the lower level design, we add these delays as constraints that must be satisfied. As long as these constraints are satisfied, we can ensure that the lower-level designs is consistent with the high-level design. If on all these port-pairs there are component constraints defined, then the lower-level design is just required to obey these component constraints.

Let's take the analysis of the FMS as an example. In the last subsection, we finished the verification of its high-level design. Figure 6 shows ports port3, port4, port5 and port6 are inherited from Figure 5, the high-level (top-level in fact) design model for CS. From the top-level RAS model we know that component CS is only subject to a component constraint c_2. Therefore, c_2: port3 $\rightarrow AF^{\leq 12}$port4 is used as the constraint of the lower-level design.

6. Conclusion

Based on the requirement of the modeling, design, and analysis of FMS, we have presented an RAS-based incremental approach to architectural modeling and verification of real-time distributed systems. The contribution of this paper is twofold: First, it provides a systematic way to link real-time system constraints to the process of formal modeling and analysis to ensure that the constraints are met at any given design level. Second, our approach is scalable in both modeling and analysis. As an application example, we illustrate the use of RAS to incrementally model a given FMS. It further explains the RAS framework as well as shows the benefits of using the RAS framework.

References

[1] M. Baldasssri and G. Bruno. Protob: An object methodology for developing discrete event dynamic system. *High-level Petri Nets: Theory and Application*, 1991.

[2] R. Bastide, C. Blanc and P. Palanque. Cooperative objects: A concurrent Petri-net based, objected-oriented language. *Procedings of the IEEE International Conference on System, Man and Cybernetics*, 3:286-291, 1993.

[3] E. Battiston, F. Cindio and G. Mauri. Objsa nets: A class of high-level nets having objects as domains. *Advances in Petri Nets*, *Lecture Notes on Computer Science*, CS 340, 1988.

[4] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3), 1991.

[5] H. V. Brussel, Y. Peng and P. Vallckenaers. Modeling flexible manufacturing systems based on Petri nets. *Annals of the CIRP*, 42(1):479-484, 1993.

[6] G. Bucci and E. Member. Compositional validation of time-critical systems using communicating time Petri nets. *IEEE Transactions on Software Engineering*, 21(12): 969-992, 1995.

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

[8] Y. Deng, C. Yang. Architecture-driven modeling of real-time concurrent systems with application in FMS, to appear in *Journal of Systems and Software*.

[9] Y. Deng, J. Wang and R. K. Sinha. Integrated Architectural Modeling of Real-Time Concurrent Systems with Applications in FMS. *Technical Report, School of Computer Science, Florida International University*, 1997.

[10] K. A. D'souza and S. K. Khator. A survey of Petri net applications in modeling controls for automated manufacturing systems. *Computers in Industry*, 24:5-16, 1994.

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

[12] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasian. Quantitative temporal reasoning. *Real-Time Systems*, 4:331-352, 1992.

[13] H. P. Huang and P. C. Chang. Specification, modeling and control of a flexible manufacturing cell.

International Journal of Production Research, 30(11):2515-2543, 1992.

[14] G. M. Knapp, and H. P. Wang, Modeling of automated storage/retrieval systems using Petri nets. *Journal of Manufacturing Systems*, 11(1):20-29, 1992.

[15] Y. Lee and S. Park. Opnets: An object-oriented high-level Petri net model for real-time systems. Journal of Systems & Software, SE-13(3):69-86, 1993.

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

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

[18] D. C. Luckham, J. Vera and S. Meldal. Three concepts of system architecture, Technical Report, Stanford University, July 1995.

[19] D. Mandrioli, A. Morzenti, M. Pezze, P. Pietro S. and S. Silva. A Petri net and logic approach to the specification and verification of real time systems. Formal Methods for Real time Computing, 1996.

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

[21] T. Murata. Petri nets: Properties, analysis and applications, *Proceedings of IEEE*, 77(4).

[22] A. Pnueli. The temporal logic of programs, *Proceedings of 18th Annual IEEE Symposium on Foundations of Computer Science*, 46-57, 1977.

[23] F. Qadri and A. Robbi. Timed Petri nets for flexible manufacturing cell design. *IEEE International Conference on Systems, Man, and Cybernetics*, Texas, 1695-1699, October 1994.

[24] C. S. Shukla and F. F. Chen. The state-of-the-art in intelligent real-time FMS control: a comprehensive survey. *Journal of Intelligent Manufacturing*, 1997.

[25] R. Sloan and U. Buy. Reduction rules for time Petri nets. Acta Informatica. 33:687-706, 1996.

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

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

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

Appendix I. Formal Description of RAS Model

The most important part of the formal description of RAS model is a recursive method of combining lower-level RAS models into a higher level model. We will describe this recursive method later. First, we describe the elements of the RAS model. An RAS model of a system consists of: (1) its externally observable and operational behavior (modeled by TPN), (2) its ports of interaction with its environment (modeled by a specific subset of places of the TPN), and (3) the constraints it needs to satisfy and the assumptions about its environment (modeled by a set of RTCTL formulas). In other words, an RAS model *RAS* is defined as a triple *RAS* = (*OPMODEL, PORT, CONSTRAINTS*), where:

(1) *OPMODEL* is the operational model of the system, described by a time Petri net, i.e., *OPMODEL* = (P, T, B, F, M_0, SI) [4].

(2) *PORT* denotes the communication ports of the model. All constraints and connections are defined on *PORT*. That is, the internal details of the design are hidden. These communication ports are a subset of the places in time Petri net used to describe the *OPMODEL*.

(3) *CONSTRAINTS* defines the behavior that the design should exhibit as well as the assumptions about its environment. They are modeled by a set of RTCTL formulas. *CONSTRAINTS* is divided into three groups: component constraints, environmental constraints, and path constraints.

Now we give a recursive procedure for combining lower level RAS models into a higher level model. A set of RAS models, RAS_1, RAS_2, …, RAS_k, can be combined to form a higher level RAS model *RAS* as follows (In this case, RAS_1, RAS_2, …, RAS_k are viewed as components or sub-architecture models):

(1) *RAS.PORT* is a subset of $\cup^{k}_{i=1} RAS_i.PORT$. The remaining ports in components, $\cup^{k}_{i=1} RAS_i.PORT - RAS.PORT$, are called the *internal* ports of *RAS*.

(2) *RAS.OPMODEL* has two parts: one is operational models from all components, and the other is a new time Petri net (called *CONNECT*) describing connections between individual components. Formally, $RAS.OPMODEL = \sum^{k}_{i=1} RAS_i.OPMODEL \cup CONNECT$, where $CONNECT = (P, T, B, F, M_0, SI)$ is a time Petri net describing the connections between the

lower level models RAS_1, RAS_2, …, RAS_k, that satisfies the following rules:

i. *CONNECT* can use any port from a component as a place. It can also introduce new places, but it can not use an internal place from a component. That is, $\cup^{k}_{i=1}(RAS_i.OPMODEL.P - RAS_i.PORT) \cap CONNECT.P = \varnothing$.

ii. The set of transitions of *CONNECT* are disjoint from the transitions in components, i.e., $\cup^{k}_{i=1}RAS_i.OPMODEL.T \cap CONNECT.T = \varnothing$.

iii. Every internal port must be connected to at least one transition in *CONNECT* through an incoming or outgoing arc.

(3) *CONSTRAINTS* is a set of RTCTL formulas describing the constraints of *RAS*.

Appendix II. Rules for the Use of RAS Model

The aim of RAS is to provide a formal approach to systematically maintain a strong correlation between (real-time) requirements and design and to verify the conformance of design to the requirements, and to offer better scalability in modeling and analysis. In order to achieve this goal, it is necessary for an RAS user to obey the following two rules on the proposing of constraints:

(1) Each component of a system must be subject to some constraints directly (there are some component constraints defined on it) or indirectly (there are some constraints that do not deal with its ports, but affect its design, i.e., the operational property is affected by these constraints). In other words, the operational property of a component must satisfy some requirements. Only in this way can we ensure the combination of designs and requirements.

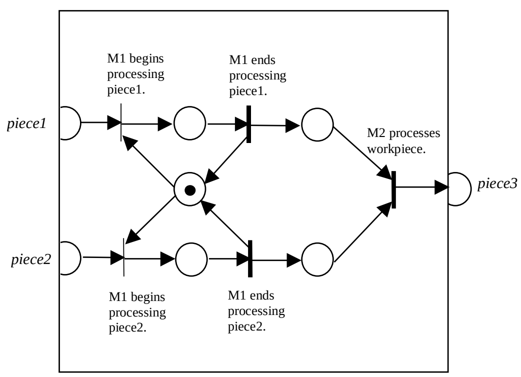

Figure 8. A simple manufacturing system.

(2) If there exists a synchronization activity for messages from some external input ports, then an environmental constraint which state the maximum and/or minimum distance among the arrival of messages to these ports is often needed. Otherwise, some constraints may make no sense. Let's consider a simple example. Consider a simple manufacturing system, which is composed of two machines, MA and MB. Two types of work-pieces, corresponding to piece1 and piece2, are assembled by MA and then processed by MB, and a new product, corresponding to piece3, is produced in turn. The Petri net model of such a simple system is shown in Figure 8, with three ports: *piece1*, *piece2* and *piece3*. If we only propose a constraint as

$$piece1 \rightarrow AF^{\leq 10}\, piece3,$$

it is not enough to describe the requirement on the system, because we don't know if piece2 is available or when piece2 is available after the arrival of piece1. But, it will make sense if we give a second constraint as

$$piece1 \rightarrow AF^{\leq 3}\, piece2.$$

In addition, in order to maintain consistency, when replacing a component design (indicated as an *old* component design) with an alternative component design, or decomposing it into a sub-architecture which is composed of several new components, we must obey the following two rules:

(1) *Same ports*, i.e., the alternative component design has exactly the same ports as the old component design, or the sub-architecture takes all ports of the old component design as all its external ports.

(2) *Conform to the same constraints*, i.e., the alternative component design or the decomposed sub-architecture must conform to all constraints which the old component design are subject to.