Incremental Modeling and Verification of Flexible Manufacturing Systems[1]

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

Abstract

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 to 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. A Real-time Architectural Specification (RAS) model and its application in the modeling of flexible manufacturing system (FMS) are presented. 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 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.

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

---

[1] Received March 1, 1998; revised August 18, 1998. 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 to 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 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 Petri net-based models do not provide explicit or adequate support to this task.

In this paper, we go beyond conventional Petri nets 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. The RAS model, including its conceptual framework 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 propose an incremental verification technique based on the RAS model. The formal description of a set of component-level TPN reduction rules which support our incremental verification technique are given in Appendix.

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, 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 are listed below [5, 7, 13, 14, 16, 20, 23, 26-31]:

- 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 identify production bottlenecks, and to assess the capacity and utilization of equipment.

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. RAS Models

In this section, we first give a conceptual framework of the RAS model. Then we give a brief introduction to TPN and RTCTL, two underlying formal methods of our RAS model. Finally, we formalize the RAS notation.

3.1 Conceptual Framework of RAS Model

Our goal is to develop both a rigorous approach to enhance the integrity of design and an evolutionary process to control complexity in system modeling and analysis. To the end, 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. As refinement to the system design goes on, these architectural constraints are also decomposed and propagated to the lower levels of system architecture (possibly new constraints are introduced). The consistency between higher-level and lower-level design constraints are either automatically maintained or verified to ensure consistency and progressive characteristics of analysis.

More specifically, an RAS model consists of three basic elements: *component models*, *inter-component connections* (*connections* in brief), and *architectural constraints* (*constraints* in brief) organized into multi-design levels. The component models describe the real-time behavior and communication interface of the components. The connections specify how the components interact with each other and, in turn, form the *system composition model*. Finally, the constraints define real-time system requirements imposed on the components and connections. All

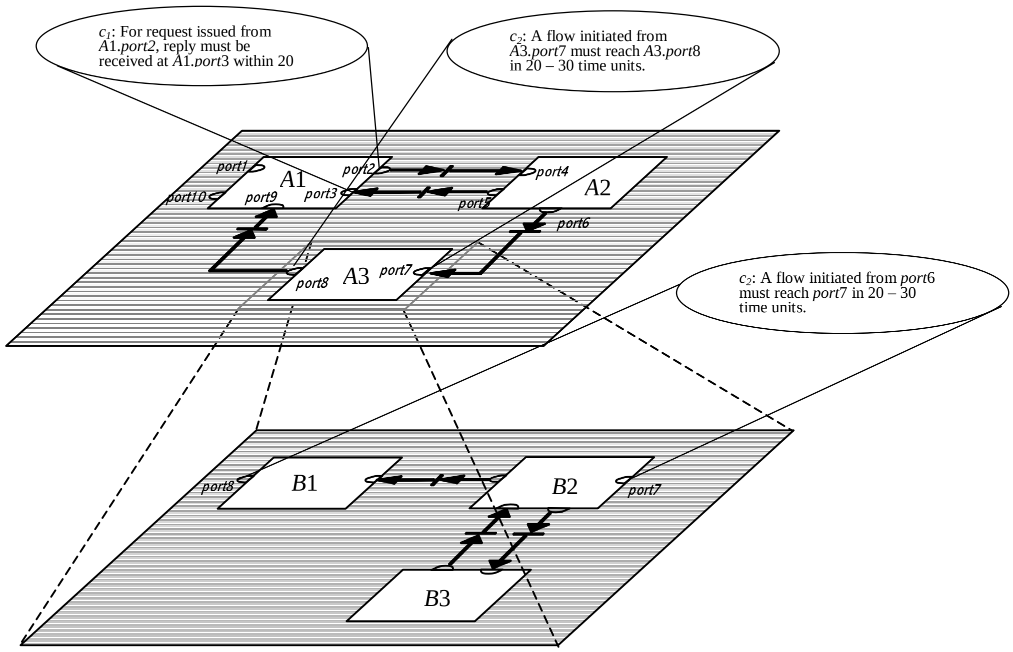

Figure 1. Framework of the RAS model.

connections are defined using only communication interfaces, which gives us the flexibility to change the design of individual components without voiding the analysis of the entire system.

A typical RAS model is illustrated in Figure 1. The high-level design has three components – *A*1, *A*2, and *A*3. Component *A*3 is further refined at the next design level into the composition of components *B*1, *B*2, and *B*3. The design at any level must satisfy the constraints specified at that level. A component model has two parts: (1) *communication ports* (denoted graphically by half circles), including *input ports* (e.g., *port*7) and *output ports* (e.g., *port*8), 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*1 and *A*2 have a request-reply relationship that is modeled by the bi-directional channel. At any given design level, ports can also be divided into *external ports*, e.g. *port*1, and *internal ports*, e.g. *port*3 and *port*4. An internal port describes interaction between components within a system (or subsystem), while an external port describes the inputs and outputs from and to the environment of a system (or subsystem).

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. In particular, time-critical system constraints are specified by RTCTL formulas defined over ports, where each port represents an *atomic proposition*, which is true at the moment that a token arrives in 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 between $A1$ and with $A2$, and constraint c_2 specifies the required processing time of component $A3$. This arrangement is critical to achieve the goals of incremental modeling and analysis described earlier, as it ensures that the component designs can be treated as black-boxes in the construction, understanding and analysis of a system's architecture. As more detailed system architecture is constructed by decomposing one of the system's components, it becomes possible for us guarantee the satisfaction of system-wide requirements by verifying the new sub-architecture meets its obligation imposed on its interface.

From the viewpoint of origination, architectural constraints are divided into two groups, one is *original constraints*, which reflect the user's requirements on the system under developed, and the other is *derived constraints*, which are intermediate constraints derived from original constraints — Notice that while original constraints may indeed be few in number, maintaining functionally correct original values may involve a large set of interacting components. Thus, to ensure these original constraints are satisfied, each of these components will, in turn, be subject to their own derived constraints. From the viewpoint of function, architectural constraints in an RAS model can also be logically divided into three classes: *component constraints*, *environmental constraints*, and *path constraints*. A component constraint describes a time-critical property of a component that its environment expects from it. An environmental constraint describes a time-critical property that a component expects from its environment. A path constraint describes a time-critical property for message transmission across components, which normally describes a system-wide (or subsystem-wide) timing requirement.

3.2 Underlying Formal Methods

In this section, we give a brief introduction to TPN and RTCTL, which are two underlying formal methods of our RAS model.

A. Time Petri Nets

A TPN is a tuple (P, T, B, F, M_0, SIM) where:

- P is a finite nonempty set of places;

- T is a finite nonempty, set of transitions; it will appear in the sequel that it may be convenient to view it as an orders set;
- B is the backward incidence function, , where N is the set of nonnegative integers;
- F is the forward Incidence function,;
- M_0 is the Initial Marking function, (P, T, I, O and M_0 together define a Petri net);
- SIM is a mapping called static interval, $SIM\colon T \to Q^* \times \left(Q^* \cup \infty\right)$, where Q^* is the set of positive rational numbers.

Let $SIM(t_I) = (\alpha^S, \beta^S)$ for some transition t_i, then the interval of numbers (α_i^s, β_i^s) is called the *static firing interval* of transition t_i, the left bound α_i^s the *static earliest firing time* (state EFT for short), and the right bound β_i^s the *static latest firing time* (static LFT for short).

A *state S* of a TPN is a pair $S = \left(M, I\right)$ consisting of a marking M and a firing interval set I which is a vector of possible firing times. The number of entries in this vector is given in the number of the transitions enabled by marking M.

Transition t_i is *firable* from state S=(M, I) at time $\tau + \theta$ if and only if:

(1) t_i is enabled by marking M at time τ ;

(2) The relative firing time θ to the absolute enabling time τ, is not smaller than the FET of transition t_i and not greater than the smallest of the LFT's of all the transitions enabled by marking M, i.e., EFT of $t_i \leq \theta \leq \min$ {LFT of t_k}, where k ranges over the set of transitions enabled by M .

Assume that transition t_i be firable at time $\tau + \theta$ from state S=(M, I). Then the state $S' = (M', I')$ reached from S by firing t_i at the relative time θ can be computed as follows.

(1) M' is computed, for all places p, as $(\forall p) M'(p) = M(p) - I(t_i, p) + O(t_i, p)$;

(2) I' is computed in three steps:

- Remove from the expression of I the intervals that are related to the transitions disabled when t_i is fired.
- Shift of the value θ towards the origin of times all remaining firing intervals, i.e., the intervals that remain enabled and so remain in I, and truncate them, when necessary, to nonnegative values.
- Introduce in the domain the static intervals of the new transitions enabled.

B. Real-Time Computational Tree Logic

An RTCTL Formula is defined as

- Each atomic proposition P is a formula.
- If p, q are formulae, then so are $p \wedge q$ and $\neg p$.
- If p, q are formulae, then so are $A(p\ U\ q)$, $E(p\ U\ q)$, and $EX\ p$.
- If p, q are formulae and k is any natural number, then so are $A(p\ U^{\leq k}\ q)$ and $E(p\ U^{\leq k}\ q)$.

A formula of RTCTL is interpreted with respect to a temporal structure $M = (S, R, L)$, where S is a set of states, R is a binary relation on S that is total (so each state has at least one successor), and L is a labeling which assigns to each state a set of atomic propositions, those intended to be true at the state. Intuitively, this temporal structure Σ represents the reachability graph of the architecture. A full-path $x = s_0, s_1, s_2, \ldots$ in Σ is an infinite sequence of states such that $(s_i, s_{i+1}) \in R$ for each i; intuitively, a full-path captures the notion of an execution sequence.

3.3 Formalization of RAS Model

An RAS consists of a set of compositions C (a composition may correspond to a design level, or the concept of sub-architecture given in Section 3.1) and a hierarchical mapping h:

RAS = (C, h).

(1) $C = (C_1, C_2, \ldots, C_k)$, and $C_i = \{ Cm, Cn, Cs\}$ for $i = 1, 2, \ldots, k$, where

- Cm is a set of components. Each $Cm_j \in C_m$ is defined by a TPN. Let

  $Cm_j.PORT\_IN = \{p \mid p \in Cm_j.P,\ {}^*P \cap Cm_j.T = \varnothing\}$;

  $Cm_j.PORT\_OUT = \{p \mid p \in Cm_j.P,\ P^* \cap Cm_j.T = \varnothing\}$;

  $Cm_j.PORT = Cm_j.PORT\_IN \cup Cm_j.PORT\_OUT$.

  $Cm_j.PORT\_IN$ is called the set of input ports of component Cm_j, $Cm_j.PORT\_OUT$ the set of output ports of Cm_j, and $Cm_j.PORT$ the set of ports of Cm_j. Moreover, for $\forall\ Cm_j, Cm_k \in Cm$,

  $Cm_j.P \cap Cm_k.P = \varnothing$,

  $Cm_j.T \cap Cm_k.T = \varnothing$.

- Cn is connection. $Cn = (P, T, I, O, M_0, SI)$ such that

$$Cn.P \cap \left[\bigcup_{Cm_j \in Cm} Cm_j.P \setminus Cm_j.PORT \right] = \varnothing,$$

$$Cn.T \cap \left[\bigcup_{Cm_j \in Cm} Cm_j.T \right] = \varnothing.$$

Also, in the TPN of composition C_i,

$$C_i.P \cap \left[\bigcup_{Cm_j \in Cm} Cm_j.P \right] \cup Cn.P,$$

$$C_i.T \cap \left[\bigcup_{Cm_j \in Cm} Cm_j.T \right] \cup Cn.T.$$

Let

$$C_i.PORT = \bigcup_{Cm_j \in Cm} Cm_j.PORT,$$

$$Cm_j.PORT\_INT = \{p \mid p \in C_i.PORT, {}^*P \cap C_i.T \neq \varnothing \wedge P^* \cap C_i.T \neq \varnothing\};$$

$$Cm_j.PORT\_EXT = \{p \mid p \in C_i.PORT, {}^*P \cap C_i.T = \varnothing \vee P^* \cap C_i.T = \varnothing\}.$$

$C_i.PORT\_INT$ is called the set of intrenal ports of component Cm_j, $Cm_j.PORT\_EXT$ the set of output ports of C_i, and $C_i.PORT$ the set of ports of C_i.

- Cs is a set of constraints. Each $Cs_j \in Cs$ is an RTCTL formula and it only uses ports as its atomic propositions. The atomic proposition is true The atomic proposition is true at the moment τ iff:

- marking transition happens at τ, and
- the port contains a token in the new marking.

In the temporal structure $\Sigma = (S, R, L)$, $S = (M, \varphi_M)$ where M is a TPN marking, and φ_M is the global time when the TPN enters M; R is a binary relation on S, which is indicated by firing transitions; and L is a mapping: $(M, \varphi_M) \rightarrow C_i.PORT$. In addition, the following condition is enforced:

$$\bigwedge_{Cs_i \in Cs} Cs_i \tag{1}$$

For $\forall$ $Cs_j \in C_i.Cs$, denote by $Cs_j.PORT$ the set of ports which are used as atomic propositions of Cs_j. If

$$Cs_k.PORT \cap Cm_j.PORT \neq \varnothing,$$

we say Cs_k is defined on Cm_j; if

$Cs_k.PORT \subseteq Cm_j.PORT$,

we say Cs_k is defined only on Cm_j. Let

$Cm_j.Cs = \{Cs_k \mid Cs_k \in C_i.Cs \wedge Cs_k.PORT \subseteq Cm_j.PORT\}$.

(2) $\forall\, C_i \in C$, $\forall\, Cm_l \in C_i.Cm$, h: $Cm_l \rightarrow C_j$, $j \neq i$, such that

- $Cm_l.PORT = C_j.PORT\_EXT$, (2)
- $Cm_l.Cs \subseteq C_j.Cs$. (3)

In the above definition, Exp. (1) states that all constraints should be consistent with each other, and it gives the *horizontal constraint consistency condition*. Eq. (2) states that when refining a component into a sub-architecture which may be composed of several new components, the sub-architecture must take all ports of the component as all its external ports. Eq. (3) states that when refining a component into a sub-architecture, the sub-architecture must conform to all constraints which the component are subject to. Such a consistency enforces that the system requirements are met in every step of the design process. Eqs. (2) and (3) together give the *vertical interface consistency conditions*.

For example, in the RAS model shown in Figure 1,

$C = (C_1, C_2)$,

$C_1.Cm = \{A1, A2, A3\}$, $C_2.Cm = \{B1, B2, B3\}$, $h(C_1.Cm.\ A3) = C_2$,

$C_1.Cn = \{c_1, c_2\}$, $C_2.Cn = \{c_2\}$,

$A1.PORT = \{port1, port2, port3, port9, port10\}$,

$A1.PORT\_IN = \{port1, port3, port9\}$, $A1.PORT\_OUT = \{port2, port10\}$,

$C_1.PORT = \{port1, port2, \ldots, port10\}$,

$C_1.PORT\_INT = \{port2, port3, \ldots, port9\}$, $C_1.PORT\_EXT = \{port1, port10\}$,

$C_2.PORT\_EXT = \{port7, port8\} = C_1.Cm.\ A3.PORT$,

$C_1.Cn = \{c_1, c_2\}$, $C_2.Cn = \{c_2\} = C_1.Cm.\ A3.Cn$.

… …

4. Incremental Modeling of an FMS

In this section, we illustrate the use of RAS to model an FMS. This will further explain the RAS framework as well as illustrate the benefits of using the RAS framework. We focus on the timing properties of an FMS, which is closely related to the execution delay of each activity of the system.

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 7 machines, indicated as M1-7. 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.

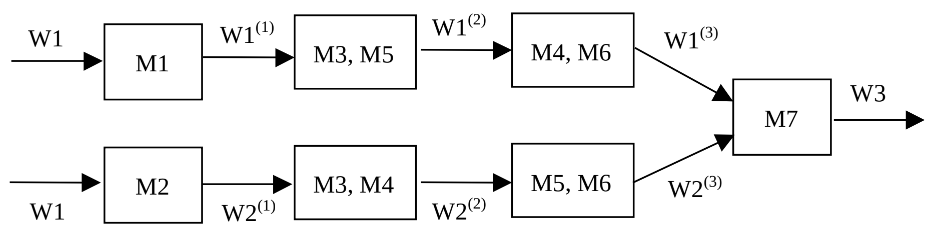

Figure 2. The processing flow of the processing subsystems.

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.

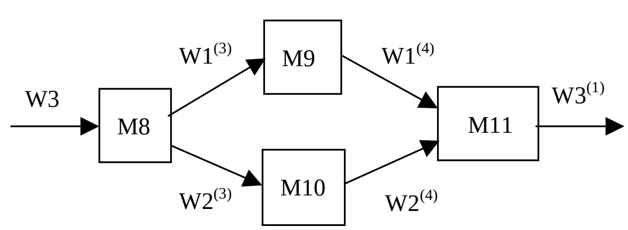

Figure 3. The processing flow of the repairing subsystems.

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 timing properties of 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 10 time units.

4.2 RAS Architecture of the System

In order to capture the profile of the operational model of the manufacturing system, we first present its RAS architecture in Figure 4, where 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. Table 1 and Table 2 show the meanings of all ports and connection transitions.

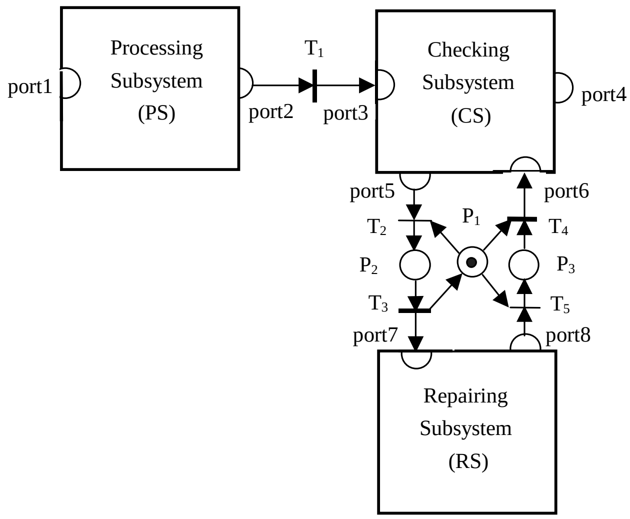

Figure 4. RAS architecture of the FMS.

Table 1. Legends of ports in Figure 4.

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

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

PLACE	DESCRIPTION	
P_1	Vehicle 2 is available	
P_2	Vehicle 2 is transferring the unqualified product to the repairing system	
P_3	Vehicle 2 is transferring the repaired product back to the checking system	
TRANSITION	DESCRIPTION	FIRING INTERVAL
T_1	Vehicle 1 transfers the product of the processing subsystem to the checking system	[1, 2]
T_2	Vehicle 2 begins transferring the unqualified product to the repairing system	- -
T_3	Vehicle 2 ends transferring the unqualified product to the repairing system	[1, 2]
T_4	Vehicle 2 begins transferring the repaired product back to the checking system	- -
T_5	Vehicle 2 ends transferring the repaired product back to the checking system	[1, 2]

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

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

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 of the processing subsystem is not longer than 12 time units. The third constraint, c_3, is

c_3: port5 $\rightarrow AF^{\leq 20}$ 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.

The importance of this step is twofold: First, the global system requirements are transformed into specific constraints on this particular architecture, more precisely, constraints imposed on the subsystems and connections between the subsystems. Consequently, we gain a clear understanding about what role each subsystem plays in satisfying the global requirements. This also eases analysis as now we deal with specific conditions imposed on each part of the system as oppose to requirement to the system as a whole. This feature becomes increasingly important as we approach to more detailed levels of architectural design where the association between a low-level system component and a system-wide requirement becomes much harder to grasp. Second, by formalizing time-critical requirements in terms of architectural constraints (based solely on the component interfaces), it not only removes ambiguity in the description, but also makes it easier to detect possible inconsistency or conflict between different (competing) requirements.

4.4 Operational Model of Each Component (Subsystem)

In this step, we use TPN as a tool to define the operational behavior of each component. Using the Petri net design methodologies, we can synthesize an ordinary Petri net model for a component based on its operations relationship among these operations. After we get such an ordinary Petri net, giving time requirements for various operation results in a TPN model, in which every transition is associated with an appropriate time delay interval.

The operational model of components are shown as in Figures 5, 6 and 7, and the legends of places and transitions involved in these models are listed in Tables 3, 4, and 5. Here, we use a thin bar to represent an immediate transition, and we use a thick bar to model timed transition. In Figure 6, 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.

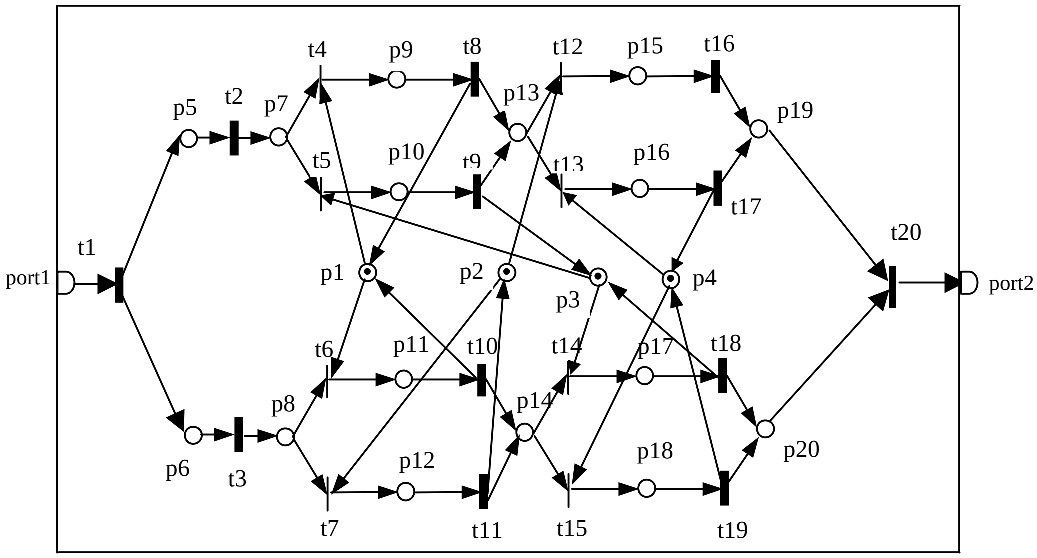

Figure 5. The operational model of the processing subsystem

Table 3. Legends of places and transitions in Figure 5.

PLACE	DESCRIPTION	
pi, i = 1,2,3,4	Mj is available, j = i+2	
p5 (p6)	W1 (W2) is ready for processing at M1 (M2)	
p7 (p8)	$W1^{(1)}$ ($W2^{(1)}$) is ready for processing at either M3 or M5 (M4)	
p9 (p10)	$W1^{(1)}$ is under processing by M3 (M5)	
p11 (p12)	$W2^{(1)}$ is under processing by M3 (M4)	
p13 (p14)	$W1^{(2)}$ ($W2^{(2)}$) is ready for processing at either M4 (M5) or M6	
p15 (p16)	$W1^{(2)}$ is under processing by M4 (M6)	
p17 (p18)	$W2^{(2)}$ is under processing by M5 (M6)	
p19 (p20)	$W1^{(3)}$ ($W2^{(3)}$) is ready for processing at either M4 (M5) or M6	
TRANSITION	DESCRIPTION	FIRING INTERVAL
t1	A pair of workpieces are loaded onto the two machines	[1, 2]
t2 (t3)	M1 (M2) processes W1 (W2)	[2, 4]
t4 (t5)	M3 (M5) begins processing $W1^{(1)}$	- -
t6 (t7)	M3 (M4) begins processing $W2^{(1)}$	- -
t8 (t9)	M3 (M5) ends processing $W1^{(1)}$	[2, 4]
t10 (t11)	M3 (M4) ends processing $W2^{(1)}$	[2, 4]
t12 (t13)	M4 (M6) begins processing $W1^{(2)}$	- -
t14 (t15)	M5 (M6) begins processing $W2^{(2)}$	- -
t16 (t17)	M4 (M6) ends processing $W1^{(2)}$	[2, 4]
t18 (t19)	M5 (M6) ends processing $W2^{(2)}$	[2, 4]
t20	M7 assembles $W1^{(3)}$ and $W2^{(3)}$	[2, 4]

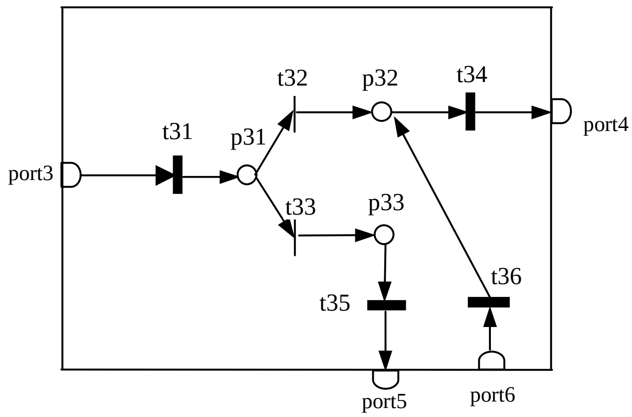

Figure 6. The operational model of the checking subsystem.

Table 4. Legends of places and transitions in Figure 6.

PLACE	DESCRIPTION	
p31	Checking result is available	
p32	Ready for sealing	
p33	Ready for being loaded onto the vehicle	
TRANSITION	DESCRITION	FIRING INTERVAL
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]

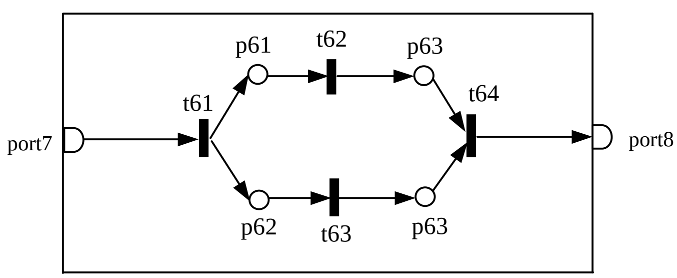

Figure 7. The operational model of the repairing subsystem.

Table 5. Legends of places and transitions in Figure 7.

PLACE	DESCRIPTION	
p61 (p62)	$W1^{(3)}$ $(W2^{(3)})$ is ready for processing at W9 (W10)	
p63 (p64)	$W1^{(4)}$ $(W2^{(4)})$is ready for processing at W11	
TRANSITION	DESCRITION	FIRING INTERVAL
t61	M8 disassembles W3	[2, 4]
t62 (t63)	M9 (M10) processes $W1^{(3)}$ $(W2^{(3)})$	[2, 4]
t64	M11 assembles $W1^{(3)}$ and $W2^{(3)}$	[3, 5]

So far, we have got a complete, executable and analyzable formal model that represents the top-level design of the manufacturing system. Next, we consider the refinement of the FMS model.

4.5 Support for Incremental Design of the FMS

In this section, we further discuss the incremental nature of RAS in the architectural modeling of the FMS. We illustrate the refinement of component of checking-subsystem (CS) into a sub-architecture which is composed of four components. Such an ability is necessary to make a modeling approach scale up.

Figure 8 shows the RAS sub-architecture of the refined checking subsystem, which is composed of four components, namely 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,

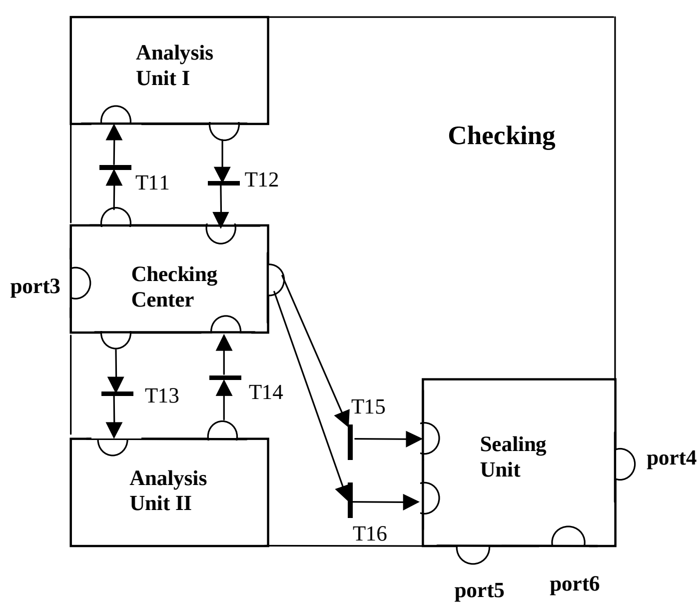

Figure 8. Refinement of the checking subsystem.

it will be returned to the sealing unit and sealed. Notice that the connection transitions T15 and T16 are the same as t32 and t33 in Figure 6.

As pointed out in Section 3.3, this refined sub-architecture must satisfy two vertical consistency conditions. The ports of the old component design shown in Figure 6 have been totally inherited by the sub-architecture; we also need to ask the sub-architecture conform to all constraints which the old component design are subject to. 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 sub-architecture must satisfy c_2 whenever the environmental constraint c_3 is satisfied.

The next important issue of our RAS model is the verification of constraints against the system operational model. We address it in the next section.

5. Constraint-Driven Compositional Verification

The modular nature of RAS model 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 two dimensional incremental verification technique that helps to achieve these benefits. In this section, we briefly introduce our verification technique and illustrate its illustrated its working on the FMS described in Section 4.

A general framework for our RAS verification technique can be described as follows: The process of verification is driven by showing that the components and their composition satisfy their corresponding constraints at every design level. We cut down the verification complexity by supporting horizontal as well as vertical composition-ability. 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 as specified by connection and cross-connection constraints 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.

The salient features of our technique are: 1) it enforces design integrity by maintaining a tight combination of design and system requirements and 2) it reduces the verification complexity by introducing many component-level reduction rules.

We first describe the verification algorithm for a given level of design. Then we discuss the verification across design levels. We assume that we have proved that all constraints are consistent each other.

5.1 Incremental Verification at a Given Design Level

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 of constant size that preserves the ports and all the externally observable time-dependent properties of the component.

The following two definitions are needed for the description of our verification algorithm.

Definition **1** Let *PORT_IN* and *PORT_OUT* be the input port set and output port set of a component, respectively. A pair of ports, <*port_in*, *port_out*>, where *port_in* $\in$ *PORT_IN* and *port_out* $\in$ *PORT_OUT*, is a *calling pair* of the component if a reply token from the environment is expected at *port_in* whenever a token is sent to some other component from *port_out*.

Definition **2** Let *PORT_IN* and *PORT_OUT* be the input port set and output port set of a component, respectively. A pair of ports, <*port_in*, *port_out*>, where *port_in* $\in$ *PORT_IN* and *port_out* $\in$ *PORT_OUT*, is a *passing pair* if a token is expected to be sent to the environment from *port_out* whenever a token from some other component is received at *port_in*.

For example, in Figure 4, <*port*5, *port*6> is a calling pair, whereas <*port*3, *port*4> is a passing pair.

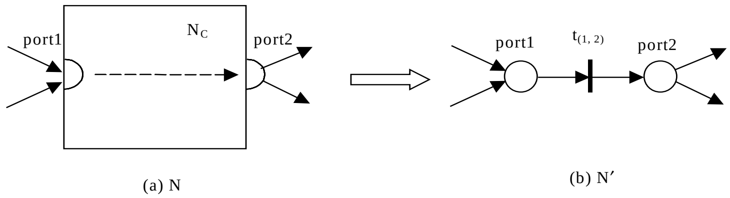

Figure 9. Illustration of component-level reduction rule 1.

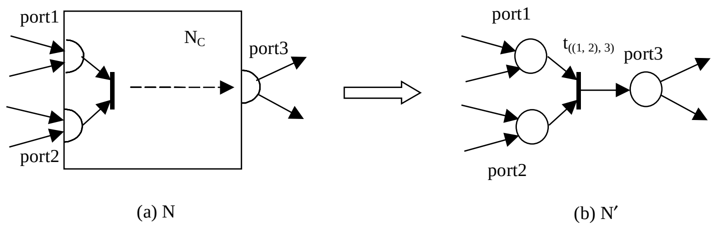

Figure 10. Illustration of component-level reduction rule 2.

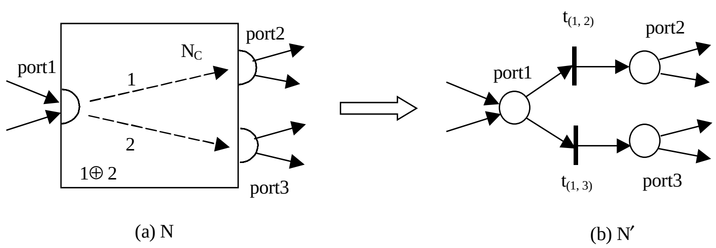

Figure 11. Illustration of component-level reduction rule 3. The token reached *port*1 will either follows token flow path 1 to reach *port*2, or follow token flow path 2 to reach *port*3.

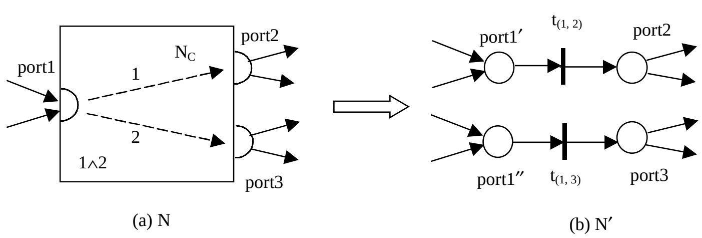

Figure 12. Illustration of component-level reduction rule 4. The arrival of a token at *port*1 will result in a token reaches each of *port*2 and *port*3.

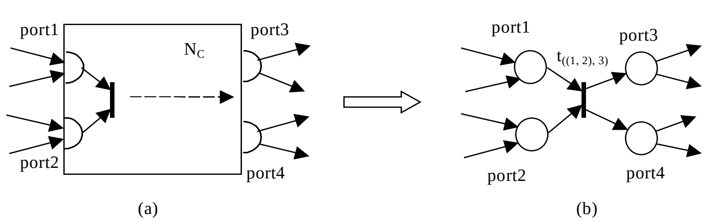

Figure 13. Illustration of component reduction rule 5.

The basic idea of the algorithm is to *close* passing pairs by connecting them with timed transitions, thereby replacing each component by a very simple net of constant size that preserves all external observable time-critical properties of the component being replaced. In Figures 9-13, we introduce a set of *component-level reduction rules* to support such closure (The formal description of these rules is included in Appendix). The reduction rules introduced here work at a much coarser level than the reduction rules given in [25], which work at individual transition level. Consequently, we need fewer applications of our rules to reduce the size of the model being analyzed. In these rules, the static firing time interval $SI(t_{(1,2)})$ of transition $t_{(1,2)}$ is the same as *D*_(port1, port2), the time delay interval of the message transfer between the epoch of port1 being marked and that of port2 being marked, and $SI(t_{((1,2),3)})$ of $t_{((1,2),3)}$ is the same as *D*_((port1, port2), port3), the time delay interval of the message transfer between the epoch of both port1 and port2 being marked and that of port3 being marked, and so on. Without the loss of generality, in our reduction rules, we assume that for any component model to be reduced, no transitions in the model are enabled in the initial state of the system. Particularly, we assume:

***Assumption* 1** The TPN model of the system under design is safe.

This assumption implies that the system takes the same statistical property of time to deal with inputs arriving in different time. This enables us to consider only one set of inputs to verify the system constraints. In this case, for any component to be reduced, no transitions in the component will be enabled by the coming tokens at its input ports until the previous input tokens have reached its output ports, which makes our reduction rules applicable. The assumption is looser than the one made in [25], which asks the underlying Petri net of a TPN is safe. However, the assumption is still a little strict: the reduction rules are applicable to verify the constraints of such systems that no queue exists. Nevertheless, these rules help to show one of the natures of our model, that is, it supports incremental verification.

Of course, we need to make sure that these reduction rules preserve timing properties of nets. Let $L(N)$ be the set of all firing schedules of TPN *N*. For any set of transitions *U*, let $L(N)|U$ be the set of all schedules of *N* where transitions not from *U* have been deleted from each of schedules.

***Definition* 3** Let N_R be the result of applying some reduction rule to a time Petri net *N*, and let *U* be any set of transitions of *N* which are left completely unmodified by this transformation. Then we say that a reduction rule is *equivalence preserving* if $L(N)|U=L(N_R)|U$.

Theorem *Reduction rules 1 - 5 are equivalence preserving.* (A proof is included in [9].)

Because of dependencies among components we need to be careful in choosing the order in which components are considered. E.g., if a component has a calling pair, it needs to interact with other components. We say that component C_i *depends on* component C_j, denoted by $C_i \rightarrow C_j$, if after C_i sends a token from the output port of its calling pair, it can not receive a token at the input port if we remove C_j away from the system. If $C_i \rightarrow C_j$ then C_j has to be analyzed before C_i. If $C_i \rightarrow C_j$ and $C_j \rightarrow C_l$, then C_l has to be analyzed before C_j, which, in turn, has to be analyzed before C_i. We can build a *dependency graph* such that there is a directed edge from component C_i to component C_j iff C_i depends on C_j. If this dependency graph contains a cycle, then we can't pick an order, so we assume that

***Assumption* 2** The dependency graph does not contain a cycle.

Since the dependency graph contains cycles may mean the existence of deadlock in a system, so in fact this assumption doesn't bring more limitation on the application of our verification technique. Based on this assumption we can (linearly) order the components (e.g., by performing a topological sort [8]) in such a way that all dependency edges go from a component higher in this order to a component lower in the order. In other words, 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 k components, which are linked together by inter-related connection structures to form the system or subsystem. Now we can describe our incremental verification method as follows:

Verification Algorithm

1. *For each component, list the reduction rule that can be applied to it.*
2. *Build the dependency graph on components and perform a topological sort. If the sorted order is $C_1, C_2, \ldots, C_k$, then we know that $\forall i > j$, C_j doesn't depend on C_i. Set $i = 1$.*
3. *For C_i,*
   3.1 *Verify environmental constraints defined on it. If violation is found, quit the algorithm, report the violation of constraints and redesign the system or subsystem.*
   3.2 *Verify path constraints defined on it. If violation is found, quit the algorithm, report the violation of constraints.*
   3.3 *Calculate time delay interval of the message transfer between each passing pair of the component;*
   3.4 *Verify component constraints. If violation is found, quit the algorithm, report the violation of constraints.*
4. *If all constrains are verified, then the algorithm ends successfully; else set $i = i+1$, and go back to 3*

We now illustrate the incremental verification on the FMS described in Section 4. To this end, we suppose that all buffers in the system can only store one part. Then the TPN model of the system 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 component PS and RS results in Figures 14(a) and (b). Applying simple reachability analysis [4], we conclude

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

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

So the constraint c_1: port1 $\rightarrow AF^{\leq 25}$ port2 is verified. Next, we consider the component CS. Figure 10(c) shows CS and its interaction with simplified PS and RS, and we use it to verify other constraints. Based on Figure 14(c) and the fact that the switch probabilities for transition t32 and t33 are 0.9 and 0.1, respectively, we obtain

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

and

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

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.

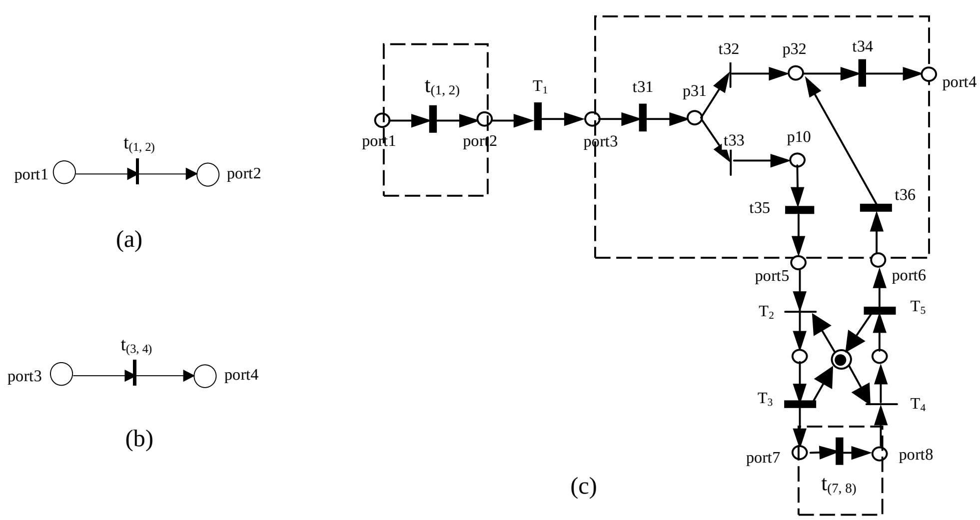

Figure 14. (a) Reduction of component PS.
(b) Reduction of component RS.
(c) The simple but equivalent model of the system.

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 8 shows ports port3, port4, port5 and port6 are inherited from Figure 6, 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, and illustrated the use of the approach to incrementally model a given FMS.. 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. We also proposed a two dimensional incremental verification technique, which further shows the benefits of our RAS model. To enlarge the applicable area of the verification technique, we are improving it by weakening its precondition for application.

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.

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

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

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

Appendix Formal Description of Reduction Rules

In this appendix we give the formal description of our component-level reduction rules for TPN's.

***Component-Level Reduction Rule* 1**

Let N be the TPN model of a system, and N_C the TPN model of a component in the system. *C.PORT_IN* = {*port*1}, *C.PORT_OUT* = {*port*2}. The component has no enabled transition under the initial marking of N. If

(1) whenever *port*1 receives a token, *port*2 is guaranteed to receive a token in the future, and

(2) *port*1 cannot receive another token until *port*2 has received a token,

then we can reduce N into N' by replacing N_C with a simple net which is composed of two places: *port*1 and *port*2, and one transition: $t_{(1,2)}$, such that

(1) $port1^* = {}^*port2 = \{t_{(1,2)}\}$, ${}^*t_{(1,2)} = \{port1\}$, $t^*_{(1,2)} = \{port2\}$, while ${}^*port1$ and $port2^*$ remain unchanged, and

(2) $SI(t_{(1,2)}) = SI\_(port1, port2)$, the latter is the time delay interval for the token transfer from *port*1 to *port*2. (See Figure 9)

***Component-Level Reduction Rule* 2**:

Let N be the TPN model of system S, and N_C the TPN model of component C of S. $C.PORT\_IN$ = {$port1$, $port2$}, $C.PORT\_OUT$ = {$port3$}, and $port1^* = port2^*$. The component has no enabled transition under the initial marking of N. If

(1) whenever both $port1$ and $port2$ receive a token, $port3$ is guaranteed to receive a token in the future, and

(2) at least one of $port1$ and $port2$ cannot receive another token until $port3$ has received a token,

then we can reduce N into N' by replacing N_C with a simple net which is composed of three places: $port1$, $port2$ and $port3$, and one transition: $t_{((1,2),3)}$, such that

(1) $port1^* = port2^* = {}^*port3 = \{t_{((1,2),3)}\}$, ${}^*t_{((1,2),3)} = \{port1, port2\}$, $t^*_{((1,2),3)} = \{port3\}$, while ${}^*port1$, ${}^*port2$ and $port3^*$ remain unchanged, and

(2) $SI(t_{((1,2),3)}) = SI\_((port1, port2), port3)$, the latter is the time delay interval from two tokens arriving in $port1$ and $port2$, respectively, to a token reaching $port3$. (See Figure 10)

***Component-Level Reduction Rule* 3:**

Let N be the TPN model of a system S, and N_C the TPN model of component C of S. $C.PORT\_IN$ = {$port1$}, $C.PORT\_OUT$ = {$port2$, $port3$}. The component has no enabled transition under the initial marking of N. If

(1) whenever $port1$ receives a token, one and only one of $port2$ and $port3$ is guaranteed to receive a token in the future, and

(2) $port1$ cannot receive another token until one of $port2$ and $port3$ has received a token,

then we can reduce N into N' by replacing N_C with a simple net which is composed of three places: $port1$, $port2$ and $port3$, and two transition: $t_{(1,2)}$ and $t_{(1,3)}$, such that

(1) $port1^* = \{t_{(1,2)}, t_{(1,3)}\}$, ${}^*port2 = \{t_{(1,2)}\}$, ${}^*port3 = \{t_{(1,3)}\}$, ${}^*t_{(1,2)} = {}^*t_{(1,3)} = \{port1\}$, $t^*_{(1,2)} = \{port2\}$, $t^*_{(1,2)} = \{port3\}$, while ${}^*port1$, $port2^*$ and $port3^*$ remain unchanged, and

(2) $SI(t_{(1,2)}) = SI\_(port1, port2)$, and $SI(t_{(1,3)}) = SI\_(port1, port3)$. (See Figure 11)

***Component-Level Reduction Rule* 4:**

Let N be the TPN model of a system S, and N_C the TPN model of component C of S. $C.PORT\_IN$ = {$port1$}, $C.PORT\_OUT$ = {$port2$, $port3$}. The component has no enabled transition under the initial marking of N. If

(1) whenever *port*1 receives a token, both of *port*2 and *port*3 are guaranteed to receive a token in the future, and

(2) *port*1 cannot receive another token until both of *port*2 and *port*3 have received a token,

then we can reduce N into N' by replacing N_C with a simple net which is composed of four places: *port*11, *port*12, *port*2 and *port*3, and two transition: $t_{(11,2)}$ and $t_{(12,3)}$, such that

(1) ${}^*port11 = {}^*port12 = {}^*port1$, $port11^* = \{t_{(11,2)}\}$, $port12^* = \{t_{(12,3)}\}$, ${}^*t_{(11,2)} = \{port11\}$, ${}^*t_{(11,3)} = \{port12\}$, $t^*_{(11,2)} = \{port2\}$, $t^*_{(12,3)} = \{port3\}$, while $port2^*$ and $port3^*$ remain unchanged, and

(2) $SI(t_{(11,2)}) = SI\_(port1, port2)$, and $SI(t_{(12,3)}) = SI\_(port1, port3)$. (See Figure 12.)

***Component-level Reduction Rule* 5:**

Let N be the TPN model of a system S, and N_C the TPN model of component C of S. $C.PORT\_IN = \{port1, port2\}$, $C.PORT\_OUT = \{port3, port4\}$, and $port1^* = port2^*$. The component has no enabled transition under the initial marking of N. If

(1) whenever both *port*1 and *port*2 receives a token, both of *port*3 and *port*4 are guaranteed to receive a token in the future, and

(2) at least one of *port*1 and *port*2 cannot receive another token until both of *port*3 and *port*4 have received a token,

then we can reduce N into N' by replacing N_C with a simple net which is composed of four places: *port*1, *port*2, *port*3 and *port*4, and one transition: $t_{((1,2),(3,4))}$, such that

(1) $port1^* = port2^* = {}^*port3 = {}^*port4 = \{t_{((1,2),(3,4))}\}$, ${}^*t_{((1,2),(3,4))} = \{port1, port2\}$ and $t^*_{((1,2),(3,4))} = \{port3, port4\}$, while ${}^*port1$, ${}^*port2$, $port3^*$ and $port4^*$ remain unchanged, and

(2) $SI(t_{((1,2),(3,4))}) = SI\_((port1, port2), (port3, port4))$. (See Figure 13)