6th Euromicro Workshop on Real-Time Systems, 1994, Västerås

Timing Analysis and Conditional Scheduling in a Real-Time System Design Environment

Zebo Peng and Anders Törne

Dept. of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden

Abstract

This paper presents an approach to carry out timing analysis and scheduling in a real-time system design environment for flexible manufacturing systems. It is built based on a formal model for specifying real-time systems using extended timed Petri nets. The designer uses the model to specify a design at a given level. The functional characteristics as well as timing properties of the design can then be analyzed with design evaluation procedures. To facilitate timing analysis, a conditional scheduling strategy has been developed to provide a pre-run-time schedule. The strategy is based on a list scheduling method with several priority schemes. This paper describes the design environment, the timing analysis procedure, and the basic strategy for scheduling.

1. Introduction

When designing complex real-time systems, such as flexible manufacturing systems (FMS), design verification must be carried out in the early design stages as well as in the final implementation phases. In order to guarantee correctness in terms of both functionality and timing, and to explore different design alternatives, methods and tools must be developed to support modeling, verification/simulation, and synthesis of such systems. These methods and tools should also be integrated in a uniform design environment in such a way that timing analysis and design verification can be done at the same time when stepwise refinement is carried out. We have been developing such a design environment at Linköping University [7], [8], [13].

One of the main features of our design environment is that it provides a formal notation to capture both the specification and its implementation as well as the intermediate designs in a *uniform* way. This means that once the specification is formulated and verified, a smooth pass to an efficient implementation can be achieved by exploring different design alternatives using iterative improvement techniques. Since the same modeling technique is used throughout the whole synthesis process, the designer needs to learn only one formal notation and the same analysis/scheduling procedures can be used for designs at different levels of abstractions, resulting in a *unified* design environment.

The modeling technique we have developed is based on timed Petri nets and used to model the basic operations, concurrency, synchronization, mutual exclusion, and sequential relations of the basic operations of real-time systems [7]. The Petri net based model is first used to capture the requirement specification of an FMS. Timing analysis can then be carried out on this Petri net based model. Since basic tasks of an FMS usually share resources, a scheduling strategy must be used whenever more than one task is trying to make use the same resource at the same time. Even though our modeling technique is not bound to any particular scheduling approach, the scheduling algorithm described in this paper is based on a non-preemptive pre-run-time scheduling strategy.

Our approach supports design of real-time systems that are *non-deterministic* in terms of operation sequences and operation time [7]. Therefore, the concept of *conditional* scheduling is introduced. It means that the schedule is not fixed but *relative* to a set of conditions which are set to be true or false depending on the real-time environment or outcomes of some operations at run-time (for example, sensor-generated information can be used to select among alternative operation sequences in an FMS). Therefore, a run-time control mechanism is needed to generate the final schedule, and our scheduling approach combines the strength of pre-run-time scheduling and dynamic scheduling.

This work was supported by the Swedish National Board for Industrial and Technical Development (NUTEK) and the Center for Industrial Information Technology (CENIIT) at Linköping University.

The issues of timing analysis and scheduling of real-time systems have been extensively studied and many algorithms have been reported in the literature [1], [2], [6], [11], [14]. Most of the previous works in scheduling model a real-time system as a set of periodic and/or aperiodic tasks with deadlines. For example, with the fixed priority approach [6], a system is modeled as a set of periodic, preemptible, and independent tasks, each with a fixed period and priority. A scheduling algorithm simply selects the highest priority task to be executed. And whenever a task with a higher priority is ready for execution, the task being executed will be preempted. Since FMSs are, in nature, event-driven, and the set of tasks to be executed in an FMS are highly dependent upon each other, a different scheduling model must be used. Petri nets have been used to formulate scheduling problems with routing flexibility and shared resources [5], [12]. In [12], for example, an on-line FMS scheduling system based on a timed Petri net representation is presented. It solves scheduling conflicts using a beam search based decision procedure which constructs partial schedules within the beam-depth and evaluates them to choose the best one. Petri nets have also been widely used in modeling real-time systems and development of synthesis techniques for real-time systems; see, for example, [2], [3] and [10].

This paper describes mainly a pre-run-time based scheduling strategy for FMS applications for timing analysis purpose. It should be pointed out that the main goal of our research is to integrate the timing analysis and scheduling techniques in a *systematic* design environment for modeling and design of complex real-time systems. Section 2 will briefly present the basic modeling technique based on timed Petri nets and the basic idea of the systematic design environment. The issues of timing analysis will then be discussed in section 3. Section 4 describes the conditional scheduling strategy and a scheduling algorithm, which is based on list scheduling with three priority schemes.

2. The Modeling Technique and Design Environment

The design methodology we have developed views a real-time system as consisting of a *controlling* sub-system and a *controlled* sub-system [7]. This view can be applied to different levels of abstractions and a complete real-time system is considered as a hierarchy of controlling/controlled sub-systems. In each of the abstraction levels, the controlled sub-system can be viewed as the environment with which the corresponding controlling sub-system interacts. In the context of a flexible manufacturing system, which is defined as a computer-controlled configuration of semi-independent manufacturing cells coupled with a material handling system, the controlling sub-system is the computer hardware together with the real-time software used to send controlling signals to the manufacturing cells. A particular configuration of the manufacturing cells forms a controlled sub-system.

The timed Petri net based model used to represent such a system is called Modified Timed Petri Net (MTPN). In MTPN, a timed Petri net is used to represent the control structure of a real-time system, namely the controlling sub-system. The controlled sub-system is modeled as an *activity diagram*. It specifies the basic components and the possible activities as well as the initial state of a controlled sub-system. The system components are modeled as a set of resources. Each resource has a finite set of possible states. A resource/state pair defines a vertex of the activity diagram. When the vertex is marked, the resource is currently in the corresponding state. An activity, or operation, is represented by an arc in the activity diagram and controlled by a set of places in the control Petri net. When one of the control places holds a token, the activity will be activated. As the result of the activity, the state of some resources of the modeled system will change. If the destination vertex set of an activity arc has only one element, the activity will result in a deterministic state transition. Otherwise, one of the destination vertices of the arc will be marked as the result of the activity, and the choice is *non-deterministic* [7].

In the MTPN model, timing information is attached to the Petri net places as well as the arcs of the activity diagram. The time attached to an arc models the time needed to complete an activity in the controlled sub-system. *Non-deterministic* characteristics of operation time will be handled by stochastic models with a upper bound and a lower bound. The time attached to a place is, however, a constant, which models the time needed to hold a control signal or the delay time needed to keep the controlled sub-system at certain state. Tokens arriving at the places must stay there for the specified time before they can be used to fire transitions. In the case when a place controls a set of activities which must be completed during the time when the control signal is on, the time attached to the place must be larger than or equal to the worst case operation time of the set of activities. In MTPN, it is assumed that transition firing is instantaneous and represents initiation or termination of activities.

We will use the following FMS example to illustrate the MTPN model, and later the timing analysis problem and the scheduling algorithm. The example shows a machining and assembly process. Three parts are to be machined and assembled — Part A, Part B and Part C. The materials for Part C may arrive in two variants (Id#1 or Id#2) and its machining operations will depend upon the

type of arrived materials which is identified by a vision system. Table 1 gives the basic required operation sequences for the individual parts and Fig. 1(a) shows the MTPN controlling sub-system denoting basically the partial order which must be satisfied by any schedules of this FMS.

Table 1: Process plans for the different parts

	1st op.	2nd op.	3rd op.
Part A	P_{11}	P_{21}	P_{12}
Part B	P_{13}	P_{22}	
Part C#1	P_{23}	P_{14}	
Part C#2	P_{24}		

The FMS example consists of two machines, a assembler and a vision sensor, together with some underlying material handling mechanism. The preliminary allocation of operations to the available resources is captured by the activity diagram given in Fig. 1(b) as the controlled subsystem. The arcs in the activity diagram denote the activities (operations) and are labelled with the controlling Petri net place which indicates when the operation will be activated. For example, when P_0 holds a token, the vision sensor will be activated and enter the *Ready* state. Note that P_0 is also the *initial* place of this example, and will hold a token initially.

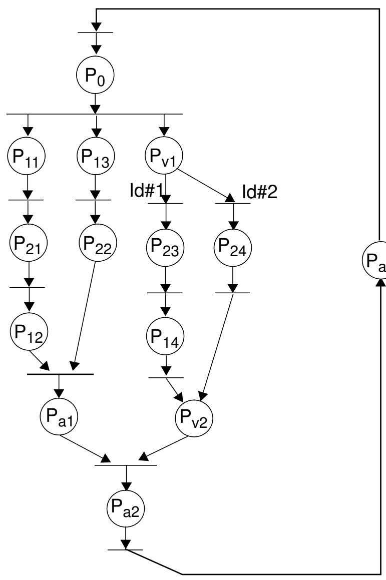
To complete the specification, timing information must be provided. Table 2 gives the operation times associated with the places which denote the times needed for the corresponding operations.

Table 2: Operation times for the places

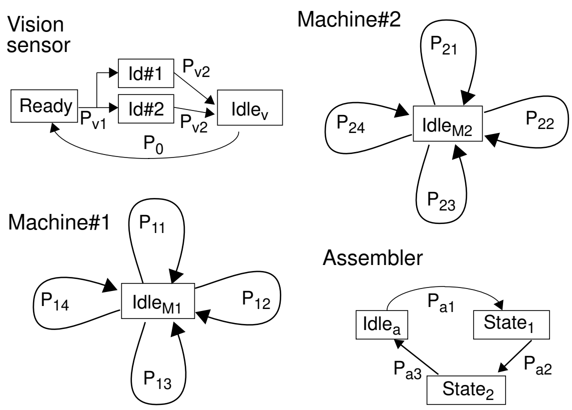
Operation	Time
P_{11}	6
P_{12}	10
P_{13}	8
P_{14}	7
P_{21}	10
P_{22}	8
P_{23}	12

Operation	Time
P_{24}	8
P_{v1}	2
P_{v2}	1
P_{a1}	6
P_{a2}	5
P_{a3}	1
P_0	1

In this FMS example, the two machines are general purpose in the sense that they can be used for different machining of the different parts, which is indicated by the several activities attached to them in Fig. 1(b). If more than one activity associated with the same machine (or any resource) will be activated at the same time, a scheduling technique must be used to decide which one is to perform first. The schedule must meet the timing constraints imposed on the execution cycle of the Petri net and other additional requirements. Before we discuss how this is



a) Controlling sub-system



b) Controlled sub-system

Fig. 1: A machining and assemble process in MTPN

achieved, we would like to briefly discuss the design process supported by our approach.

We assume that the designers start the design process by using the MTPN model to capture the basic requirement of the system to be designed, as in the case of the above example. They can then use Petri net analysis or simulation technique to verify the basic properties of the modeled system. For example, by generating the reachability graph of the Petri net, safeness property can be checked [7]. Once the functionality and timing property of the design is verified, the designers can proceed with the refinement of each activity in the controlled sub-system into the next level MTPN. The timing model associated with the activity will then become an end-to-end timing constraint for the refined MTPN model. In the case when the time attached to a control place must be larger than the

controlled sub-system's associated activities, the lower level controlling sub-system must guarantee that its controlled sub-system will finish its operations within the specified time. If this requirement cannot be satisfied, two solutions can be used. The first one is to change the scheduling strategy. The second one is to introduce more resources into the system and thus increase its parallelism. The implication of these two solutions will be discussed in the next two sections.

The designers can also choose to use the bottom-up approach. That is, they start with the basic level of design and continue with the higher level by considering the lower level design as an activity to be used to build the higher level MTPN model. In this case, the timing analysis procedure will be used to analyze the timing characteristics of the current MTPN, which will become the timing model of the high-level activity.

3. Timing Analysis

A timing constraint can formally be expressed as the time needed for a token to travel from an initial place to a final place in the MTPN notation. This implies an end-to-end deadline — we are interested in whether the whole set of operations will be completed before a given time, not the individual operations' deadlines. Such a constraint is usually introduced due to the timing requirement imposed by the external environment or a higher abstraction/design level in the design hierarchy, as discussed in section 2. In the latter case, the initial place represents the activation of an activity in a higher design level and the final place the completion of this activity. For example, in the example given in Fig. 1, a timing constraint may be imposed on the time for a token to flow from P_0 to P_{a3}, which indicates the completion of a product cycle of this machining and assemble process. This constraint is required by a higher level MTPN model since this manufacturing process is a part of a more complicated FMS.

The same notion of timing constraints can also be used to express other timing requirements in the FMS. For example, we can assume that there exists an extra timing constraint between P_{21} and P_{a1} (from the time when the operation controlled by P_{21} terminates to the time when the operation controlled by P_{a1} starts, to be more precise). This is due to that once the second machining operation of part A is done, it must be assembled within the given time interval <5, 15> (this could, for example, represent the time interval the temperature of the part is in the appropriate range).

There are two basic approaches for timing analysis which can be used by the designers. The first approach is based on the concept of a *critical path* between the two places, i.e., the longest path needed to finish the set of operations initialized by the starting place and terminating at the finishing place. A critical path thus consists of a sequence of control places which dominates the time needed for a token to flow from the starting place to the finishing place. It is analogous to critical paths in hardware systems and similar analysis algorithms can be adapted for this purpose [9]. In the example given in Fig. 1, the critical path consists of P_0, P_{11}, P_{21}, P_{12}, P_{a1}, P_{a2} and P_{a3}, and has a length of 39 time units. The critical path length is however a lower-bound of the needed execution time, which can only be achieved when it is assumed that whenever a token is deposited into a place, the execution of its corresponding activities will start. This means that maximal parallelism is supported and, in the FMS example, at least three machines are needed.

When limited amount of resources is available, a scheduling algorithm can be used to resolve conflict when simultaneous access to resources is not possible. The timing analysis algorithm must then take into account the scheduling strategy, which will be discussed in the next section. If we do not want to enforce any specific scheduling strategy, a mutual exclusion construct can be introduced to the set of places which can hold tokens at the same time and their corresponding activities share the same resource. In this way, these places will not longer be able to hold token simultaneously and only one resource is sufficient for the set of operations. We can then use critical path analysis method to find the longest path of operations and check whether it violates the timing constraint.

With this method, we assume that if several operations are enabled (i.e., ready for execution if the needed resource is available) at the same time, they can be executed in *any* order. This often results in the situation that the critical path length is longer than the maximal execution time which is allowed. In this case, additional partial order can be introduced into the Petri net to enforce a specific order in which the places receive tokens. For example, in the FMS example given in Fig. 1, an extra place and transition can be introduced before P_{13} by the designers. In this way, P_{11} will get a token first, which represents a decision to schedule P_{11}'s operation before P_{13}'s. This manipulation of the design is called a *transformation*. Transformations can also be applied to change the resource allocation. For example, some of the operations bound to Machine#1 can be moved to execute on Machine#2, and vice versa, in our example. With a sequence of such transformations, the designers explore the design space by enforcing different scheduling/resource allocation strategies.

If the above method does not work well or an automatic synthesis approach is desired, the second approach for timing analysis should be used, which assumes that a

scheduling strategy, as to be described in the next section, is used before timing analysis is performed.

4. The Conditional Scheduling Strategy

The problem of scheduling in the MTPN model arises mainly when several places hold tokens simultaneously and their corresponding activities share the same resources. For example, in the example given in Fig. 1, P_{11} and P_{13} will hold tokens simultaneously when their input transition is fired. Since both operations are mapped onto Machine#1, a scheduling decision must be made.

Scheduling is also needed when several transitions are enabled at the same time, but cannot be fired simultaneously. In our design environment, transitions are mainly used to model the changes of system states, which do not take time. In real implementation, the transition firing will take some small amount of time to execute and may share the same processor due to the need of condition checking. However, we assume that the transition firing times are very small compared to activity time and, therefore, negligible. We will consequently not consider the scheduling problem due to simultaneous transition firing.

In an FMS, most activities represent manufacturing processes which cannot be preempted. Therefore, we assume a non-preemptive scheduling model. Since a non-preemptive scheduling model is used, a dynamic scheduling approach may fail a timing constraint even if the resource utilization is not very high [14]. For example, a process *P* can block a time-critical process *Q* because *P* is released first and *Q* cannot preempt *P* even if *Q* will miss its deadline by waiting first for *P* to finish. We have therefore also assumed a pre-run-time scheduling approach.

In the MTPN model, the release of activities (i.e., the enabling of activity arcs by Petri net places holding tokens) can be non-deterministic (i.e., depending on conditions given by the environment), and the activity time can also be non-deterministic. Further the activity time can also be un-bounded, since the MTPN model allows self-loops in the Petri net [10]. Therefore, we cannot have a pure pre-run-time schedule.

To solve the above problem, we define *conditional scheduling* as a schedule class which comprises a set of basic schedules. Which basic schedule is really used will be decided at run-time based on a set of conditions. This means we must have a run-time control mechanism, as discussed before. The run-time arbitrator (dispatch), however, is very simple, since it only needs to check the conditions. This arbitrator can be captured by the notation of finite state machines and implemented with electronics circuitry [9].

We have selected to use a simple list-based scheduling algorithm for generating the final conditional schedule. The algorithm maintains a list of ready places, i.e., places with tokens. The places are ordered according to a three-level priority scheme. The first level to ne considered is related to deadlines given as extra local timing requirement, such as the one given between P_{21} and P_{a1} in the FMS example. The earliest deadline first approach is used to order the operations in this priority level.

The second level is defined by the distance from the place to the final place, i.e., how much time it is needed to pass a token from the current place to the final place. The longer the distance the higher the priority. This reflects the strategy to consider the most time-consuming set of operations first, since the main objective is to meet the end-to-end deadline. The second level will be considered when the first level does not resolve the conflict.

If the second priority level gives again equal value for the places in conflict, the third level priority will be used, which is defined by the time needed for the completion of the current activities. The shorter the operation time, the higher its priority.

The basic function of the scheduling algorithm is to generate the conditional reachability graph, using the priority scheme to resolve resource conflicts whenever needed. The algorithm is based on the Petri net reachability graph generation algorithm [10] with the following modifications. First of all, each reachable marking is associated with a time indicating how much time this marking will hold. The algorithm must keep track of the remaining operation times of the places when it generates the next marking.

The second modification is that when generating the next marking, a set of transitions will be fired simultaneously if possible instead of one single transition. Further only the maximally firable sets of transitions will be considered. A *firable* transition set is a set of transitions which are enabled by the current marking and the intersection of their guarding conditions are not always false. A *maximally* firable transition set is defined as a firable transition set that is not proper subset of any of the firable transition sets of the current marking. The maximally firable transition set concept is introduced to get rid of redundant markings in the generated schedule so as to reduce its complexity. A similar concept has also been used in digital hardware synthesis [9].

Finally, a one-step look ahead mechanism is used to check whether any of the ready places will be in resource-conflict with the places which will hold tokens when the enabled transitions are fired. If this is the case, the conflict will also be resolved by the priority scheme. For example, at the marking $\{P_{11}, P_{13}, P_{24}\}$, P_{24} is not in resource con-

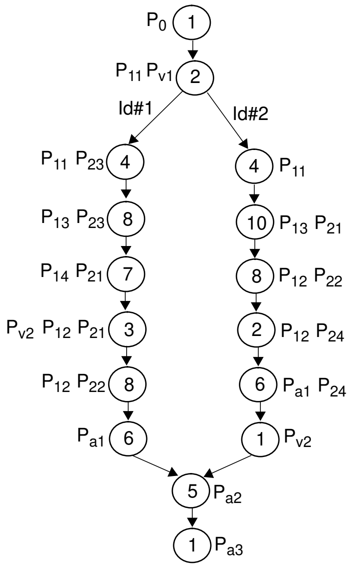


Fig. 2: The conditional schedule generated for the example given in Fig. 1

flict with any other places in the ready place set. However, since P_{21} will hold a token when P_{11}'s output transition is fired, and P_{21} and P_{24} are in resource conflict, they must be ranked according to the priority scheme.

Fig. 2 illustrates the conditional schedule generated for the example given in Fig. 1. In the figure, each node represents a schedule step, the number inside the circle represent the operation time of this schedule step and the place names beside the node denotes which operations are assigned to this step. The example contains conditional scheduling, i.e., depending on the outcome of the non-deterministic vision operation P_{v1}, different schedules will be selected. The selection condition is given in Fig. 2 as a label on arcs between schedule step nodes.

After the algorithm is applied to the Petri net controlling sub-system, timing analysis can be done by searching all of the alternative paths from the initial schedule step to the final schedule step. The result can then be used to check whether any of them violates the timing constraints.

5. Conclusions

We have described an approach to incorporate timing analysis and scheduling in the design process of real-time systems. The approach is based on an extended timed Petri net modeling technique and allows the designers to use iterative improvement transformations during the design process.

The application of the approach to synthesis of flexible manufacturing system has been discussed in the paper. The basic idea is that the designers will start with a behavioral specification of the system. The specification is modeled by the MTPN notation. By assuming a pre-run-time scheduling strategy, the specification can be viewed as an naive implementation, and timing analysis can be carried out to check whether given timing constraints will be met. If some timing constraints can not be met, additional resource can be introduced. And the timing analysis can be carried out again. This process will be repeated until the final implementation is generated which satisfies all timing/resource constraints.

References

[1] T. P. Baker and A. Shaw, "The cyclic executive model and Ada," *J. Real-Time Systems*, Vol. 1, No. 1, pp. 7-25, 1989.

[2] B. Berthomieu, and M. Diaz, "Modeling and verification of time dependent systems using time Petri nets," *IEEE Trans. on Software Eng.*, Vol. 17, No. 3, pp.259-273, 1991.

[3] M. Kamath, and N. Viswanadham, "Application of Petri net based models in the modeling and analysis of flexible manufacturing systems," *Proc. 1986 IEEE Inter. Conf. on Robotics and Automation*, pp. 312-317, 1986.

[4] R. Kurki-Suonio, "Stepwise design of real-time systems," *IEEE Trans. on Software Eng.*, Vol. 19, No. 1, 56-69, 1993.

[5] D. Y. Lee and F. DiCesare, "FMS scheduling using Petri nets and heuristic search," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1057-1062, 1992.

[6] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. of ACM*, Vol. 20, No. 1, pp. 46-61, 1973.

[7] Z. Peng and A. Törne, "A Petri net based modelling and synthesis technique for real-time systems," *Proc. 5th Euromicro Workshop on Real-Time Systems*, Oulu, Finland, June 22-24, IEEE Computer Society Press, pp.204-209, 1993.

[8] Z. Peng and A. Törne, "A synthesis approach to real-time system design," *Proc. 2nd Swedish National Conf. on Real-Time Systems*, Stockholm, Sweden, Aug. 25-26, 1993.

[9] Z. Peng and K. Kuchcinski, "Automated transformation of algorithms into register-transfer level implementation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 2, pp. 150-166, 1994.

[10] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-hall, Englewood Cliffs, New Jersey, 1981.

[11] L. Sha and J. B. Goodenough, "Real-time scheduling theory and Ada," *IEEE Computer*, Vol. 23, No. 4, pp. 53-62, 1990.

[12] H. M. Shih and T. Sekiguchi, "A timed Petri net and beam search based on-line FMS scheduling system with routing flexibility," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2548-2553, 1991.

[13] A. Törne, "The instruction and control of multi-machine environments," in G. Rzevski (Editor) *Application of Artificial Intelligence in Engineering*, Springer-Verlag, pp.137-152, 1990.

[14] J. Xu and L. Parnas, "On satisfying timing constraints in hard-real-time systems," *IEEE Trans. Software Eng.* Vol.19, No. 1, pp. 70-84, 1993.

6th Euromicro Workshop on Real-Time Systems, 1994, Västerås

Timing Analysis and Conditional Scheduling in a Real-Time System Design Environment

Zebo Peng and Anders Törne

Dept. of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden

Abstract

This paper presents an approach to carry out timing analysis and scheduling in a real-time system design environment for flexible manufacturing systems. It is built based on a formal model for specifying real-time systems using extended timed Petri nets. The designer uses the model to specify a design at a given level. The functional characteristics as well as timing properties of the design can then be analyzed with design evaluation procedures. To facilitate timing analysis, a conditional scheduling strategy has been developed to provide a pre-run-time schedule. The strategy is based on a list scheduling method with several priority schemes. This paper describes the design environment, the timing analysis procedure, and the basic strategy for scheduling.

1. Introduction

When designing complex real-time systems, such as flexible manufacturing systems (FMS), design verification must be carried out in the early design stages as well as in the final implementation phases. In order to guarantee correctness in terms of both functionality and timing, and to explore different design alternatives, methods and tools must be developed to support modeling, verification/simulation, and synthesis of such systems. These methods and tools should also be integrated in a uniform design environment in such a way that timing analysis and design verification can be done at the same time when stepwise refinement is carried out. We have been developing such a design environment at Linköping University [7], [8], [13].

One of the main features of our design environment is that it provides a formal notation to capture both the specification and its implementation as well as the intermediate designs in a *uniform* way. This means that once the specification is formulated and verified, a smooth pass to an efficient implementation can be achieved by exploring different design alternatives using iterative improvement techniques. Since the same modeling technique is used throughout the whole synthesis process, the designer needs to learn only one formal notation and the same analysis/scheduling procedures can be used for designs at different levels of abstractions, resulting in a *unified* design environment.

The modeling technique we have developed is based on timed Petri nets and used to model the basic operations, concurrency, synchronization, mutual exclusion, and sequential relations of the basic operations of real-time systems [7]. The Petri net based model is first used to capture the requirement specification of an FMS. Timing analysis can then be carried out on this Petri net based model. Since basic tasks of an FMS usually share resources, a scheduling strategy must be used whenever more than one task is trying to make use the same resource at the same time. Even though our modeling technique is not bound to any particular scheduling approach, the scheduling algorithm described in this paper is based on a non-preemptive pre-run-time scheduling strategy.

Our approach supports design of real-time systems that are *non-deterministic* in terms of operation sequences and operation time [7]. Therefore, the concept of *conditional* scheduling is introduced. It means that the schedule is not fixed but *relative* to a set of conditions which are set to be true or false depending on the real-time environment or outcomes of some operations at run-time (for example, sensor-generated information can be used to select among alternative operation sequences in an FMS). Therefore, a run-time control mechanism is needed to generate the final schedule, and our scheduling approach combines the strength of pre-run-time scheduling and dynamic scheduling.

This work was supported by the Swedish National Board for Industrial and Technical Development (NUTEK) and the Center for Industrial Information Technology (CENIIT) at Linköping University.

The issues of timing analysis and scheduling of real-time systems have been extensively studied and many algorithms have been reported in the literature [1], [2], [6], [11], [14]. Most of the previous works in scheduling model a real-time system as a set of periodic and/or aperiodic tasks with deadlines. For example, with the fixed priority approach [6], a system is modeled as a set of periodic, preemptible, and independent tasks, each with a fixed period and priority. A scheduling algorithm simply selects the highest priority task to be executed. And whenever a task with a higher priority is ready for execution, the task being executed will be preempted. Since FMSs are, in nature, event-driven, and the set of tasks to be executed in an FMS are highly dependent upon each other, a different scheduling model must be used. Petri nets have been used to formulate scheduling problems with routing flexibility and shared resources [5], [12]. In [12], for example, an on-line FMS scheduling system based on a timed Petri net representation is presented. It solves scheduling conflicts using a beam search based decision procedure which constructs partial schedules within the beam-depth and evaluates them to choose the best one. Petri nets have also been widely used in modeling real-time systems and development of synthesis techniques for real-time systems; see, for example, [2], [3] and [10].

This paper describes mainly a pre-run-time based scheduling strategy for FMS applications for timing analysis purpose. It should be pointed out that the main goal of our research is to integrate the timing analysis and scheduling techniques in a *systematic* design environment for modeling and design of complex real-time systems. Section 2 will briefly present the basic modeling technique based on timed Petri nets and the basic idea of the systematic design environment. The issues of timing analysis will then be discussed in section 3. Section 4 describes the conditional scheduling strategy and a scheduling algorithm, which is based on list scheduling with three priority schemes.

2. The Modeling Technique and Design Environment

The design methodology we have developed views a real-time system as consisting of a *controlling* sub-system and a *controlled* sub-system [7]. This view can be applied to different levels of abstractions and a complete real-time system is considered as a hierarchy of controlling/controlled sub-systems. In each of the abstraction levels, the controlled sub-system can be viewed as the environment with which the corresponding controlling sub-system interacts. In the context of a flexible manufacturing system, which is defined as a computer-controlled configuration of semi-independent manufacturing cells coupled with a material handling system, the controlling sub-system is the computer hardware together with the real-time software used to send controlling signals to the manufacturing cells. A particular configuration of the manufacturing cells forms a controlled sub-system.

The timed Petri net based model used to represent such a system is called Modified Timed Petri Net (MTPN). In MTPN, a timed Petri net is used to represent the control structure of a real-time system, namely the controlling sub-system. The controlled sub-system is modeled as an *activity diagram*. It specifies the basic components and the possible activities as well as the initial state of a controlled sub-system. The system components are modeled as a set of resources. Each resource has a finite set of possible states. A resource/state pair defines a vertex of the activity diagram. When the vertex is marked, the resource is currently in the corresponding state. An activity, or operation, is represented by an arc in the activity diagram and controlled by a set of places in the control Petri net. When one of the control places holds a token, the activity will be activated. As the result of the activity, the state of some resources of the modeled system will change. If the destination vertex set of an activity arc has only one element, the activity will result in a deterministic state transition. Otherwise, one of the destination vertices of the arc will be marked as the result of the activity, and the choice is *non-deterministic* [7].

In the MTPN model, timing information is attached to the Petri net places as well as the arcs of the activity diagram. The time attached to an arc models the time needed to complete an activity in the controlled sub-system. *Non-deterministic* characteristics of operation time will be handled by stochastic models with a upper bound and a lower bound. The time attached to a place is, however, a constant, which models the time needed to hold a control signal or the delay time needed to keep the controlled sub-system at certain state. Tokens arriving at the places must stay there for the specified time before they can be used to fire transitions. In the case when a place controls a set of activities which must be completed during the time when the control signal is on, the time attached to the place must be larger than or equal to the worst case operation time of the set of activities. In MTPN, it is assumed that transition firing is instantaneous and represents initiation or termination of activities.

We will use the following FMS example to illustrate the MTPN model, and later the timing analysis problem and the scheduling algorithm. The example shows a machining and assembly process. Three parts are to be machined and assembled — Part A, Part B and Part C. The materials for Part C may arrive in two variants (Id#1 or Id#2) and its machining operations will depend upon the

type of arrived materials which is identified by a vision system. Table 1 gives the basic required operation sequences for the individual parts and Fig. 1(a) shows the MTPN controlling sub-system denoting basically the partial order which must be satisfied by any schedules of this FMS.

Table 1: Process plans for the different parts

	1st op.	2nd op.	3rd op.
Part A	P_{11}	P_{21}	P_{12}
Part B	P_{13}	P_{22}	
Part C#1	P_{23}	P_{14}	
Part C#2	P_{24}		

The FMS example consists of two machines, a assembler and a vision sensor, together with some underlying material handling mechanism. The preliminary allocation of operations to the available resources is captured by the activity diagram given in Fig. 1(b) as the controlled subsystem. The arcs in the activity diagram denote the activities (operations) and are labelled with the controlling Petri net place which indicates when the operation will be activated. For example, when P_0 holds a token, the vision sensor will be activated and enter the *Ready* state. Note that P_0 is also the *initial* place of this example, and will hold a token initially.

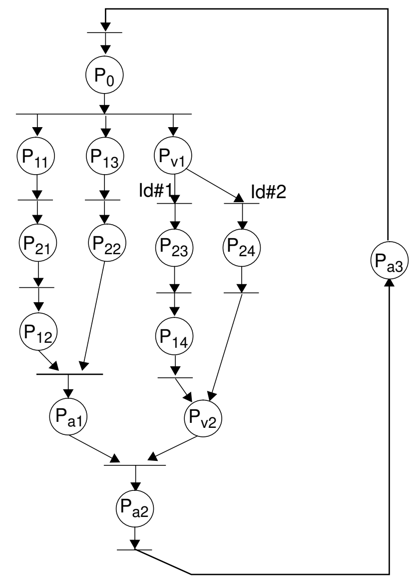
To complete the specification, timing information must be provided. Table 2 gives the operation times associated with the places which denote the times needed for the corresponding operations.

Table 2: Operation times for the places

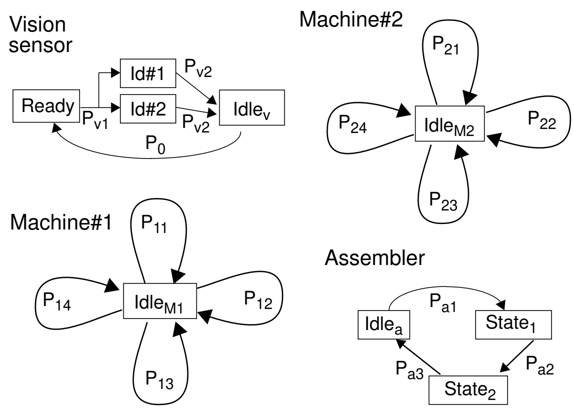
Operation	Time
P_{11}	6
P_{12}	10
P_{13}	8
P_{14}	7
P_{21}	10
P_{22}	8
P_{23}	12

Operation	Time
P_{24}	8
P_{v1}	2
P_{v2}	1
P_{a1}	6
P_{a2}	5
P_{a3}	1
P_0	1

In this FMS example, the two machines are general purpose in the sense that they can be used for different machining of the different parts, which is indicated by the several activities attached to them in Fig. 1(b). If more than one activity associated with the same machine (or any resource) will be activated at the same time, a scheduling technique must be used to decide which one is to perform first. The schedule must meet the timing constraints imposed on the execution cycle of the Petri net and other additional requirements. Before we discuss how this is



a) Controlling sub-system



b) Controlled sub-system

Fig. 1: A machining and assemble process in MTPN

achieved, we would like to briefly discuss the design process supported by our approach.

We assume that the designers start the design process by using the MTPN model to capture the basic requirement of the system to be designed, as in the case of the above example. They can then use Petri net analysis or simulation technique to verify the basic properties of the modeled system. For example, by generating the reachability graph of the Petri net, safeness property can be checked [7]. Once the functionality and timing property of the design is verified, the designers can proceed with the refinement of each activity in the controlled sub-system into the next level MTPN. The timing model associated with the activity will then become an end-to-end timing constraint for the refined MTPN model. In the case when the time attached to a control place must be larger than the

controlled sub-system's associated activities, the lower level controlling sub-system must guarantee that its controlled sub-system will finish its operations within the specified time. If this requirement cannot be satisfied, two solutions can be used. The first one is to change the scheduling strategy. The second one is to introduce more resources into the system and thus increase its parallelism. The implication of these two solutions will be discussed in the next two sections.

The designers can also choose to use the bottom-up approach. That is, they start with the basic level of design and continue with the higher level by considering the lower level design as an activity to be used to build the higher level MTPN model. In this case, the timing analysis procedure will be used to analyze the timing characteristics of the current MTPN, which will become the timing model of the high-level activity.

3. Timing Analysis

A timing constraint can formally be expressed as the time needed for a token to travel from an initial place to a final place in the MTPN notation. This implies an end-to-end deadline — we are interested in whether the whole set of operations will be completed before a given time, not the individual operations' deadlines. Such a constraint is usually introduced due to the timing requirement imposed by the external environment or a higher abstraction/design level in the design hierarchy, as discussed in section 2. In the latter case, the initial place represents the activation of an activity in a higher design level and the final place the completion of this activity. For example, in the example given in Fig. 1, a timing constraint may be imposed on the time for a token to flow from P_0 to P_{a3}, which indicates the completion of a product cycle of this machining and assemble process. This constraint is required by a higher level MTPN model since this manufacturing process is a part of a more complicated FMS.

The same notion of timing constraints can also be used to express other timing requirements in the FMS. For example, we can assume that there exists an extra timing constraint between P_{21} and P_{a1} (from the time when the operation controlled by P_{21} terminates to the time when the operation controlled by P_{a1} starts, to be more precise). This is due to that once the second machining operation of part A is done, it must be assembled within the given time interval <5, 15> (this could, for example, represent the time interval the temperature of the part is in the appropriate range).

There are two basic approaches for timing analysis which can be used by the designers. The first approach is based on the concept of a *critical path* between the two places, i.e., the longest path needed to finish the set of operations initialized by the starting place and terminating at the finishing place. A critical path thus consists of a sequence of control places which dominates the time needed for a token to flow from the starting place to the finishing place. It is analogous to critical paths in hardware systems and similar analysis algorithms can be adapted for this purpose [9]. In the example given in Fig. 1, the critical path consists of P_0, P_{11}, P_{21}, P_{12}, P_{a1}, P_{a2} and P_{a3}, and has a length of 39 time units. The critical path length is however a lower-bound of the needed execution time, which can only be achieved when it is assumed that whenever a token is deposited into a place, the execution of its corresponding activities will start. This means that maximal parallelism is supported and, in the FMS example, at least three machines are needed.

When limited amount of resources is available, a scheduling algorithm can be used to resolve conflict when simultaneous access to resources is not possible. The timing analysis algorithm must then take into account the scheduling strategy, which will be discussed in the next section. If we do not want to enforce any specific scheduling strategy, a mutual exclusion construct can be introduced to the set of places which can hold tokens at the same time and their corresponding activities share the same resource. In this way, these places will not longer be able to hold token simultaneously and only one resource is sufficient for the set of operations. We can then use critical path analysis method to find the longest path of operations and check whether it violates the timing constraint.

With this method, we assume that if several operations are enabled (i.e., ready for execution if the needed resource is available) at the same time, they can be executed in *any* order. This often results in the situation that the critical path length is longer than the maximal execution time which is allowed. In this case, additional partial order can be introduced into the Petri net to enforce a specific order in which the places receive tokens. For example, in the FMS example given in Fig. 1, an extra place and transition can be introduced before P_{13} by the designers. In this way, P_{11} will get a token first, which represents a decision to schedule P_{11}'s operation before P_{13}'s. This manipulation of the design is called a *transformation*. Transformations can also be applied to change the resource allocation. For example, some of the operations bound to Machine#1 can be moved to execute on Machine#2, and vice versa, in our example. With a sequence of such transformations, the designers explore the design space by enforcing different scheduling/resource allocation strategies.

If the above method does not work well or an automatic synthesis approach is desired, the second approach for timing analysis should be used, which assumes that a

scheduling strategy, as to be described in the next section, is used before timing analysis is performed.

4. The Conditional Scheduling Strategy

The problem of scheduling in the MTPN model arises mainly when several places hold tokens simultaneously and their corresponding activities share the same resources. For example, in the example given in Fig. 1, P_{11} and P_{13} will hold tokens simultaneously when their input transition is fired. Since both operations are mapped onto Machine#1, a scheduling decision must be made.

Scheduling is also needed when several transitions are enabled at the same time, but cannot be fired simultaneously. In our design environment, transitions are mainly used to model the changes of system states, which do not take time. In real implementation, the transition firing will take some small amount of time to execute and may share the same processor due to the need of condition checking. However, we assume that the transition firing times are very small compared to activity time and, therefore, negligible. We will consequently not consider the scheduling problem due to simultaneous transition firing.

In an FMS, most activities represent manufacturing processes which cannot be preempted. Therefore, we assume a non-preemptive scheduling model. Since a non-preemptive scheduling model is used, a dynamic scheduling approach may fail a timing constraint even if the resource utilization is not very high [14]. For example, a process *P* can block a time-critical process *Q* because *P* is released first and *Q* cannot preempt *P* even if *Q* will miss its deadline by waiting first for *P* to finish. We have therefore also assumed a pre-run-time scheduling approach.

In the MTPN model, the release of activities (i.e., the enabling of activity arcs by Petri net places holding tokens) can be non-deterministic (i.e., depending on conditions given by the environment), and the activity time can also be non-deterministic. Further the activity time can also be un-bounded, since the MTPN model allows self-loops in the Petri net [10]. Therefore, we cannot have a pure pre-run-time schedule.

To solve the above problem, we define *conditional scheduling* as a schedule class which comprises a set of basic schedules. Which basic schedule is really used will be decided at run-time based on a set of conditions. This means we must have a run-time control mechanism, as discussed before. The run-time arbitrator (dispatch), however, is very simple, since it only needs to check the conditions. This arbitrator can be captured by the notation of finite state machines and implemented with electronics circuitry [9].

We have selected to use a simple list-based scheduling algorithm for generating the final conditional schedule. The algorithm maintains a list of ready places, i.e., places with tokens. The places are ordered according to a three-level priority scheme. The first level to ne considered is related to deadlines given as extra local timing requirement, such as the one given between P_{21} and P_{a1} in the FMS example. The earliest deadline first approach is used to order the operations in this priority level.

The second level is defined by the distance from the place to the final place, i.e., how much time it is needed to pass a token from the current place to the final place. The longer the distance the higher the priority. This reflects the strategy to consider the most time-consuming set of operations first, since the main objective is to meet the end-to-end deadline. The second level will be considered when the first level does not resolve the conflict.

If the second priority level gives again equal value for the places in conflict, the third level priority will be used, which is defined by the time needed for the completion of the current activities. The shorter the operation time, the higher its priority.

The basic function of the scheduling algorithm is to generate the conditional reachability graph, using the priority scheme to resolve resource conflicts whenever needed. The algorithm is based on the Petri net reachability graph generation algorithm [10] with the following modifications. First of all, each reachable marking is associated with a time indicating how much time this marking will hold. The algorithm must keep track of the remaining operation times of the places when it generates the next marking.

The second modification is that when generating the next marking, a set of transitions will be fired simultaneously if possible instead of one single transition. Further only the maximally firable sets of transitions will be considered. A *firable* transition set is a set of transitions which are enabled by the current marking and the intersection of their guarding conditions are not always false. A *maximally* firable transition set is defined as a firable transition set that is not proper subset of any of the firable transition sets of the current marking. The maximally firable transition set concept is introduced to get rid of redundant markings in the generated schedule so as to reduce its complexity. A similar concept has also been used in digital hardware synthesis [9].

Finally, a one-step look ahead mechanism is used to check whether any of the ready places will be in resource-conflict with the places which will hold tokens when the enabled transitions are fired. If this is the case, the conflict will also be resolved by the priority scheme. For example, at the marking $\{P_{11}, P_{13}, P_{24}\}$, P_{24} is not in resource con-

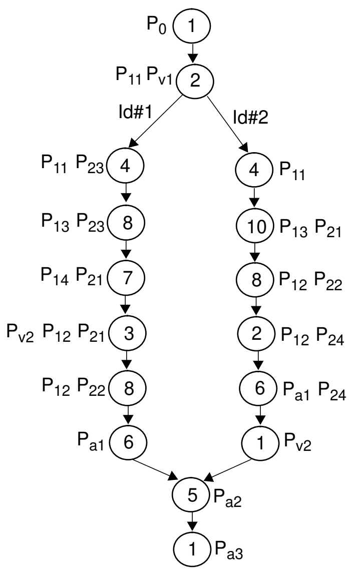


Fig. 2: The conditional schedule generated for the example given in Fig. 1

flict with any other places in the ready place set. However, since P_{21} will hold a token when P_{11}'s output transition is fired, and P_{21} and P_{24} are in resource conflict, they must be ranked according to the priority scheme.

Fig. 2 illustrates the conditional schedule generated for the example given in Fig. 1. In the figure, each node represents a schedule step, the number inside the circle represent the operation time of this schedule step and the place names beside the node denotes which operations are assigned to this step. The example contains conditional scheduling, i.e., depending on the outcome of the non-deterministic vision operation P_{v1}, different schedules will be selected. The selection condition is given in Fig. 2 as a label on arcs between schedule step nodes.

After the algorithm is applied to the Petri net controlling sub-system, timing analysis can be done by searching all of the alternative paths from the initial schedule step to the final schedule step. The result can then be used to check whether any of them violates the timing constraints.

5. Conclusions

We have described an approach to incorporate timing analysis and scheduling in the design process of real-time systems. The approach is based on an extended timed Petri net modeling technique and allows the designers to use iterative improvement transformations during the design process.

The application of the approach to synthesis of flexible manufacturing system has been discussed in the paper. The basic idea is that the designers will start with a behavioral specification of the system. The specification is modeled by the MTPN notation. By assuming a pre-run-time scheduling strategy, the specification can be viewed as an naive implementation, and timing analysis can be carried out to check whether given timing constraints will be met. If some timing constraints can not be met, additional resource can be introduced. And the timing analysis can be carried out again. This process will be repeated until the final implementation is generated which satisfies all timing/resource constraints.

References

[1] T. P. Baker and A. Shaw, "The cyclic executive model and Ada," *J. Real-Time Systems*, Vol. 1, No. 1, pp. 7-25, 1989.

[2] B. Berthomieu, and M. Diaz, "Modeling and verification of time dependent systems using time Petri nets," *IEEE Trans. on Software Eng.*, Vol. 17, No. 3, pp.259-273, 1991.

[3] M. Kamath, and N. Viswanadham, "Application of Petri net based models in the modeling and analysis of flexible manufacturing systems," *Proc. 1986 IEEE Inter. Conf. on Robotics and Automation*, pp. 312-317, 1986.

[4] R. Kurki-Suonio, "Stepwise design of real-time systems," *IEEE Trans. on Software Eng.*, Vol. 19, No. 1, 56-69, 1993.

[5] D. Y. Lee and F. DiCesare, "FMS scheduling using Petri nets and heuristic search," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1057-1062, 1992.

[6] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. of ACM*, Vol. 20, No. 1, pp. 46-61, 1973.

[7] Z. Peng and A. Törne, "A Petri net based modelling and synthesis technique for real-time systems," *Proc. 5th Euromicro Workshop on Real-Time Systems*, Oulu, Finland, June 22-24, IEEE Computer Society Press, pp.204-209, 1993.

[8] Z. Peng and A. Törne, "A synthesis approach to real-time system design," *Proc. 2nd Swedish National Conf. on Real-Time Systems*, Stockholm, Sweden, Aug. 25-26, 1993.

[9] Z. Peng and K. Kuchcinski, "Automated transformation of algorithms into register-transfer level implementation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 2, pp. 150-166, 1994.

[10] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-hall, Englewood Cliffs, New Jersey, 1981.

[11] L. Sha and J. B. Goodenough, "Real-time scheduling theory and Ada," *IEEE Computer*, Vol. 23, No. 4, pp. 53-62, 1990.

[12] H. M. Shih and T. Sekiguchi, "A timed Petri net and beam search based on-line FMS scheduling system with routing flexibility," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2548-2553, 1991.

[13] A. Törne, "The instruction and control of multi-machine environments," in G. Rzevski (Editor) *Application of Artificial Intelligence in Engineering*, Springer-Verlag, pp.137-152, 1990.

[14] J. Xu and L. Parnas, "On satisfying timing constraints in hard-real-time systems," *IEEE Trans. Software Eng.* Vol.19, No. 1, pp. 70-84, 1993.

6th Euromicro Workshop on Real-Time Systems, 1994, Västerås

Timing Analysis and Conditional Scheduling in a Real-Time System Design Environment

Zebo Peng and Anders Törne

Dept. of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden

Abstract

This paper presents an approach to carry out timing analysis and scheduling in a real-time system design environment for flexible manufacturing systems. It is built based on a formal model for specifying real-time systems using extended timed Petri nets. The designer uses the model to specify a design at a given level. The functional characteristics as well as timing properties of the design can then be analyzed with design evaluation procedures. To facilitate timing analysis, a conditional scheduling strategy has been developed to provide a pre-run-time schedule. The strategy is based on a list scheduling method with several priority schemes. This paper describes the design environment, the timing analysis procedure, and the basic strategy for scheduling.

1. Introduction

When designing complex real-time systems, such as flexible manufacturing systems (FMS), design verification must be carried out in the early design stages as well as in the final implementation phases. In order to guarantee correctness in terms of both functionality and timing, and to explore different design alternatives, methods and tools must be developed to support modeling, verification/simulation, and synthesis of such systems. These methods and tools should also be integrated in a uniform design environment in such a way that timing analysis and design verification can be done at the same time when stepwise refinement is carried out. We have been developing such a design environment at Linköping University [7], [8], [13].

One of the main features of our design environment is that it provides a formal notation to capture both the specification and its implementation as well as the intermediate designs in a *uniform* way. This means that once the specification is formulated and verified, a smooth pass to an efficient implementation can be achieved by exploring different design alternatives using iterative improvement techniques. Since the same modeling technique is used throughout the whole synthesis process, the designer needs to learn only one formal notation and the same analysis/scheduling procedures can be used for designs at different levels of abstractions, resulting in a *unified* design environment.

The modeling technique we have developed is based on timed Petri nets and used to model the basic operations, concurrency, synchronization, mutual exclusion, and sequential relations of the basic operations of real-time systems [7]. The Petri net based model is first used to capture the requirement specification of an FMS. Timing analysis can then be carried out on this Petri net based model. Since basic tasks of an FMS usually share resources, a scheduling strategy must be used whenever more than one task is trying to make use the same resource at the same time. Even though our modeling technique is not bound to any particular scheduling approach, the scheduling algorithm described in this paper is based on a non-preemptive pre-run-time scheduling strategy.

Our approach supports design of real-time systems that are *non-deterministic* in terms of operation sequences and operation time [7]. Therefore, the concept of *conditional* scheduling is introduced. It means that the schedule is not fixed but *relative* to a set of conditions which are set to be true or false depending on the real-time environment or outcomes of some operations at run-time (for example, sensor-generated information can be used to select among alternative operation sequences in an FMS). Therefore, a run-time control mechanism is needed to generate the final schedule, and our scheduling approach combines the strength of pre-run-time scheduling and dynamic scheduling.

This work was supported by the Swedish National Board for Industrial and Technical Development (NUTEK) and the Center for Industrial Information Technology (CENIIT) at Linköping University.

The issues of timing analysis and scheduling of real-time systems have been extensively studied and many algorithms have been reported in the literature [1], [2], [6], [11], [14]. Most of the previous works in scheduling model a real-time system as a set of periodic and/or aperiodic tasks with deadlines. For example, with the fixed priority approach [6], a system is modeled as a set of periodic, preemptible, and independent tasks, each with a fixed period and priority. A scheduling algorithm simply selects the highest priority task to be executed. And whenever a task with a higher priority is ready for execution, the task being executed will be preempted. Since FMSs are, in nature, event-driven, and the set of tasks to be executed in an FMS are highly dependent upon each other, a different scheduling model must be used. Petri nets have been used to formulate scheduling problems with routing flexibility and shared resources [5], [12]. In [12], for example, an on-line FMS scheduling system based on a timed Petri net representation is presented. It solves scheduling conflicts using a beam search based decision procedure which constructs partial schedules within the beam-depth and evaluates them to choose the best one. Petri nets have also been widely used in modeling real-time systems and development of synthesis techniques for real-time systems; see, for example, [2], [3] and [10].

This paper describes mainly a pre-run-time based scheduling strategy for FMS applications for timing analysis purpose. It should be pointed out that the main goal of our research is to integrate the timing analysis and scheduling techniques in a *systematic* design environment for modeling and design of complex real-time systems. Section 2 will briefly present the basic modeling technique based on timed Petri nets and the basic idea of the systematic design environment. The issues of timing analysis will then be discussed in section 3. Section 4 describes the conditional scheduling strategy and a scheduling algorithm, which is based on list scheduling with three priority schemes.

2. The Modeling Technique and Design Environment

The design methodology we have developed views a real-time system as consisting of a *controlling* sub-system and a *controlled* sub-system [7]. This view can be applied to different levels of abstractions and a complete real-time system is considered as a hierarchy of controlling/controlled sub-systems. In each of the abstraction levels, the controlled sub-system can be viewed as the environment with which the corresponding controlling sub-system interacts. In the context of a flexible manufacturing system, which is defined as a computer-controlled configuration of semi-independent manufacturing cells coupled with a material handling system, the controlling sub-system is the computer hardware together with the real-time software used to send controlling signals to the manufacturing cells. A particular configuration of the manufacturing cells forms a controlled sub-system.

The timed Petri net based model used to represent such a system is called Modified Timed Petri Net (MTPN). In MTPN, a timed Petri net is used to represent the control structure of a real-time system, namely the controlling sub-system. The controlled sub-system is modeled as an *activity diagram*. It specifies the basic components and the possible activities as well as the initial state of a controlled sub-system. The system components are modeled as a set of resources. Each resource has a finite set of possible states. A resource/state pair defines a vertex of the activity diagram. When the vertex is marked, the resource is currently in the corresponding state. An activity, or operation, is represented by an arc in the activity diagram and controlled by a set of places in the control Petri net. When one of the control places holds a token, the activity will be activated. As the result of the activity, the state of some resources of the modeled system will change. If the destination vertex set of an activity arc has only one element, the activity will result in a deterministic state transition. Otherwise, one of the destination vertices of the arc will be marked as the result of the activity, and the choice is *non-deterministic* [7].

In the MTPN model, timing information is attached to the Petri net places as well as the arcs of the activity diagram. The time attached to an arc models the time needed to complete an activity in the controlled sub-system. *Non-deterministic* characteristics of operation time will be handled by stochastic models with a upper bound and a lower bound. The time attached to a place is, however, a constant, which models the time needed to hold a control signal or the delay time needed to keep the controlled sub-system at certain state. Tokens arriving at the places must stay there for the specified time before they can be used to fire transitions. In the case when a place controls a set of activities which must be completed during the time when the control signal is on, the time attached to the place must be larger than or equal to the worst case operation time of the set of activities. In MTPN, it is assumed that transition firing is instantaneous and represents initiation or termination of activities.

We will use the following FMS example to illustrate the MTPN model, and later the timing analysis problem and the scheduling algorithm. The example shows a machining and assembly process. Three parts are to be machined and assembled — Part A, Part B and Part C. The materials for Part C may arrive in two variants (Id#1 or Id#2) and its machining operations will depend upon the

type of arrived materials which is identified by a vision system. Table 1 gives the basic required operation sequences for the individual parts and Fig. 1(a) shows the MTPN controlling sub-system denoting basically the partial order which must be satisfied by any schedules of this FMS.

Table 1: Process plans for the different parts

	1st op.	2nd op.	3rd op.
Part A	P_{11}	P_{21}	P_{12}
Part B	P_{13}	P_{22}	
Part C#1	P_{23}	P_{14}	
Part C#2	P_{24}		

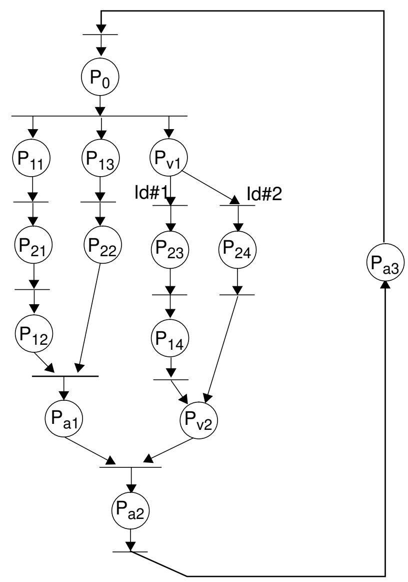
The FMS example consists of two machines, a assembler and a vision sensor, together with some underlying material handling mechanism. The preliminary allocation of operations to the available resources is captured by the activity diagram given in Fig. 1(b) as the controlled subsystem. The arcs in the activity diagram denote the activities (operations) and are labelled with the controlling Petri net place which indicates when the operation will be activated. For example, when P_0 holds a token, the vision sensor will be activated and enter the *Ready* state. Note that P_0 is also the *initial* place of this example, and will hold a token initially.

To complete the specification, timing information must be provided. Table 2 gives the operation times associated with the places which denote the times needed for the corresponding operations.

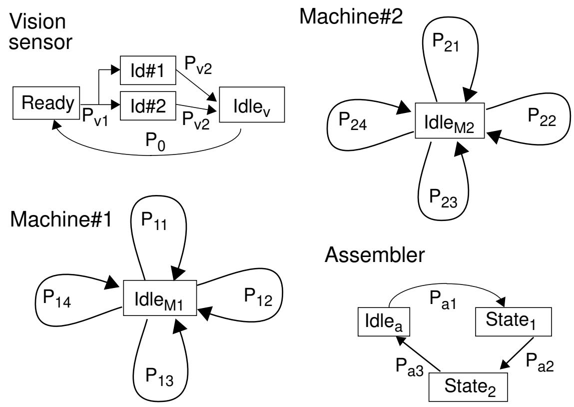
Table 2: Operation times for the places

Operation	Time	Operation	Time
P_{11}	6	P_{24}	8
P_{12}	10	P_{v1}	2
P_{13}	8	P_{v2}	1
P_{14}	7	P_{a1}	6
P_{21}	10	P_{a2}	5
P_{22}	8	P_{a3}	1
P_{23}	12	P_0	1

In this FMS example, the two machines are general purpose in the sense that they can be used for different machining of the different parts, which is indicated by the several activities attached to them in Fig. 1(b). If more than one activity associated with the same machine (or any resource) will be activated at the same time, a scheduling technique must be used to decide which one is to perform first. The schedule must meet the timing constraints imposed on the execution cycle of the Petri net and other additional requirements. Before we discuss how this is



a) Controlling sub-system



b) Controlled sub-system

Fig. 1: A machining and assemble process in MTPN

achieved, we would like to briefly discuss the design process supported by our approach.

We assume that the designers start the design process by using the MTPN model to capture the basic requirement of the system to be designed, as in the case of the above example. They can then use Petri net analysis or simulation technique to verify the basic properties of the modeled system. For example, by generating the reachability graph of the Petri net, safeness property can be checked [7]. Once the functionality and timing property of the design is verified, the designers can proceed with the refinement of each activity in the controlled sub-system into the next level MTPN. The timing model associated with the activity will then become an end-to-end timing constraint for the refined MTPN model. In the case when the time attached to a control place must be larger than the

controlled sub-system's associated activities, the lower level controlling sub-system must guarantee that its controlled sub-system will finish its operations within the specified time. If this requirement cannot be satisfied, two solutions can be used. The first one is to change the scheduling strategy. The second one is to introduce more resources into the system and thus increase its parallelism. The implication of these two solutions will be discussed in the next two sections.

The designers can also choose to use the bottom-up approach. That is, they start with the basic level of design and continue with the higher level by considering the lower level design as an activity to be used to build the higher level MTPN model. In this case, the timing analysis procedure will be used to analyze the timing characteristics of the current MTPN, which will become the timing model of the high-level activity.

3. Timing Analysis

A timing constraint can formally be expressed as the time needed for a token to travel from an initial place to a final place in the MTPN notation. This implies an end-to-end deadline — we are interested in whether the whole set of operations will be completed before a given time, not the individual operations' deadlines. Such a constraint is usually introduced due to the timing requirement imposed by the external environment or a higher abstraction/design level in the design hierarchy, as discussed in section 2. In the latter case, the initial place represents the activation of an activity in a higher design level and the final place the completion of this activity. For example, in the example given in Fig. 1, a timing constraint may be imposed on the time for a token to flow from P_0 to P_{a3}, which indicates the completion of a product cycle of this machining and assemble process. This constraint is required by a higher level MTPN model since this manufacturing process is a part of a more complicated FMS.

The same notion of timing constraints can also be used to express other timing requirements in the FMS. For example, we can assume that there exists an extra timing constraint between P_{21} and P_{a1} (from the time when the operation controlled by P_{21} terminates to the time when the operation controlled by P_{a1} starts, to be more precise). This is due to that once the second machining operation of part A is done, it must be assembled within the given time interval <5, 15> (this could, for example, represent the time interval the temperature of the part is in the appropriate range).

There are two basic approaches for timing analysis which can be used by the designers. The first approach is based on the concept of a *critical path* between the two places, i.e., the longest path needed to finish the set of operations initialized by the starting place and terminating at the finishing place. A critical path thus consists of a sequence of control places which dominates the time needed for a token to flow from the starting place to the finishing place. It is analogous to critical paths in hardware systems and similar analysis algorithms can be adapted for this purpose [9]. In the example given in Fig. 1, the critical path consists of P_0, P_{11}, P_{21}, P_{12}, P_{a1}, P_{a2} and P_{a3}, and has a length of 39 time units. The critical path length is however a lower-bound of the needed execution time, which can only be achieved when it is assumed that whenever a token is deposited into a place, the execution of its corresponding activities will start. This means that maximal parallelism is supported and, in the FMS example, at least three machines are needed.

When limited amount of resources is available, a scheduling algorithm can be used to resolve conflict when simultaneous access to resources is not possible. The timing analysis algorithm must then take into account the scheduling strategy, which will be discussed in the next section. If we do not want to enforce any specific scheduling strategy, a mutual exclusion construct can be introduced to the set of places which can hold tokens at the same time and their corresponding activities share the same resource. In this way, these places will not longer be able to hold token simultaneously and only one resource is sufficient for the set of operations. We can then use critical path analysis method to find the longest path of operations and check whether it violates the timing constraint.

With this method, we assume that if several operations are enabled (i.e., ready for execution if the needed resource is available) at the same time, they can be executed in *any* order. This often results in the situation that the critical path length is longer than the maximal execution time which is allowed. In this case, additional partial order can be introduced into the Petri net to enforce a specific order in which the places receive tokens. For example, in the FMS example given in Fig. 1, an extra place and transition can be introduced before P_{13} by the designers. In this way, P_{11} will get a token first, which represents a decision to schedule P_{11}'s operation before P_{13}'s. This manipulation of the design is called a *transformation*. Transformations can also be applied to change the resource allocation. For example, some of the operations bound to Machine#1 can be moved to execute on Machine#2, and vice versa, in our example. With a sequence of such transformations, the designers explore the design space by enforcing different scheduling/resource allocation strategies.

If the above method does not work well or an automatic synthesis approach is desired, the second approach for timing analysis should be used, which assumes that a

scheduling strategy, as to be described in the next section, is used before timing analysis is performed.

4. The Conditional Scheduling Strategy

The problem of scheduling in the MTPN model arises mainly when several places hold tokens simultaneously and their corresponding activities share the same resources. For example, in the example given in Fig. 1, P_{11} and P_{13} will hold tokens simultaneously when their input transition is fired. Since both operations are mapped onto Machine#1, a scheduling decision must be made.

Scheduling is also needed when several transitions are enabled at the same time, but cannot be fired simultaneously. In our design environment, transitions are mainly used to model the changes of system states, which do not take time. In real implementation, the transition firing will take some small amount of time to execute and may share the same processor due to the need of condition checking. However, we assume that the transition firing times are very small compared to activity time and, therefore, negligible. We will consequently not consider the scheduling problem due to simultaneous transition firing.

In an FMS, most activities represent manufacturing processes which cannot be preempted. Therefore, we assume a non-preemptive scheduling model. Since a non-preemptive scheduling model is used, a dynamic scheduling approach may fail a timing constraint even if the resource utilization is not very high [14]. For example, a process *P* can block a time-critical process *Q* because *P* is released first and *Q* cannot preempt *P* even if *Q* will miss its deadline by waiting first for *P* to finish. We have therefore also assumed a pre-run-time scheduling approach.

In the MTPN model, the release of activities (i.e., the enabling of activity arcs by Petri net places holding tokens) can be non-deterministic (i.e., depending on conditions given by the environment), and the activity time can also be non-deterministic. Further the activity time can also be un-bounded, since the MTPN model allows self-loops in the Petri net [10]. Therefore, we cannot have a pure pre-run-time schedule.

To solve the above problem, we define *conditional scheduling* as a schedule class which comprises a set of basic schedules. Which basic schedule is really used will be decided at run-time based on a set of conditions. This means we must have a run-time control mechanism, as discussed before. The run-time arbitrator (dispatch), however, is very simple, since it only needs to check the conditions. This arbitrator can be captured by the notation of finite state machines and implemented with electronics circuitry [9].

We have selected to use a simple list-based scheduling algorithm for generating the final conditional schedule. The algorithm maintains a list of ready places, i.e., places with tokens. The places are ordered according to a three-level priority scheme. The first level to ne considered is related to deadlines given as extra local timing requirement, such as the one given between P_{21} and P_{a1} in the FMS example. The earliest deadline first approach is used to order the operations in this priority level.

The second level is defined by the distance from the place to the final place, i.e., how much time it is needed to pass a token from the current place to the final place. The longer the distance the higher the priority. This reflects the strategy to consider the most time-consuming set of operations first, since the main objective is to meet the end-to-end deadline. The second level will be considered when the first level does not resolve the conflict.

If the second priority level gives again equal value for the places in conflict, the third level priority will be used, which is defined by the time needed for the completion of the current activities. The shorter the operation time, the higher its priority.

The basic function of the scheduling algorithm is to generate the conditional reachability graph, using the priority scheme to resolve resource conflicts whenever needed. The algorithm is based on the Petri net reachability graph generation algorithm [10] with the following modifications. First of all, each reachable marking is associated with a time indicating how much time this marking will hold. The algorithm must keep track of the remaining operation times of the places when it generates the next marking.

The second modification is that when generating the next marking, a set of transitions will be fired simultaneously if possible instead of one single transition. Further only the maximally firable sets of transitions will be considered. A *firable* transition set is a set of transitions which are enabled by the current marking and the intersection of their guarding conditions are not always false. A *maximally* firable transition set is defined as a firable transition set that is not proper subset of any of the firable transition sets of the current marking. The maximally firable transition set concept is introduced to get rid of redundant markings in the generated schedule so as to reduce its complexity. A similar concept has also been used in digital hardware synthesis [9].

Finally, a one-step look ahead mechanism is used to check whether any of the ready places will be in resource-conflict with the places which will hold tokens when the enabled transitions are fired. If this is the case, the conflict will also be resolved by the priority scheme. For example, at the marking $\{P_{11}, P_{13}, P_{24}\}$, P_{24} is not in resource con-

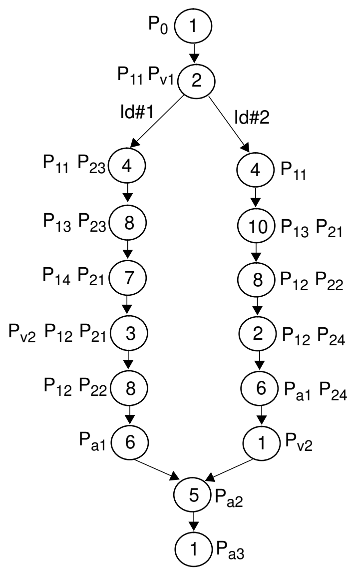


Fig. 2: The conditional schedule generated for the example given in Fig. 1

flict with any other places in the ready place set. However, since P_{21} will hold a token when P_{11}'s output transition is fired, and P_{21} and P_{24} are in resource conflict, they must be ranked according to the priority scheme.

Fig. 2 illustrates the conditional schedule generated for the example given in Fig. 1. In the figure, each node represents a schedule step, the number inside the circle represent the operation time of this schedule step and the place names beside the node denotes which operations are assigned to this step. The example contains conditional scheduling, i.e., depending on the outcome of the non-deterministic vision operation P_{v1}, different schedules will be selected. The selection condition is given in Fig. 2 as a label on arcs between schedule step nodes.

After the algorithm is applied to the Petri net controlling sub-system, timing analysis can be done by searching all of the alternative paths from the initial schedule step to the final schedule step. The result can then be used to check whether any of them violates the timing constraints.

5. Conclusions

We have described an approach to incorporate timing analysis and scheduling in the design process of real-time systems. The approach is based on an extended timed Petri net modeling technique and allows the designers to use iterative improvement transformations during the design process.

The application of the approach to synthesis of flexible manufacturing system has been discussed in the paper. The basic idea is that the designers will start with a behavioral specification of the system. The specification is modeled by the MTPN notation. By assuming a pre-run-time scheduling strategy, the specification can be viewed as an naive implementation, and timing analysis can be carried out to check whether given timing constraints will be met. If some timing constraints can not be met, additional resource can be introduced. And the timing analysis can be carried out again. This process will be repeated until the final implementation is generated which satisfies all timing/resource constraints.

References

[1] T. P. Baker and A. Shaw, "The cyclic executive model and Ada," *J. Real-Time Systems*, Vol. 1, No. 1, pp. 7-25, 1989.

[2] B. Berthomieu, and M. Diaz, "Modeling and verification of time dependent systems using time Petri nets," *IEEE Trans. on Software Eng.*, Vol. 17, No. 3, pp.259-273, 1991.

[3] M. Kamath, and N. Viswanadham, "Application of Petri net based models in the modeling and analysis of flexible manufacturing systems," *Proc. 1986 IEEE Inter. Conf. on Robotics and Automation*, pp. 312-317, 1986.

[4] R. Kurki-Suonio, "Stepwise design of real-time systems," *IEEE Trans. on Software Eng.*, Vol. 19, No. 1, 56-69, 1993.

[5] D. Y. Lee and F. DiCesare, "FMS scheduling using Petri nets and heuristic search," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1057-1062, 1992.

[6] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. of ACM*, Vol. 20, No. 1, pp. 46-61, 1973.

[7] Z. Peng and A. Törne, "A Petri net based modelling and synthesis technique for real-time systems," *Proc. 5th Euromicro Workshop on Real-Time Systems*, Oulu, Finland, June 22-24, IEEE Computer Society Press, pp.204-209, 1993.

[8] Z. Peng and A. Törne, "A synthesis approach to real-time system design," *Proc. 2nd Swedish National Conf. on Real-Time Systems*, Stockholm, Sweden, Aug. 25-26, 1993.

[9] Z. Peng and K. Kuchcinski, "Automated transformation of algorithms into register-transfer level implementation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 2, pp. 150-166, 1994.

[10] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-hall, Englewood Cliffs, New Jersey, 1981.

[11] L. Sha and J. B. Goodenough, "Real-time scheduling theory and Ada," *IEEE Computer*, Vol. 23, No. 4, pp. 53-62, 1990.

[12] H. M. Shih and T. Sekiguchi, "A timed Petri net and beam search based on-line FMS scheduling system with routing flexibility," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2548-2553, 1991.

[13] A. Törne, "The instruction and control of multi-machine environments," in G. Rzevski (Editor) *Application of Artificial Intelligence in Engineering*, Springer-Verlag, pp.137-152, 1990.

[14] J. Xu and L. Parnas, "On satisfying timing constraints in hard-real-time systems," *IEEE Trans. Software Eng.* Vol.19, No. 1, pp. 70-84, 1993.