10

Using Timed Coloured Petri Nets for Modelling, Simulation and Scheduling of Production Systems

Andrzej Bożek
Rzeszow University of Technology, Poland

1. Introduction

The right choice of a system model is the important issue related to developing of production management systems. An abstract representation of the system is needed for e.g. production scheduling or "what-if" scenarios generating. Disjunctive graphs, permutations with repetition as well as mathematical programming relations are the most common used representations. However, these representations make possible to model systems with rather simple and homogenous structures. Different extensions are added to models, as setup and transport times, machine calendars, resource restrictions, etc., to represent other system features. These models are often carefully designed and find application in robust scheduling algorithms. Sometimes, however, the high efficiency of the scheduling is not the primary goal, but a flexible modelling formalism is more important. This chapter proposes a solution dedicated especially for these cases.

Petri nets are successfully used for modelling and scheduling of production systems. The presented solution is based on hierarchical timed coloured Petri nets (HTCPN), the formalism introduced by Jensen. This formalism has been chosen because of its generality and flexibility. Token colouring lets tokens carry information that can be represented by complex data types and can be processed by arc expressions. The hierarchy allows to decompose a structure of a system to nested modules. Multiple instances of an once developed module can be parametrised and placed into a model many times. Timestamps assigned to tokens and the model global clock make possible to simulate the time flow that is especially important for scheduling.

2. Petri nets in production scheduling

Petri nets are often used approach in modelling, simulation, analyzing and scheduling of production systems. The excellent review has been prepared by Tuncel and Bayhan (2007). The authors verified 72 publications that concern using Petri nets in the production scheduling. The works have been differed on the basis of a scheduling approach, a Petri net type and an application area. Four main scheduling approaches have been enumerated: heuristic rule based systems, search algorithms, mathematical programming approaches and meta-heuristics. Very different variants of Petri net formalism are used, there are *stochastic Petri nets* (SPN), *priority nets*, *coloured transition-timed Petri nets*, *object-oriented Petri nets* (OOPN), *generalised symmetric/asymmetric nets* (GSN/GAN), and others. The most authors use Petri nets for modelling and scheduling of *flexible manufacturing systems* (FMS). However, the definition of the FMS is very general and the detailed application can be unknown. The other areas of application are, for example, job shop scheduling, AGV systems, cyclic scheduling, wafer fabrication, demanufacturing systems. In the breakdown of the 72 publications, at least 13 of them refer to timed Petri nets, and at least 7 refer to coloured Petri nets.

The work published by Camurri, Franchi and Gandolfo (1991) is probably one of the first in which both coloured and timed Petri nets have been used for production modelling and scheduling. The authors use coloured tokens to distinguish scheduled jobs. It is used a Petri net variant that assigns enabling times to transitions. These enabling times model processing times of operations. There are also the *executor subsystem* for a net model simulation and the *scheduling subsystem* for scheduling decision making proposed in the work.

Aized (2010) refers to *Coloured Petri Net* (CPN) formalism introduced by Jensen (Jensen & Kristensen, 2009). The author emphasizes the flexibility of CPN in modelling and scheduling of *Discrete Event Dynamical Systems* (DEDS). He discusses the model of a semi-conductor manufacturing system (called *multiple cluster tool system*) that has been implemented with the use of CPN formalism and *CPN Tools* software.

In the work of Zhang, Gu and Song (2008), the Jensen's CPN formalism has been also used for modelling of production systems. The authors place elements that represent batch processing, setup operations and transport operations in a one net structure.

3. Formalism of hierarchical timed coloured Petri nets

There are many variants of the Petri net formalism. Basic Petri nets, so-called *low-level nets* or *P/T nets*, have the syntax based on places, transitions and arcs. In practice, *high-level Petri net* formalisms are often used. These formalisms include additional extensions, e.g. colouring, time representation, special types of arcs, priority of transitions, hierarchy, and others. In this work, the formalism of *Coloured Petri Nets* introduced by Jensen is used (Jensen & Kristensen, 2009). There are two main reasons of that choice:

1. This formalism has an extensive syntax and semantics that makes modelling very comfortable and flexible.
2. There is a software called *CPN Tools* (CPN Tools Homepage, 2011) that fully supports the *Coloured Petri Nets* formalism. *CPN Tools* makes possible to edit, simulate and analyse of Petri net models.

Colouring, time representation and hierarchy representation are the main features of the formalism of *Coloured Petri Nets*. Because of that, this formalism will be denoted shortly as HTCPN (*Hierarchical Timed Coloured Petri Net*) in the remainder of this chapter.

The formal definition of the syntax of HTCPN, except for hierarchy representation, is as follows (Jensen & Kristensen, 2009):

A timed non-hierarchical coloured Petri net is a nine-tuple $CPN_T = (P, T, A, \Sigma, V, C, G, E, I)$ where:

1. P is a finite set of *places*.
2. T is a finite set of *transitions* such that $P \cap T = \emptyset$.
3. $A \subseteq P \times T \cup T \times P$ is a set of direct *arcs.*
4. Σ is a finite set of non-empty *colour sets* (types). Each colour set is either untimed or timed.
5. V is a finite set of *typed variables* such that $Type[v] \in \Sigma$ for all variables $v \in V$.
6. $C : P \rightarrow \Sigma$ is a *colour set function* that assigns a colour set to each place. A place p is timed if $C(p)$ is timed, otherwise p is untimed.
7. $G : T \rightarrow EXPR_V$ is a *guard function* that assigns a guard to each transition t such that $Type[G(t)] = Bool$.
8. $E : A \rightarrow EXPR_V$ is an *arc expression function* that assigns an arc expression to each arc a such that
   - $Type[E(a)] = C(p)_{MS}$ if p is untimed (*MS*: multi set),
   - $Type[E(a)] = C(p)_{TMS}$ if p is timed (*TMS*: timed multi set),
   where p is the place connected to the arc a.
9. $I : P \rightarrow EXPR_\emptyset$ is an *initialisation function* that assigns an initialisation expression to each place p such that
   - $Type[I(p)] = C(p)_{MS}$ if p is untimed,
   - $Type[I(p)] = C(p)_{TMS}$ if p is timed.

The hierarchy feature has been omitted in the definition above, because its strict formulation is quite complex. However, this concept is intuitively very simple. Selected substructures of a net model can be placed in special blocks called *modules*. Any module can be inserted to the net structure many times, where it is represented by a special kind of transition, so-called *substitution transition*.

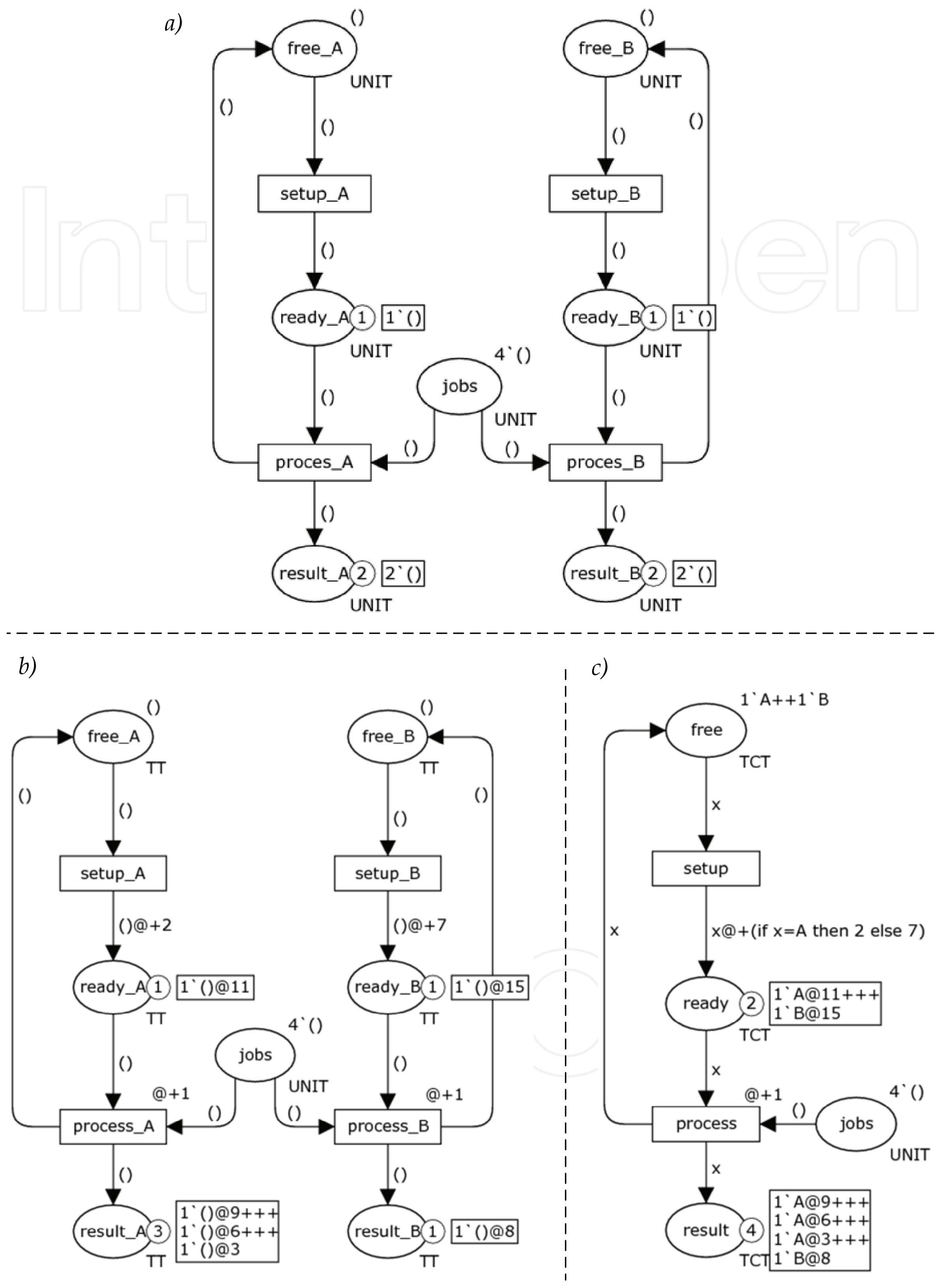
The formal definition of the HTCPN semantics is also quite complex. It is mainly based on concept of timed multisets. This definition will not be quoted here. Instead, a few simple examples of HTCPN structures will be presented that demonstrates basics of the semantics. The first of the structures (fig. 1a) represents a model equivalent to a low-level net. From the formal point of view, it is a coloured net because place markings and arc expressions are typed. However, the type (colour set) of all places and arcs is *UNIT*. It is the simplest type predefined in the *CPN Tools* environment that has only one value, denoted by the empty pair of brackets (). A token of type *UNIT* can be either present or absent and it does not carry any additional information. Thereby, *UNIT* type tokens emulate indistinguishable black tokens and the net structure presented in the figure 1a is semantically equivalent to a basic P/T net. An expression that represents n tokens of type *UNIT* has the form *n*`(), if n = 1, it can be simplified to the notation (). The discussed net (fig. 1a) has 7 places and 4 transitions. The initial marking assigns 1 token to the places *free_A* and *free_B* as well as 4 tokens to the place *jobs*. Every arc carries 1 token at a time. The structure can be interpreted as a model of a simply production system in which 2 machines (*A* and *B*) have to process 4 jobs. Every machine has 2 states (*free* and *ready*). Changes between the states are modelled by moving tokens when transitions *setup* and *process* are executed. These executions represent events in the system. It is obvious that only a one transition from the pair (*setup_A*, *process_A*) and also (*setup_B*, *process_B*) can be enabled (that means ready for execution) at a time. Thus, at the most two transitions can be enabled simultaneously in the whole net. If two or more transitions are enabled, the selection of that one to be executed is nondeterministic. This rule is an important characteristic of Petri nets that makes possible to determine many variants of evolution of a concurrent discrete event system. Different variants can be a result of a lack of full information about a system or it can be an immanent feature of a modelled system. In the result of the execution of the model presented in the figure 1a, it is possible to get different final marking in the places *result_A* and *result_B* with the restriction that the total number of tokens in these places has to be 4. This relates to the fact that times of processes and setups are not known, so it is undetermined how many jobs will be processed by any of machines. The final marking denoted in the figure 1a (rectangles close to places) is only one of the possibilities.

The model in the figure 1b represents a timed Petri net. This net structure is similar to that previously discussed (fig. 1a) but a timed extension has been added. The time representation is based on two elements in the HTCPN formalism. Token *timestamps* are the first element and the *global clock* is the second one. The HTCPN net can have either untimed and timed places. Every token in a timed place has attached a special value called *timestamp*. The condition of transition enabling is most restricted for the timed HTCPN than for untimed. A transition is consider to be enabled if and only if there are sets of tokens in its input places sufficient to cover demand of all input arcs (identically as for an untimed net), but for timed places only the tokens having timestamps not greater than a value of the global clock are taken for this calculation. In other words, a timed token can be taken from a place only if its timestamp is less than or equal to a value of the global clock.

Respecting given enabling definition, the algorithm of a timednet execution can be formulated as follows:

1. Set the global clock to 0.
2. If there are enabled transitions go to 4 else go to 3.
3. Increase the value of the global clock to the smallest possible value for which there are enabled transitions, if it is impossible then STOP.
4. Select randomly a subset of the enabled transitions (more precisely it should be a subset of so-called *binding elements*).
5. Execute selected transitions.
6. Go to 2.

The notation <*untimed marking*>@*TS*, used for defining a marking of timed places, denotes that tokens represented by the expression <*untimed marking*> have the timestamp *TS*. It is thought *TS* = 0 if @*TS* postfix is omitted. The notation <*untimed marking*>@+*d* on an input arc of a timed place denotes that tokens represented by the expression <*untimed marking*> will be inserted to the destination place with the timestamp *TS* = *GC* + *d*, where *GC* is the value of the global clock in the moment when the transition connected with the arc is executed. The inscription @+*d* can be also added to a transition and then the value *d* applies to all output arcs of the transition. The parameter *d* is called *delay*. It can be informally thought as a time of the event modelled by the execution of the transition.



colset TT = UNIT timed; colset TCT = with A | B timed; var x : TCT;

Fig. 1. The example CPN structures: a) simple P/T net, b) timed net, c) timed coloured net.

In the model presented in the figure 1b, the timed type *TT* (*Timed Type*) is assigned to all places, except *jobs*. This type represents black tokens (*UNIT*) with timestamps. The initial marking is the same as in the previous model (fig. 1a), but the tokens in the places *free_A* and *free_B* get implicitly timestamps equal 0. Arc and transition delays define duration of modelled activities: the setup A lasts 2 (units of time), the setup B lasts 7 and the both processes last 1. At the start of the model execution, two enabled transitions *setup_A* and *setup_B* are fired (with the nondeterministic selection). After that, the place *ready_A* has the marking 1`()@2 and *ready_B* has the marking 1`()@7. Next, the global clock is increased to the value 2 and the transition *process_A* becomes enabled and it is executed. After this execution, the place *free_A* has the marking 1`()@3 and the marking of the place *free_B* remains 1`()@7, thus, the global clock is increased to the value 3 and the transition *setup_A* is executed, and so on. The final marking of the net model (fig. 1b) specifies that the machine *A* will process three jobs, that will finish in the moments 3, 6 and 9, while the machine *B* will process a one ajob to the moment 8. An analysis or a detailed simulation of the net model shows that it is only one possible final marking in this example.

Colouring of the HTCPN is syntactically represented by assigning colour sets (types) to places, defining typed variables and assigning expressions to arcs and transitions. It introduces elements of a textual programming language with complex data types to the Petri nets modelling. The syntax definition does not specify the language exactly. In *CPN Tools* the functional programming language called *CPN ML* is used, that is an extension of the *Standard ML* language. There are two main advantages of using colouring in Petri nets:

1. It is possible to build more compact structures, because similar substructures can be folded to a one subnet by replacing structural diversity with the diversity of token values.
2. It is possible to execute procedures of arithmetic computation and data processing that would be complex or even unfeasible with the use only structural net elements.

The example of a coloured HTCPN model has been shown in the figure 1c. The model is fully functionally equivalent to the one previously discussed (fig. 1b). Each pair of places (*free_A*, *free_B*), (*ready_A*, *ready_B*), (*result_A*, *result_B*) has been replaced with a one place, respectively: *free*, *ready* and *result*. The modelled machines are distinguished by token values. Each token has one of the two values *A* or *B* with a timestamp, that is specified by enumerated colour set definition *TCT* (*Timed Coloured Type*). Variables assigned to input arcs of transitions are often used in the coloured net models. These variables bind a value of a token consumed by a transition and this value can be used in output arc expressions. For example, the expression *x*+@(*if x* = *A then* 2 *else* 7) has been used in the considered model to infer a delay value from a machine type.

4. Modelling production systems by hierarchical timed coloured Petri nets

The HTCPN is a powerful formalism for modelling and simulation of production systems. In this heading the HTCPN patterns and substructures will be introduced that reflect some features of production systems (sequence constraints, batch processing, buffer limits, etc.). These patterns and substructures are in majority loose coupled, so can be added to models independently. Some of the concepts presented in the next subheadings have been adopted from a literature and others have been fully introduced by the author. All the concepts form a compact review that presents how to construct a HTCPN model of even complex production system.

4.1 Production system structure

There are used many different models of production systems. The popular models considered in the context of modelling and scheduling problems are *single machine models*, *parallel machine models*, *flow shop*, *job shop* and *open shop* (Pinedo, 2008). It is possible to find applications of Petri nets in problems connected with various models. However, in this chapter, the model called *flexible job shop* (FJS) (Pinedo, 2008) has been taken into consideration. There are two reasons of this choice:

1. FJS is the strongly generalized model of a production system. It covers flow shop, job shop and parallel machine environments. So, the FJS can be used for modelling many of systems, including majority of *flexible manufacturing systems* (Christo & Cardeira, 2007).
2. The presented modelling concepts have been put into practice in a screw factory, where the production system has the FJS structure.

Models of production systems with sequence constraints, also the FJS model, are often represented by the disjunctive graph formalism (Brucker, 2007). A disjunctive graph of a hypothetical FJS structure is presented in the figure 2a. There are 3 machines (*X*, *Y*, *Z*) and 3 jobs (*A*, *B*, *C*) in the system. The job *A* has 2 operations, the first of them is processed on the machine *X* (and it lasts 45), the second one can be processed on the machine *Y* (30) or *Z* (25). Similarly, the job *B* is processed on the machine *X* (30) or *Y* (110) and next on the machine *Z* (40), whereas the job *C* is processed on machines *X* (50), *Y* (45) and *Z* (15) in turn.

It is easy to construct the HTCPN model that represents the same FJS system as a given disjunctive graph. For example, the graph in the figure 2a and the net model in the figure 2b relate to the same FJS system. The rules of the net model construction has been assumed as follows:

1. For each operation there is a place *init_<operation>*.
2. For each job there is a place *end_<job>*.
3. For each machine there is a place *mach_<machine>*.
4. For each pair operation-machine there is a transition *proc_<operationMachine>*. The transition has a time delay inscription @+*d*, where *d* is the processing time for the pair operation-machine.
5. For each transition *proc_<operationMachine>* there are:
   - The arc directed from the place *init_<operation>* to the transition,
   - The arc directed from the transition to the place *init_<nextOperation>*, if there exists a next operation in a job, or else to the place *end_<job>*.
   - The bidirectional arc that connects the transition and the place *mach_<machine>*.
6. All the places have the colour set *JOB*, that is defined as *UNIT timed*.
7. All the places *init_<operation>* that represent the first operations in jobs and also all the places *mach_<machine>* have initial marking (), all other places have empty initial marking.
8. All the arcs have the arc expression ().

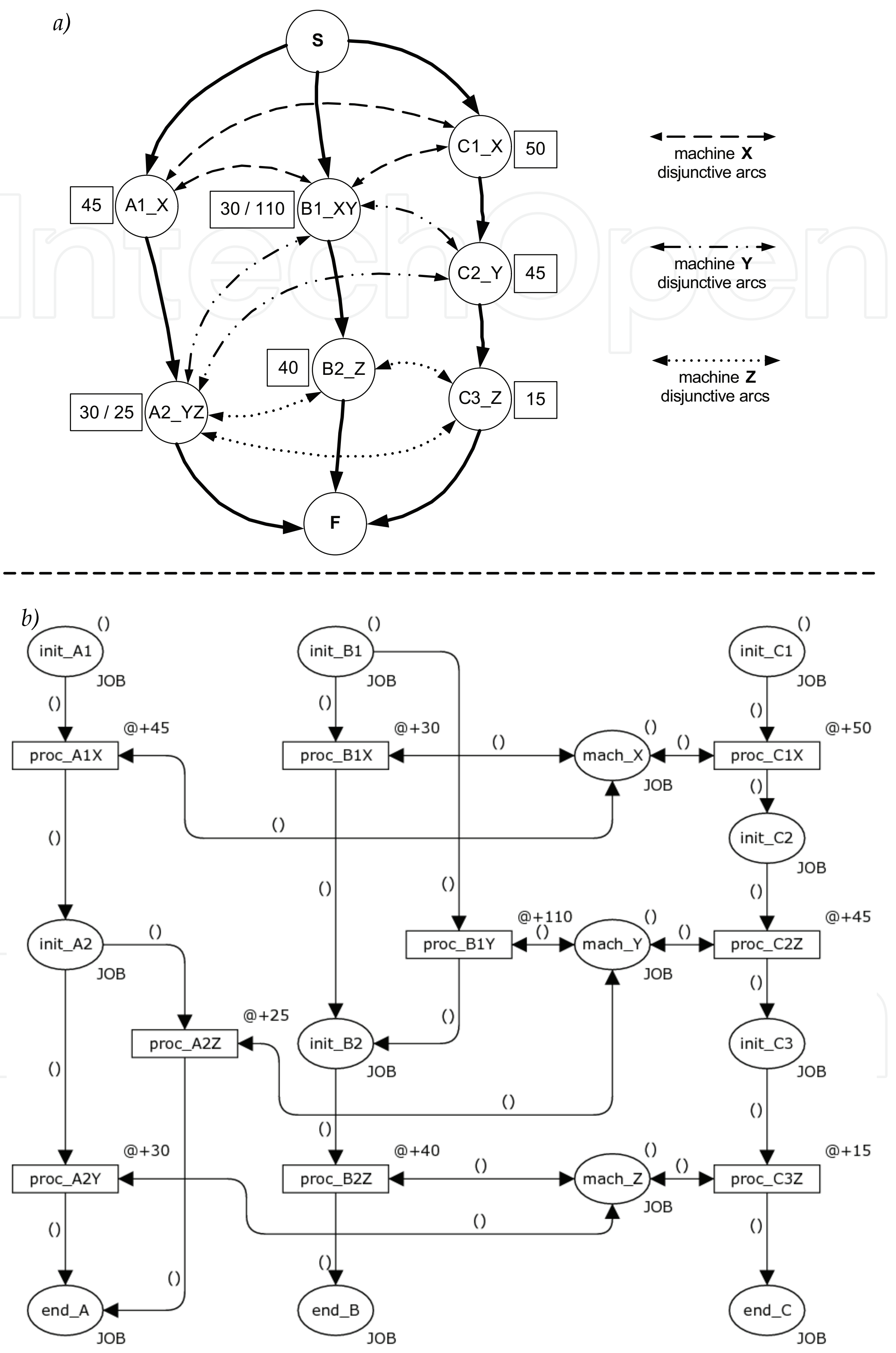
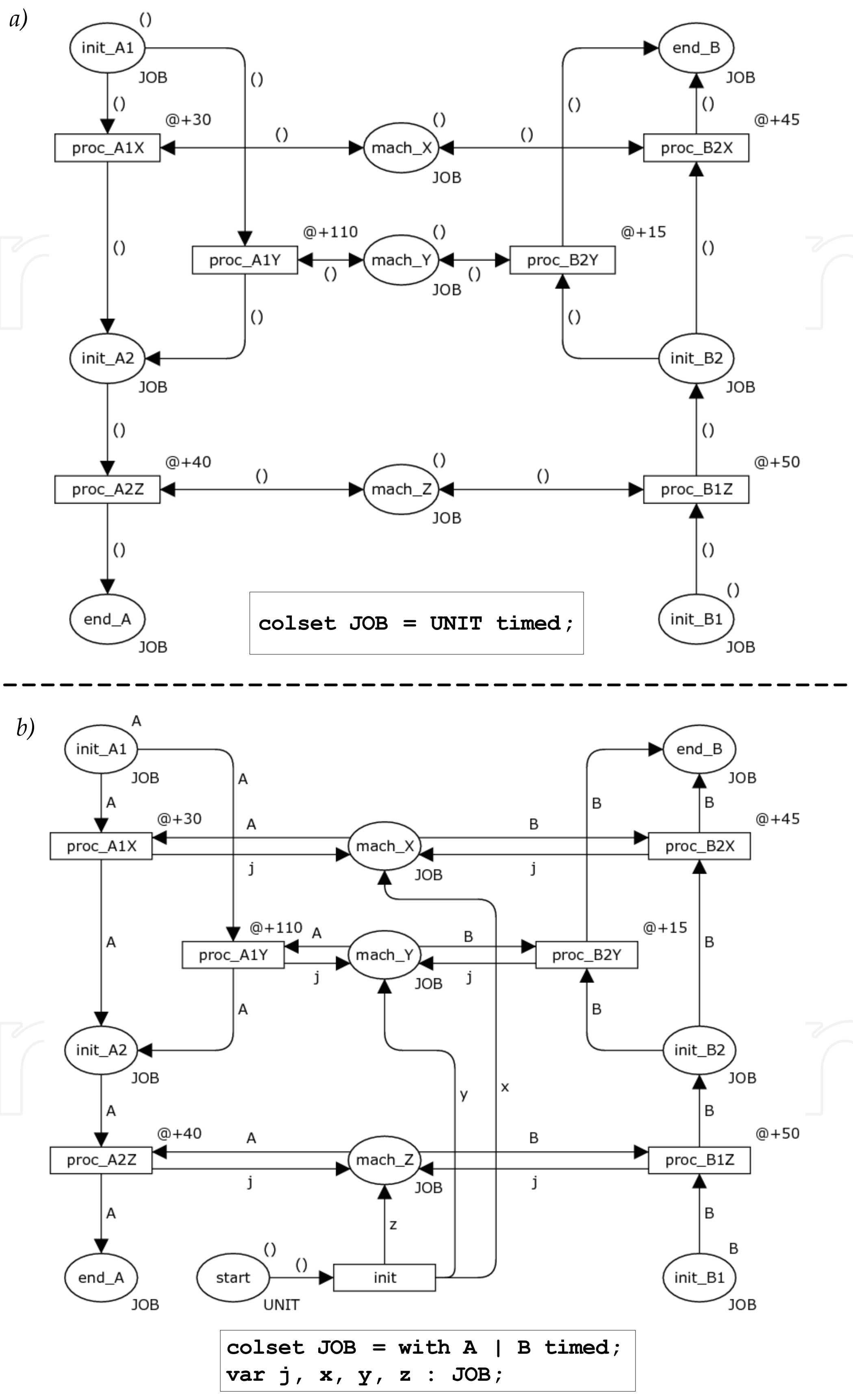


Fig. 2. The representation of the FJS system: a) disjunctive graph, b) timed Petri net model.



```
colset JOB = UNIT timed;
```

```
colset JOB = with A | B timed;
var j, x, y, z : JOB;
```

Fig. 3. Different sequencing rules: a) non-delay sequences, b) all sequences.

For each job tokens flow from the *init* place of the first operation, through consecutive operation places, to the *end* place. The delay added to timestamps as a result of executing of *proc* transitions causes that no operation can start until a previous one finishes and a machine token is blocked while operation processing.

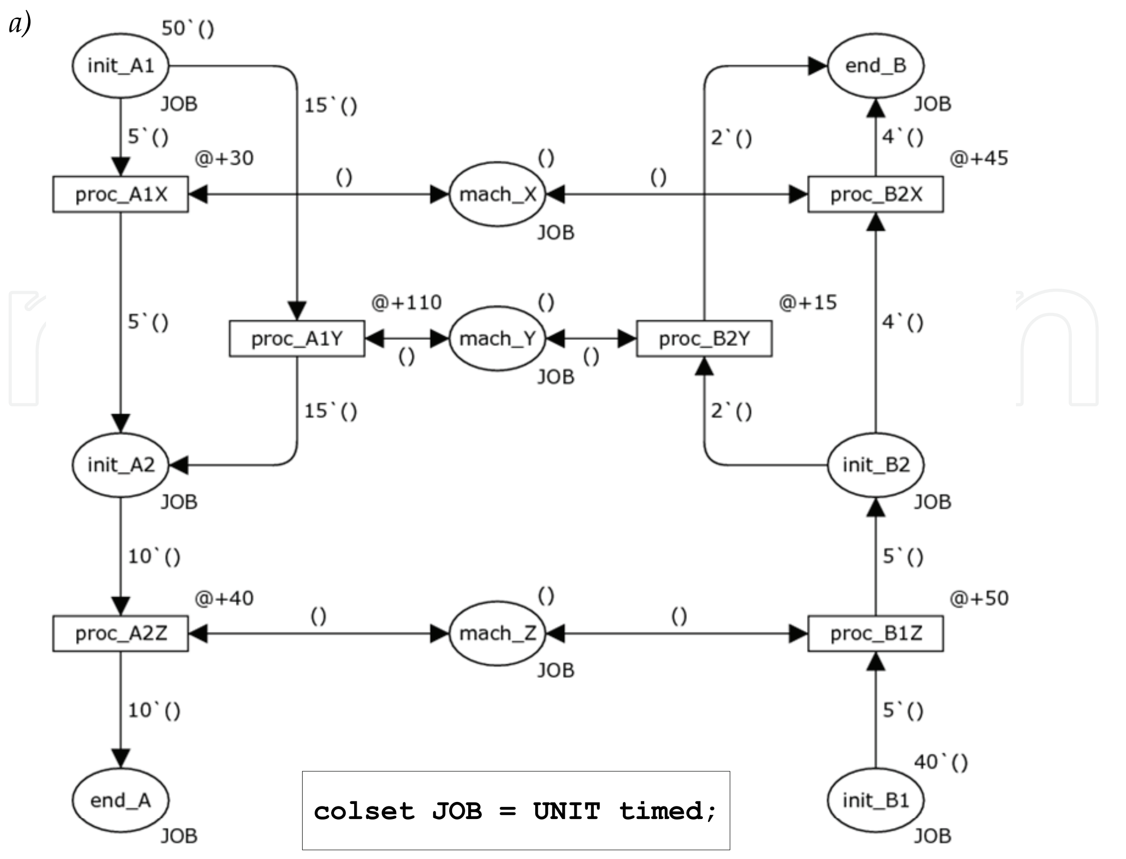
It can be noted that the FJS system representations in the figures 2a and 2b are not completely functionally equivalent. While the disjunctive graph (fig. 2a) represents all possible operation sequences, the net structure (fig. 2b) models only so-called *non-delay* schedules (Pinedo, 2008). It is because the *mach* places are marked by non-coloured tokens and operations to processing cannot be reserved in advance. So, if any operation is waiting for a machine and a machine is free, processing starts immediately. However, it is also not difficult to build the net structure that models all sequences and is fully equivalent to disjunctive graph. An example is shown in the figure 3b. The models in the figures 3a and 3b represent the same simple production system, but the first one has been constructed on the basis of the previously given rules and the second one has been prepared to generate all sequences. The type *JOB* has been redefined (fig. 3b) to have a set of values (in the example *A* and *B*) that represent processed jobs. Tokens coming in to the *mach* places have randomly selected colours (*A* or *B*), because these tokens are generated by so-called *free variables* (Jensen & Kristensen, 2009). So, a sequence of operations for each machine is selected non-deterministically, and thus any execution scenarios can be achieved.

4.2 Batch processing

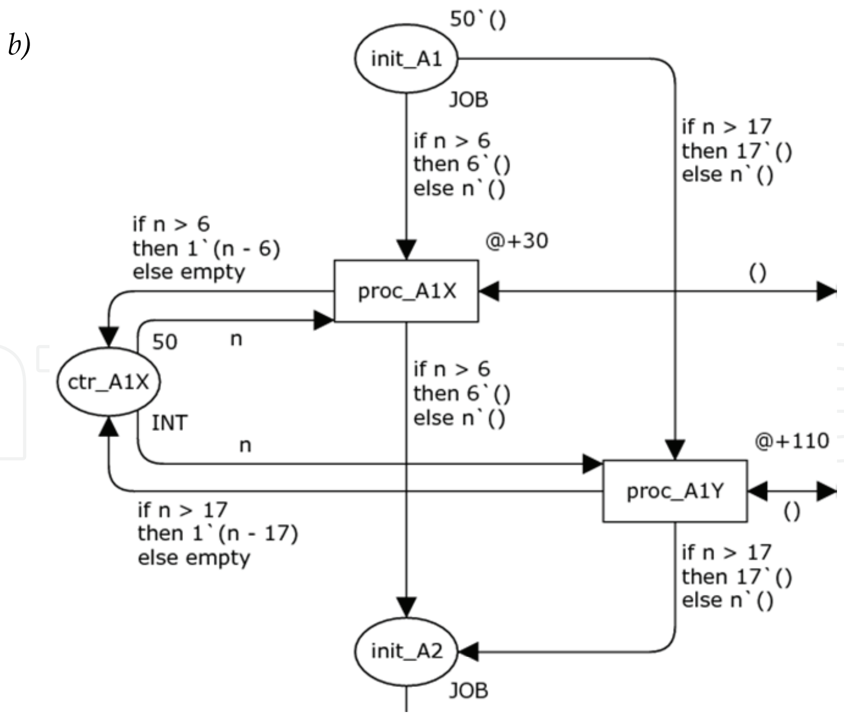
It is not very challenging and useful to create HTCPN models equivalent to disjunctive graphs, because there are large number of effective simulation and scheduling algorithms that use the disjunctive graph representation. The usefulness of modelling by Petri nets relates mainly to flexibility in adding different extensions to a model. Some examples of that extensions are presented in the remainder of this heading.

Operations do not have to consist of a single processing action. An initial marking of the first *init* place of a job can have many tokens instead of one. Moreover, arcs connected to *init* places can also carry many tokens at a time. This extension makes possible to model batch processing. In any operation some amount (batch) of a job is processed. That processing amount can be assign individually for each machine-job pair. An example is presented in the figure 4a. The job *A* has the total amount 50. In the first operation of the job *A* the amount 5 is processed, if it is done on the machine *X*, or the amount 15, if it is done on the machine *Y*. In the second operation of the job *A* the amount 10 is processed, and so on. The concept of modelling of batch processing presented in the figure 4a is adopted from the solution proposed by Zhang, Gu and Song (2008).

It is a possible situation that not all amount of a job will be processed, because a last batch in the *init* place will be smaller than a processing amount of every machine. That is unavoidable for most combinations of real-world parameters. It does not have to be a problem. If an amount is large and it is processed in many batches, a small rest at the end of the processing could be probably neglected. But if it is necessary to process full amount, the solution shown in the figure 4b can be used. The additional place (*ctr_A1*) has been introduced that is connected to all *proc* transitions related to a one operation (here *A1*) and this place counts the amount that remains unprocessed. If this amount is smaller than a standard batch size for a machine the smaller last batch is processed.



```
colset JOB = UNIT timed;
```



```
colset JOB = UNIT timed;
var n : INT;
```

Fig. 4. Modelling batch production: a) Petri net model, b) processing of smaller last batch.

4.3 Extended machine model

The simple machine representation proposed in the prior subheadings can be insufficient for modelling of real production. Additional operation rules and parameters have to be allowed for. For example, an extended machine model is presented that combines following features:

1. Setup times between operations are taken into consideration.
2. An availability calendar is defined for a machine.
3. A machine processes operations batch by batch. Batches are processed without interrupting, so a batch processing starts only if it can be finished in the same bracket of availability.

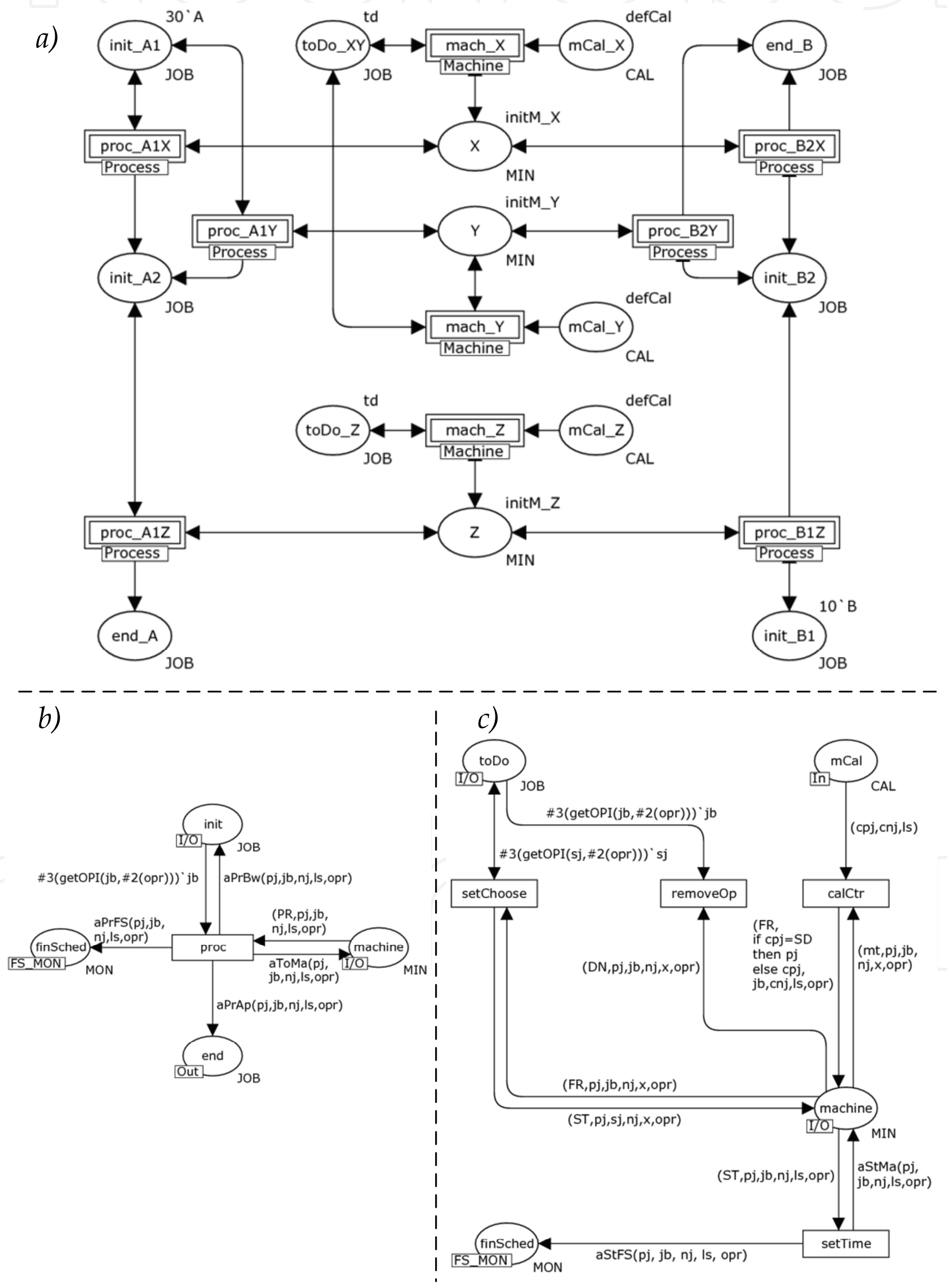
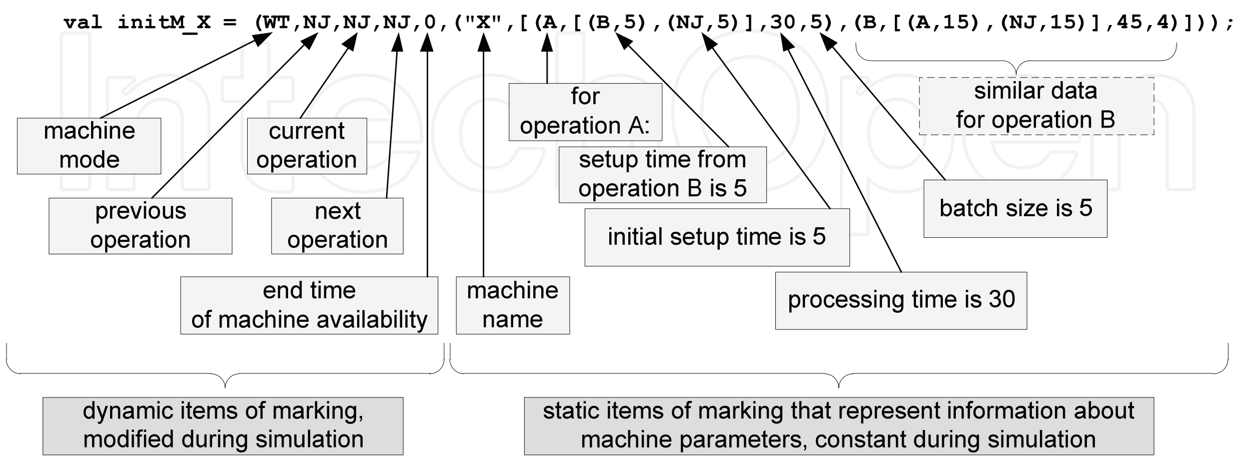


Fig. 5. The HTCPN structure with an extended machine model: a) the top-level structure, b) the inside view of *Process* module, c) the inside view of *Machine* module.

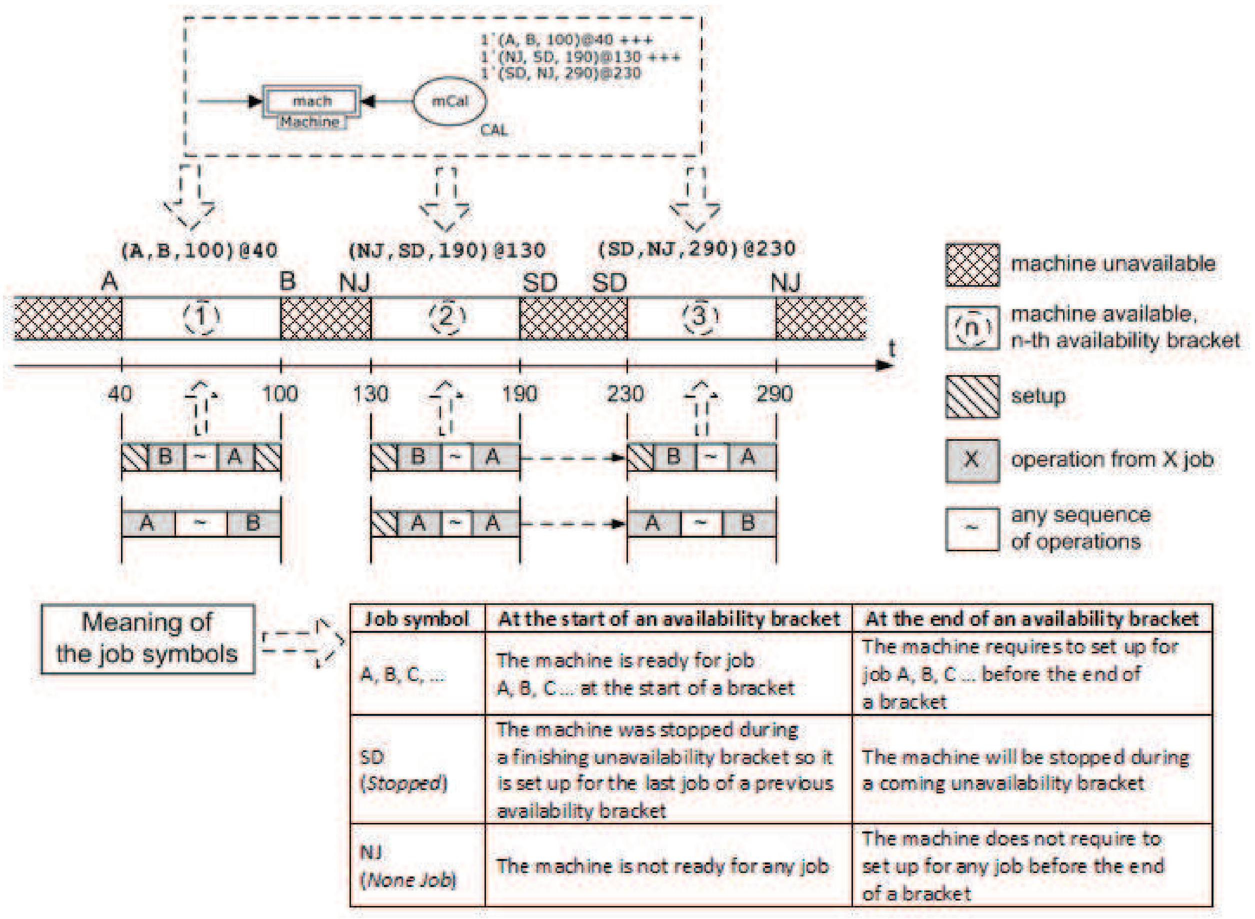
The HTCPN model of a simple production system with the extended machine model is shown in the figure 5. The hierarchical two-level net structure has been used. Details of substructures that represent batch processing logic and machine model logic has been moved to the modules *Process* (fig. 5b) and *Machine* (fig. 5c). Thus, the top-level structure (fig. 5a) remains quite simple and legible.



```
val initM_X = (WT,NJ,NJ,NJ,0,("X",[(A,[(B,5),(NJ,5)],30,5),(B,[(A,15),(NJ,15)],45,4)]));
```

machine mode
previous operation
current operation
next operation
end time of machine availability
machine name
for operation A:
setup time from operation B is 5
initial setup time is 5
processing time is 30
batch size is 5
similar data for operation B
dynamic items of marking, modified during simulation
static items of marking that represent information about machine parameters, constant during simulation

Fig. 6. The example definition of a token that represent machine state and parameters.



Job symbol	At the start of an availability bracket	At the end of an availability bracket
A, B, C, ...	The machine is ready for job A, B, C ... at the start of a bracket	The machine requires to set up for job A, B, C ... before the end of a bracket
SD (*Stopped*)	The machine was stopped during a finishing unavailability bracket so it is set up for the last job of a previous availability bracket	The machine will be stopped during a coming unavailability bracket
NJ (*None Job*)	The machine is not ready for any job	The machine does not require to set up for any job before the end of a bracket

Fig. 7. The representation of a machine availability calendar by a substructure of a HTCPN model.

Each place that represents a machine state (*X*, *Y*, *Z* in fig. 5a) is connected to the *Machine* substitution transition. These transitions include internal logic of machines. Additional places *toDo* control amount of batches that remains to be processed. Places *mCal* (a one place per a machine) represent calendars of machines availability. The structures of the modules *Process* and *Machine* are quite complex and these will not be detailed discussed here.

The marking of the places *X*, *Y*, *Z* consists of two sets of items (fig. 6). The first set groups items that represent dynamic elements of a machine logic. These items specify a present state of a machine, executed operations (previous, current and next) and end time of a machine availability bracket. This information is needed for control a simulation of the machine work. The second set groups items that represent parameters of machines that remain constant during the simulation. These parameters specify setup times for all possible previous operations, a processing time (per batch) and a batch size for each operation that the machine can execute.

The illustration in the figure 7 explains the HTCPN implementation of a machine availability calendar. The initial marking of the *mCal* place defines the set of availability brackets. The brackets are defined not only by start and end moments, but also by start and end job symbols. These symbols are needed to determine setup operations at the starts and at the ends of the brackets. The presented calendar substructure can be easily modified or extended to represent varying efficiency of a machine, periodical changes of availability, etc.

4.4 Transport subsystems

It is often necessary to model also transport processes in a production system. Transport is usually an activity realized between two consecutive operations. So, this activity can be modelled by additional net elements inserted between transitions that represent operations before and after transport (fig. 8a). In the simplest case, it can be a one transition with time delay (fig. 8a, *trans_A12*). This transition adds a transport delay to carried tokens. It should be noted that the transition is fired immediately when a sufficient number of tokens waits in an input place. Thus, intervals of transport time of many batches can overlap (fig. 8a). This is the behaviour that characterize for example a conveyor belt. Of course, the size of process batch and the size of related transport batch don not have to be equal.

If dedicated transport is used, for example a forklift truck, it can be modelled as shown in the figure 8b. The transition *trans_A12* carries tokens from the place *fin_A1* to the place *init_A2* and simultaneously it adds delay to the timestamp of the token in the place *TRANS*. Because of that, no following transport can start until the delay time expires. If this delay time is greater than transport time, the difference represents return time of the vehicle. For example, the transport modelled in the figure 8b lasts 50 and return time is 30.

In the figure 8c, a hypothetical situation is presented where three operations share a pair of transport vehicles (two tokens in the place *TRANS*). Transport between the operations *A1* and *A2*, as well as *B2* and *B3* can be realized by any one of the vehicles. However, the transport between the operations *A2* and *A3* requires two vehicles at the same time.

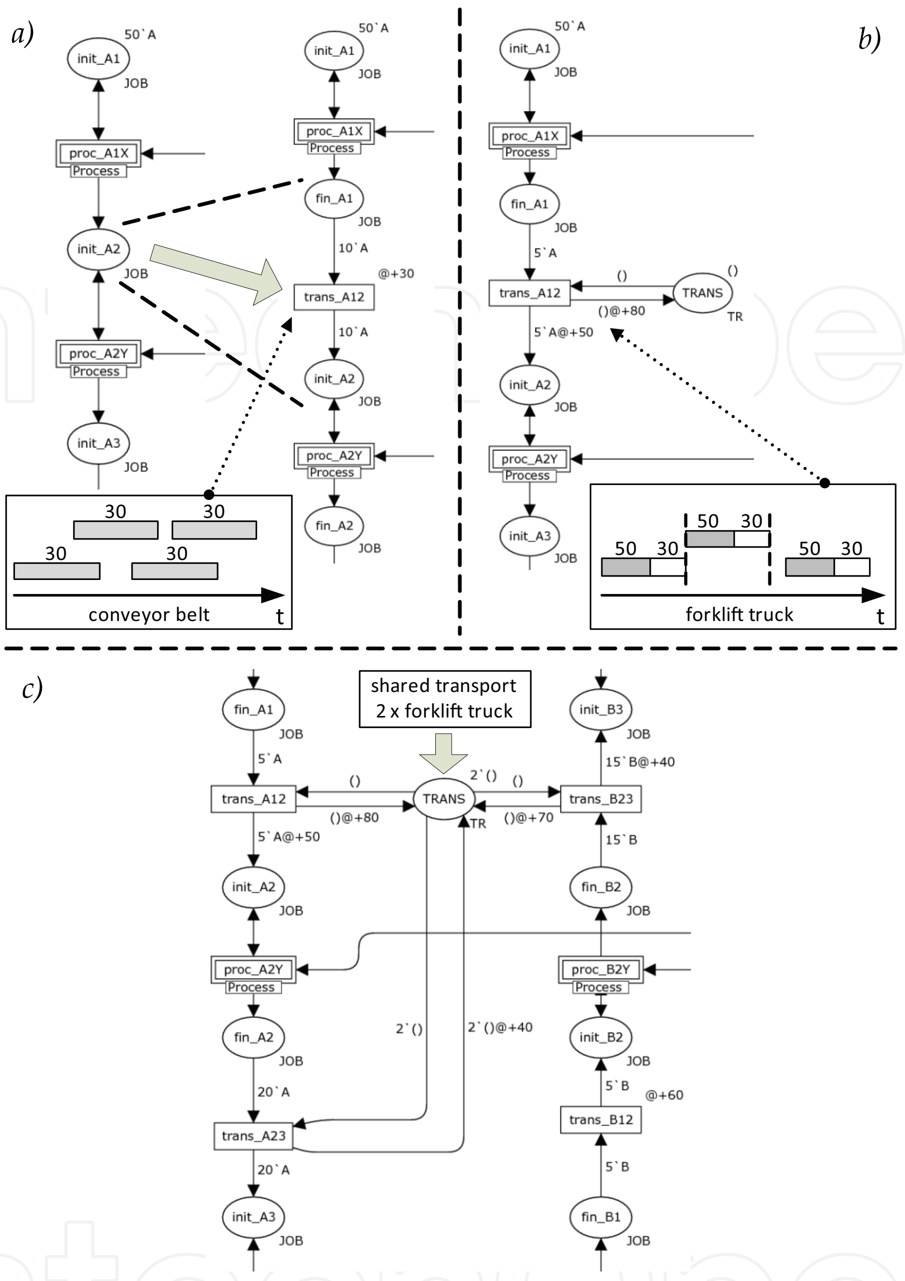


Fig. 8. Modelling of transport: a) transport delay, b) dedicated transport, c) shared transport.

4.5 Limiting structures

It is quite simple to model basic limits characteristic of production systems by Petri nets. The limit of release time (not-earlier-than restriction) is often used. It can be modelled as shown in the figure 9a. The additional place *rel_A2Z* with the initial marking ()@140 causes that the operation represented by the transition *proc_A2Z* cannot be executed ahead of time 140.

If a place models a buffer of material waiting for processing, its capacity sometimes needs to be limited. It can be done using so-called *anti-place*, that is a place connected with the same transitions as an original place but with arcs directed inversely. The total number of tokens in a place and its anti-place remains constant, so the initial marking of anti-place determines maximal capacity of the related place, provided that this place is initially empty. For example (fig. 9b), the place *init_A2* has the capacity limited to 20, because of the anti-place *buf_A2Z*.

An anti-place does not have to relate to a one place in a net structure, it can encompass a greater subnet. An example is presented in the figure 9c. The anti-place *ctr* causes that the total number of tokens in the route between the transitions *proc_B1Z1*/*proc_B1Z2* and *proc_B2Y*, so in the places *fin_B1* and *init_B2* jointly, cannot be greater than 15.

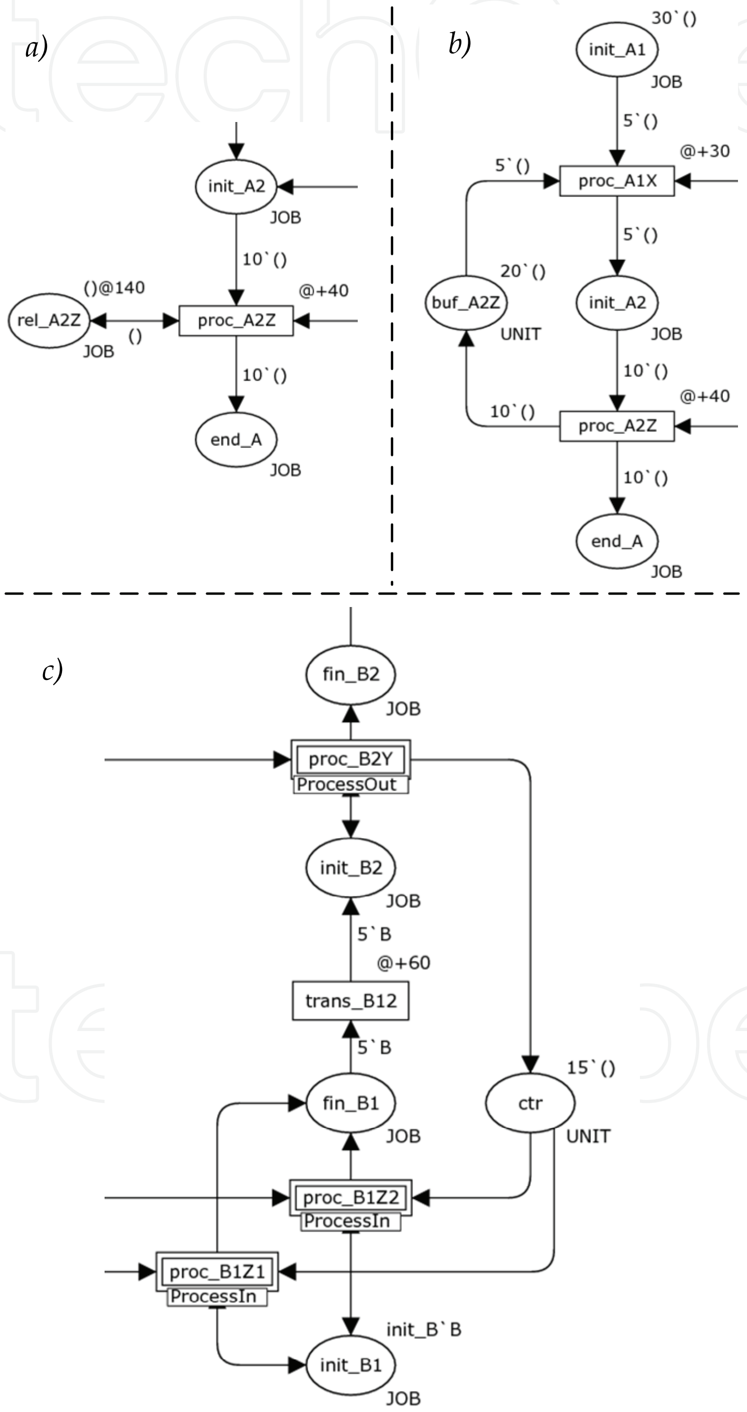


Fig. 9. Limiting structures: a) not-earlier-than restriction, b) place capacity limit, c) subnet capacity limit.

5. Design of scheduling module for screw factory

The introduced rules of production system modelling by the HTCPN have been applied in practice. A prototype scheduling module has been created for a medium size screw factory. This is the part of a larger project that strives for implementing an innovative *flexible manufacturing system* (Christo & Cardeira, 2007). Hardware and software infrastructure for the project has been created and the real-time monitoring subsystem has been developed so far (Żabiński & Mączka, 2010). It has given a chance to have very up-to-date information for a scheduling module.

The general concept of the solution is that all needed information about the state and the structure of the production system is acquired from the main computer system of the factory. This information is used for generating the HTCPN structure that reflects present state of the production. The generated net structure is simulated to foresee next events in the system. The simulation is driven by a priority based controller that makes choices if a decision is needed which of machines or jobs to select for processing. The set of events registered during the simulation is transformed to a production schedule and it is presented in the form of a Gantt chart. All the procedure of data acquiring, net generation and simulation as well as results presentation is executed automatically on demand of a planist.

5.1 Production profile

The production structure in the factory has a form of a flexible job shop system. Beside the standard relations and constraints characteristic of FJS, there are specific features of the system:

1. Batch processing is applied. Batch sizes are determined by sizes of containers for screws.
2. For each machine an individual calendar of availability is defined.
3. For each operation a not-earlier-than restriction is given.
4. For each triple (machine, previous job, next job) a setup time is defined.
5. There are virtually only non-delay production sequences executed.
6. For some of jobs due dates are specified.

There are about hundred machines taken into account in scheduling. A typical schedule includes a few hundred jobs and each job consists of two to ten operations.

5.2 Software structure

The general software structure of the developed flexible manufacturing system is presented in the figure 10. The *main system logic server* unit represents a fundamental business logic of the system that currently realizes tasks of real-time monitoring, data acquisition and generating of statistics. The part that performs the scheduling algorithm consists of the *CPN scheduler and "what-if" generator server/client application* blocks.

The scheduling module acquires data from the main system logic server. This data can be divided into two parts:

1. A set of information that defines the current production state registered by the real-time monitoring subsystem. This information specifies real states of machines (working, stopped, setup, etc.) and jobs (processing, stopped, cancelled, etc.).

2. A set of information that has been defined regardless of the production state. It represents jobs that have to be processed in the future and all their parameters (an amount, a batch size, a technological route, processing times, etc.).

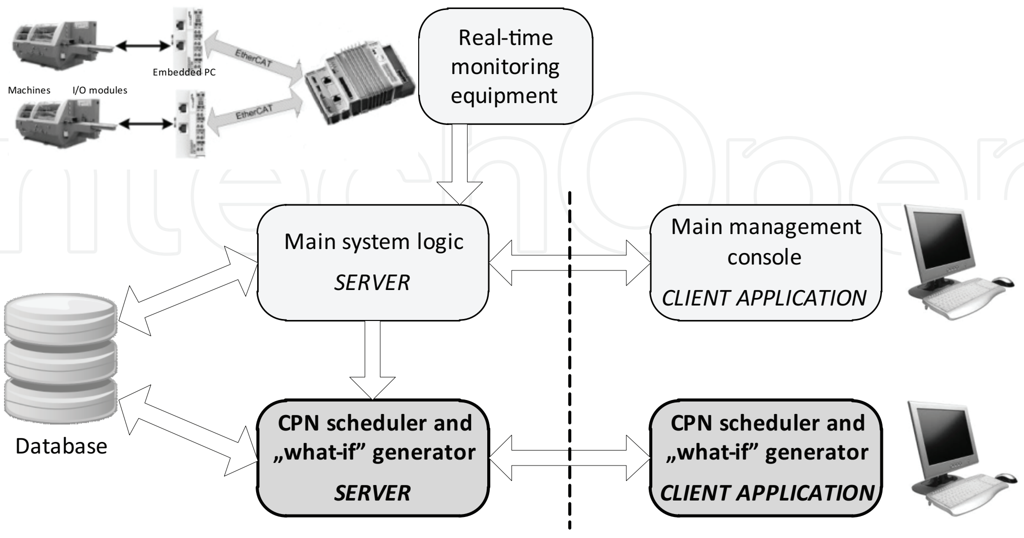


Fig. 10. The general structure of production monitoring/planning/scheduling system.

Both the sets of information are needed for generating up-to-date schedules. The scheduling module has also an access to a database where it stores its own persistent data.

In the figure 11, the inside view of the *CPN scheduler and "what-if" generator sever* module has been shown. The block A (*generator of HTCPN structure*) generates HTCPN structure using data taken from the *main system logic server*. The generated net structure has a form of a set of connected objects. The objects are coded in Java programming language. Any object is either a place or a transition. Arc expressions are included in the transition objects. Net construction rules are similar to that discussed in the subheading 4.3.

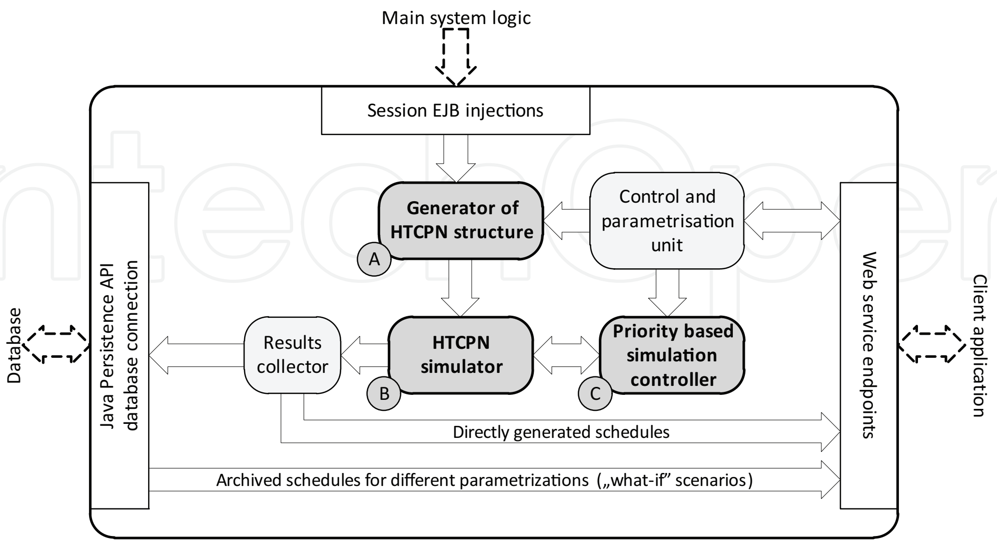

Main system logic
Session EJB injections
Generator of HTCPN structure
A
Control and parametrisation unit
Java Persistence API database connection
Web service endpoints
Database
Client application
Results collector
HTCPN simulator
B
Priority based simulation controller
C
Directly generated schedules
Archived schedules for different parametrizations („what-if" scenarios)

Fig. 11. The inside view of the CPN scheduler and "what-if" generator sever module.

The block B (*HTCPN simulator*) simulates the net structure generated by the block A. If a decision is needed during a simulation, the block C (*priority based simulation controller*) is invoked. The *control and parametrisation unit* block uses data received from the client panel to control details of the blocks A and C execution, for example, due dates of jobs can be changed. Events generated during a simulation are registered by the *result collector* block. The set of results is written to the database and it is sent to the client panel where the results are presented on a Gantt chart. It is also possible to take from the database schedules previously generated for different data sets and parametrisations. These schedules can be compared to discover how changes of parameters affect results ("what-if" scenarios).

5.3 HTCPN simulator

The *HTCPN simulator* block (fig. 11, block B) has been developed specially for the presented scheduling module. It is possible to use the HTCPN simulation engine that is part of *CPN Tools* but this possibility has not been chosen. It has been caused by the several reasons:

1. A net structure has to be generated dynamically for a current state of production. The *CPN Tool* engine supports simulation of a previously edited net diagram.
2. The dedicated simulation block is faster, because it does not have to implement all elements of HTCPN formalism. The elements of the formalism that are not used in the models can be omitted.
3. A special connection between the simulator and the priority based simulation controller is used in the dedicated implementation.

The block diagram of the Petri net simulation algorithm is shown in the figure 12. At the start of any step of the simulation (fig. 12, section A) an enable time of all transitions is verified and there are created two lists. The list *enabledTrans* includes transitions with the enabled time not greater than the value of the global clock and the list *minTimeTrans* includes transitions with the smallest value of the enable time. The smallest value of the enable time is kept in the variable *minTime*. If the value of *minTime* is not valid (*minTime* = *Long.MAX_VALUE*), there are no enabled transitions and the simulation stops (fig. 12, element B). If the value of *minTime* is greater than the value of the global clock, the global clock has to be increased (fig. 12, section C).

A special solution has been used to connect the *HTCPN simulator* with the *priority based simulation controller*. Transitions have been divided into two sets: the set of *decision* transitions and the set of *non-decision* transitions. Only *decision* transitions can be in a conflict in a one simulation step, that is, if one of these transitions will be executed, some others can lost enabling. That conflict represents a scheduling decision. Remaining transitions are consider as *non-decision*. For example, only *proc* transitions in the diagram from the figure 5 have the *decision* type, other transitions are *non-decision* (*setChoose*, *calCtr*, *removeOp*, *setTime*). If any *non-decision* transition is enabled in a given simulation step, it is executed immediately (fig. 12, element D). If a step has only *decision* transitions enabled, the *priority based simulation controller* (called also *optimizer*) selects one of them to the execution (fig. 12, section E).

It should be noted that implemented rules of the simulation and the transition selection do not violate the HTCPN formalism. It is permissible during the simulation to inspect enabled transitions and to select them in a deterministic way. It is also possible in *CPN Tools*.

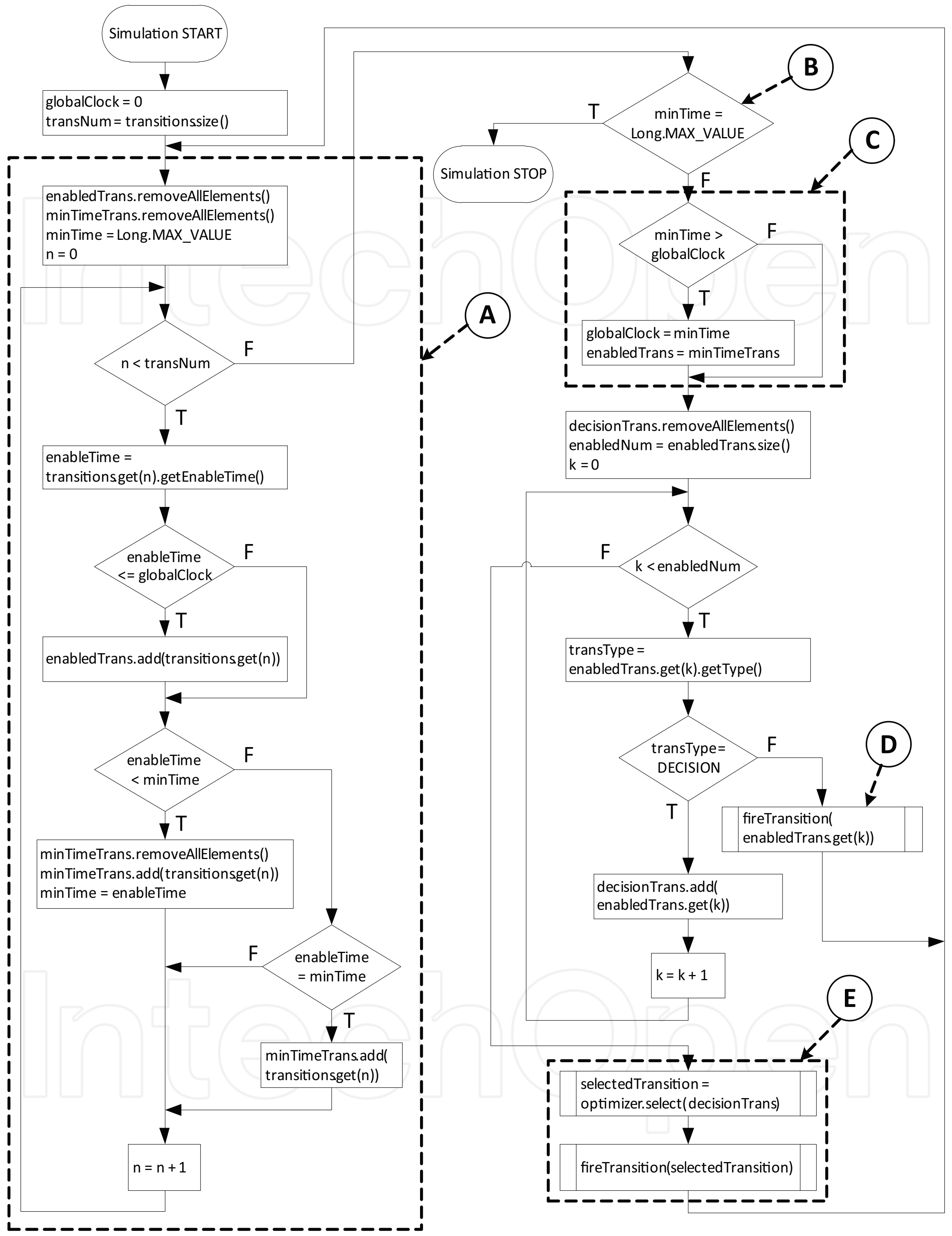

Simulation START
globalClock = 0
transNum = transitionssize()
enabledTrans.removeAllElements()
minTimeTrans.removeAllElements()
minTime = Long.MAX_VALUE
n = 0
n < transNum
enableTime = transitions.get(n).getEnableTime()
enableTime <= globalClock
enabledTrans.add(transitions.get(n))
enableTime < minTime
minTimeTrans.removeAllElements()
minTimeTrans.add(transitionsget(n))
minTime = enableTime
enableTime = minTime
minTimeTrans.add(transitionsget(n))
n = n + 1
minTime = Long.MAX_VALUE
Simulation STOP
minTime > globalClock
globalClock = minTime
enabledTrans = minTimeTrans
decisionTrans.removeAllElements()
enabledNum = enabledTrans.size()
k = 0
k < enabledNum
transType = enabledTrans.get(k).getType()
transType= DECISION
fireTransition(enabledTrans.get(k))
decisionTrans.add(enabledTrans.get(k))
k = k + 1
selectedTransition = optimizer.select(decisionTrans)
fireTransition(selectedTransition)
A
B
C
D
E
T
F

Fig. 12. The block diagram of the Petri net simulation algorithm.

5.4 Priority rules of scheduling

If a conflict between jobs or machines is present, a choice is made on the basis of priority rules. Two main parameters are taken into consideration when priorities are determined:

1. Any of jobs can have assigned a hard (constant) priority. Hard priorities are represented by natural numbers that have to be assigned in sequence.
2. Any of jobs can have assigned a due date.

The rules that control the job selection are hierarchically ordered:

1. The hard priorities are the most important.
2. If there are no operations with the hard priorities (at a given simulation step), the value of the critical ratio is taken under consideration, but the critical ratio value has to be below the threshold fixed as the algorithm parameter. The jobs that have no due date assigned are omitted in this selection step.
3. If the critical ratio of all waiting operations is above the fixed threshold, the operation from a job that has been started earliest is scheduled first. It is done to balance execution time of started jobs.
4. If there are no operations from started jobs, i.e. each waiting operation is the first operation of a job, release dates of jobs are considered in the FIFO order.

The hard priorities should not be overuse because it disrupts functionality of dynamic scheduling rules.

It is rare but possible in the considered production system that many machines wait for a selected job. In that case, the machine having a minimal processing time is chosen.

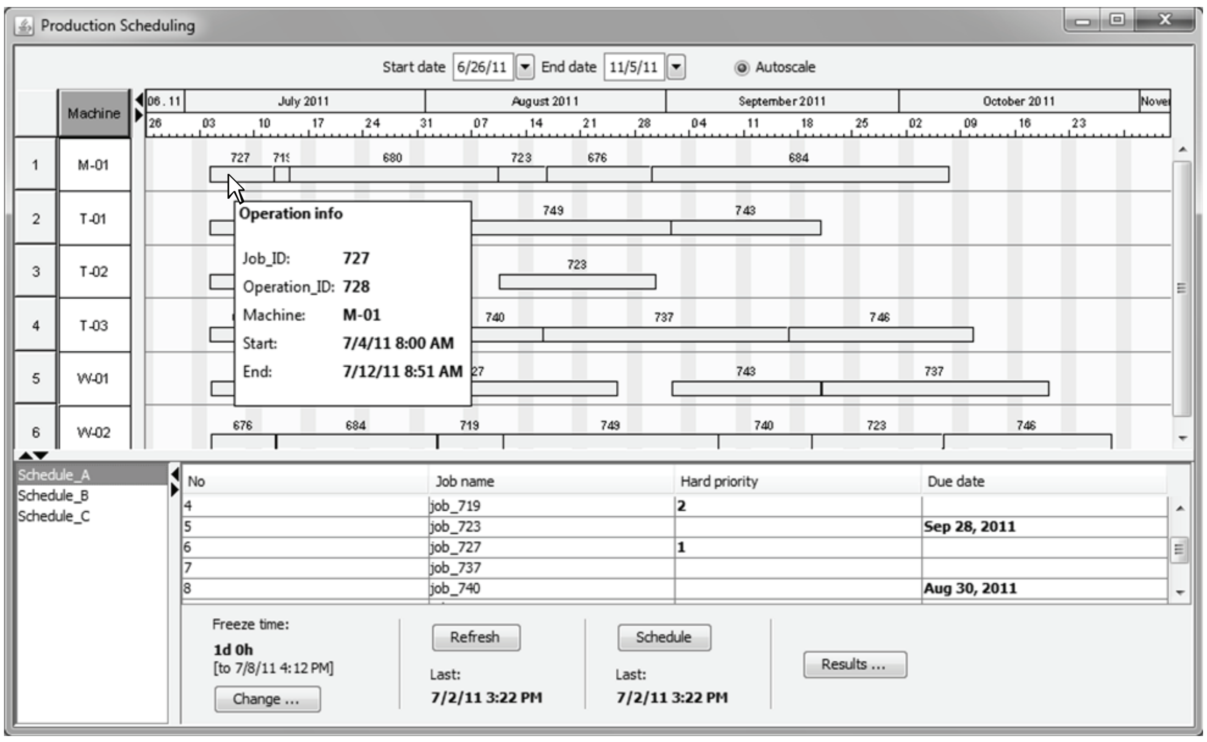
6. Scheduling module in practice

The main window of the scheduling client application is presented in the figure 13. In the bottom left corner of the window there is a list of generated schedules. A planist can remove and create new schedules freely. If any of the schedules from the list is selected, a table with schedule parameters appears in the bottom panel of the window and a Gantt chart appears in the upper panel. Only the operations that are planned to start later than the moment defined by *freeze time* parameter can be reordered in a scheduling process.

It is possible to only refresh a schedule (*Refresh* button) or to initiate the full scheduling process (*Schedule* button). When the schedule is refreshing, sequence relations between operations are not modified, only newest information about the production progress is taken from the real-time monitoring subsystem to update start and end dates of operations. In the refreshing mode, the *priority based simulation controller* (fig. 11) reconstructs operation order saved in a database. In the full scheduling process, operations are ordered on the basis of dynamically determining priorities that have been presented in the subheading 5.4.

A planist can manually modify the generated schedule by moving some operations using the interactive Gantt chart in drag-and-drop fashion. The HTCPN simulator controls modifications and refuses the changes that violate sequence constraints. The optional values of hard priorities and due dates (fig. 13) that influence final priorities can be set by a planist and can be different for each variant of a schedule.

Various schedules can differ in parameters of jobs (hard priorities, due dates) and also in the changes manually done by a planist. It is possible to compare different schedules and to choose the most favourable. The window *scheduling results* (fig. 14) is used for results comparison.



Operation info

Job_ID:	**727**
Operation_ID:	**728**
Machine:	**M-01**
Start:	**7/4/11 8:00 AM**
End:	**7/12/11 8:51 AM**

Schedule_A
Schedule_B
Schedule_C

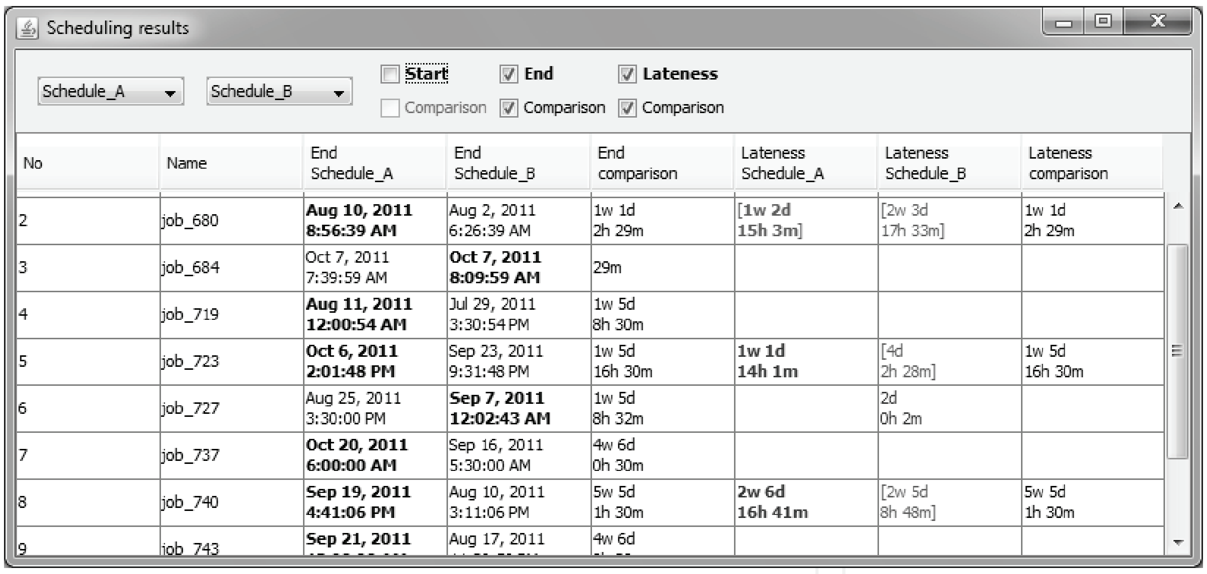
No	Job name	Hard priority	Due date
4	job_719	**2**	
5	job_723		**Sep 28, 2011**
6	job_727	**1**	
7	job_737		
8	job_740		**Aug 30, 2011**

Freeze time:
1d 0h
[to 7/8/11 4:12 PM]

Last:
7/2/11 3:22 PM

Last:
7/2/11 3:22 PM

Fig. 13. The main window of the scheduling client application.



Schedule_A   Schedule_B

[ ] **Start** [✓] **End** [✓] **Lateness**
[ ] Comparison [✓] Comparison [✓] Comparison

No	Name	End Schedule_A	End Schedule_B	End comparison	Lateness Schedule_A	Lateness Schedule_B	Lateness comparison
2	job_680	**Aug 10, 2011 8:56:39 AM**	Aug 2, 2011 6:26:39 AM	1w 1d 2h 29m	**[1w 2d 15h 3m]**	[2w 3d 17h 33m]	1w 1d 2h 29m
3	job_684	Oct 7, 2011 7:39:59 AM	**Oct 7, 2011 8:09:59 AM**	29m			
4	job_719	**Aug 11, 2011 12:00:54 AM**	Jul 29, 2011 3:30:54 PM	1w 5d 8h 30m			
5	job_723	**Oct 6, 2011 2:01:48 PM**	Sep 23, 2011 9:31:48 PM	1w 5d 16h 30m	**1w 1d 14h 1m**	[4d 2h 28m]	1w 5d 16h 30m
6	job_727	Aug 25, 2011 3:30:00 PM	**Sep 7, 2011 12:02:43 AM**	1w 5d 8h 32m		2d 0h 2m	
7	job_737	**Oct 20, 2011 6:00:00 AM**	Sep 16, 2011 5:30:00 AM	4w 6d 0h 30m			
8	job_740	**Sep 19, 2011 4:41:06 PM**	Aug 10, 2011 3:11:06 PM	5w 5d 1h 30m	**2w 6d 16h 41m**	[2w 5d 8h 48m]	5w 5d 1h 30m
9	job_743	**Sep 21, 2011**	Aug 17, 2011	4w 6d			

Fig. 14. The result window of the scheduling client application.

Any pair of the generated schedules can be selected and compared. A lateness value is computed for the jobs that have defined a due date. If a job is not late, a negative value of a lateness is presented in square brackets. For example (fig. 14), *job_723* is late over one week in the *Schedule_A*, but it is scheduled about four days in advance in the *Schedule_B*.

7. Conclusion

In the chapter, the hierarchical timed coloured Petri net (HTCPN) formalism has been presented as a powerful and flexible tool for modelling and simulation of complex production systems. It has been proposed how to represent different elements and behaviours of production systems with the use of the HTCPN formalism and how to build complete models of production systems. The main contribution of the work is the presentation of the set of modelling rules together with the evidence that these are useful in practice. In contrast to the majority of other works, basic concepts of differing jobs by token colours and representation of processing time in a timed net have been moved to a background. Instead, high level structures have been taken into consideration that represent batch processing, machine calendars, transport, etc.

It is possible to point two main kinds of application of the HTCPN formalism in production modelling, simulation and scheduling:

1. Models can be built and simulated only in *CPN Tools* or a similar environment. The design process of a net model is fast and convenient in the such case. This approach can be used for finding and verification of production system properties.
2. The HTCPN formalism can be used for creating a new algorithm of simulation and scheduling.

There are, of course, advantages and disadvantages of using of the HTCPN formalism. The HTCPN formalism combines structural modelling and textual programming. So, typical for discrete event systems mechanisms as synchronisation, resources sharing or mutual exclusion can be represented structurally, while others as complex numerical computation can be coded in the textual programming language. The great advantage is also that the time representation is included in the formalism. On the other hand, complex HTCPN models are difficult to the formal analysis and the state space inspection. The next disadvantage is that the HTCPN net model can have relatively high computational complexity of simulation, especially if no optimisation is used.

The future work related to the scheduling system in the screw factory includes:

1. Design of a more optimised block of the HTCPN simulator, because not all possibilities of optimisation have been implemented in the present solution.
2. Verification and adjusting priority rules used for scheduling.
3. Verification whether it is possible to combine developed rules of HTCPN modelling with process mining procedures (van der Aalst, 2010) to build net models on the basis of system logs.

In the future work, it is also planned to use advanced optimisation procedures, as meta-heuristic algorithms, for control a net model simulation, instead of simply priority rules.

8. References

Aized, T. (2010). Petri Net as a Manufacturing System Scheduling Tool, In: *Advances in Petri Net Theory and Applications*, Aized, pp. 43-58, InTech, ISBN 978-953-307-108-4, Croatia

Bożek, A. & Żabiński, T. (2010). Colored timed Petri Nets as tool of off-line simulating for intelligent manufacturing systems. *Przegląd Elektrotechniczny (Electrical Review),* Vol.2010, No.9, (September 2010), pp. 101-105, ISSN 0033-2097

Brucker, P. (2007). *Scheduling Algorithms*, Springer-Verlag, ISBN 978-3-540-69515-8, Berlin Heidelberg

Camurri, A.; Franchi, P. & Gandolfo F. (1991). A timed colored Petri nets approach to process scheduling, *Proceedings of CompEuro '91, Advanced Computer Technology*, ISBN 0-8186-2141-9, Bologna Italy, May 1991

Christo, C. & Cardeira, C. (2007). Trends in Intelligent Manufacturing Systems, *Proceedings of* IEEE *International Symposium on Industrial Electronics,* ISBN 978-1-4244-0754-5, Vigo, June 2007

CPN Tools Homepage. (n.d.)., 30.06.2011, Available from http://cpntools.org

Jensen, K. & Kristensen, M. L. (2009). *Coloured Petri Nets: Modelling and Validation of Concurrent Systems,* Springer-Verlag, ISBN 978-3-642-00283-0, Berlin Heidelberg

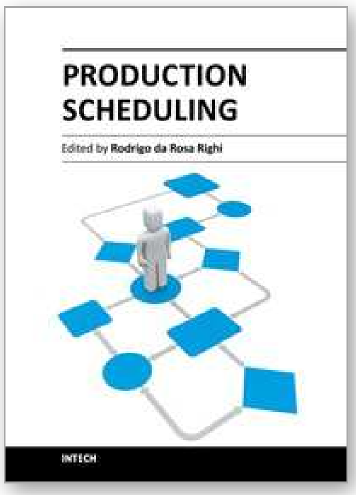
Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems,* Springer, ISBN 978-0-387-78934-7, New York

Tuncel, G. & Bayhan, G. M. (2007). Applications of Petri nets in production scheduling: a review. *International Journal of Advanced Manufacturing Technology,* Vol.34, No.7-8, (October 2007), pp. 762-773, ISSN 0268-3768

Van der Aalst, W. M. P. (2010). Process Discovery: Capturing the Invisible. *Computational Intelligence Magazine, IEEE*, Vol.5, No.1, (February 2010), pp. 28-41, ISSN 1556-603X

Zhang, H.; Gu, M. & Song, X. (2008). Modeling and Analysis of Real-life Job Shop Scheduling Problems by Petri nets, *Proceedings of 41st Annual Simulation Symposium,* ISBN 978-0-7695-3143-4, Ottawa, April 2008

Żabiński, T. & Mączka, T. (2010). Human System Interface for Manufacturing Control - Industrial Implementation, *Proceedings of 3rd International Conference on Human System Interaction HIS*, ISBN 978-1-4244-7561-2, Rzeszów Poland, May 2010



Production Scheduling
Edited by Prof. Rodrigo Righi

ISBN 978-953-307-935-6
Hard cover, 242 pages
Publisher InTech
Published online 11, January, 2012
Published in print edition January, 2012

Generally speaking, scheduling is the procedure of mapping a set of tasks or jobs (studied objects) to a set of target resources efficiently. More specifically, as a part of a larger planning and scheduling process, production scheduling is essential for the proper functioning of a manufacturing enterprise. This book presents ten chapters divided into five sections. Section 1 discusses rescheduling strategies, policies, and methods for production scheduling. Section 2 presents two chapters about flow shop scheduling. Section 3 describes heuristic and metaheuristic methods for treating the scheduling problem in an efficient manner. In addition, two test cases are presented in Section 4. The first uses simulation, while the second shows a real implementation of a production scheduling system. Finally, Section 5 presents some modeling strategies for building production scheduling systems. This book will be of interest to those working in the decision-making branches of production, in various operational research areas, as well as computational methods design. People from a diverse background ranging from academia and research to those working in industry, can take advantage of this volume.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Andrzej Bozek (2012). Using Timed Coloured Petri Nets for Modelling, Simulation and Scheduling of Production Systems, Production Scheduling, Prof. Rodrigo Righi (Ed.), ISBN: 978-953-307-935-6, InTech, Available from: http://www.intechopen.com/books/production-scheduling/using-timed-coloured-petri-nets-for-modelling-simulation-and-scheduling-of-production-systems