Scheduling With Chameleon Nets

Tamás Kis[1] Klaus-Peter Neuendorf [1,2] Paul Xirouchakis[1]

[1]Swiss Federal Institute of Technology
Mechanical Engineering Department
Design and Production Informatics Lab.
CH-1015 Lausanne, Switzerland

e-mail: tamas.kis@epfl.ch

[2] Humboldt-University Berlin
Computer Science Department
Axel-Springer Str. 54A
D-10117 Berlin, Germany

e-mail: neuendor@informatik.hu-berlin.de

Abstract

Chameleon nets are high level Petri nets based on the multi-level Petri net model of Object Systems introduced in [Valk 95]. The original model is extended with time thus allowing hierarchical and modular specification of real time systems. The original synchronisation mechanism is modified by the explicit definition of all possible synchronised transition steps. The general unfolding construction gives an equivalent Time Petri net which defines the semantics of the system at a larger granularity and serves this way as a first basis for the analysis of Chameleon nets. Important for scheduling applications is the use of the right time model as shown by an example. Finally some special analysis techniques, like symmetries and linear invariants of Chameleon nets are discussed.

Keywords: *High level Petri nets, Time Petri nets, scheduling.*

1. Motivation

Scheduling of discrete part manufacturing workshops has great practical importance as well as impact on the performance of the shop that in turn could be responsible for the success of an enterprise. Optimisation in terms of time and cost is usually complicated due to the combinatorial explosion of the inevitable sequencing and assignment decisions in the course of scheduling.

Our interest here is *Job-Shop Scheduling*, which serves as a template for many practical applications. In particular, we want to formalise the basic problem with the extensions of sequence dependent setups and alternative routings for the jobs. Furthermore, we aim to generate schedules that minimise the maximal completion time or makespan. For further clarifications on scheduling terminology and for an overview on scheduling problems and algorithms the reader is referred to [MacCarthy and Liu 1993] or to any textbook about scheduling.

A Job-Shop Scheduling problem has different components that must be represented by any formalisation. These components are the jobs, the machines with their capabilities and the relation between jobs and machines. Our detailed problem statement is given as follows.

The essential information attached to each job is its *routing plan*. A routing plan is given as an *And-Or graph*, whose nodes and hyperedges specify the alternative routings available for completing the job. We do not give a formal treatment of And-Or graphs and routing-plans, which is done elsewhere [Cho et al. 1994], we will provide a Petri net formalism instead. The nodes of the routing plans are *operations*. Each operation specifies a machine, the required setup of the machine and the machining time on that machine in the specified setup.

The capabilities of the machines are regarded merely as available setups and setup changing actions. By a setup of a machine is meant its tooling and other services that can be reconfigured by changing the setup. We have the assumption that the possible setups of the machines are pre-calculated based on the job-mix.

All actions in a workshop takes some time so that the machining operations and setup changing actions have a time interval attribute representing the uncertainty in the course of physical world execution.

In the sequel we first introduce Chameleon nets that are two level Petri nets augmented by time. Then we show an unfolding construction resulting in a Time Petri net. Section 4 deals with a scheduling example. The last section is engaged with the analysis of two-level Petri nets.

2. Chameleon nets

In this section we are going to define a two level Petri net model which incorporates a hierarchy of nets. We do not aim at the deep discussion of hierarchical nets of the here proposed kind, we restrict ourselves to simple cases that are powerful enough to express our scheduling problem. For extensions to higher net classes and problems related to that see [Valk 95, 96]. First we introduce object systems that will be enriched by time in a second step.

Definition 2.1 (Object Systems**)**

The tuple OS = (SN, TN, Y) is called an *object system* where

(1) SN is the *system net*,

(2) $TN = \{TN_j \mid 1 \leq j \leq n\}$ is the set of *token nets*, and

(3) Y is the set of *synchronised transition steps*,

if the following holds:

1) The coloured Petri net $SN = (\Sigma, P, T, F, C, V, E, D_0)$ is called a *system net*, if

   a) $\Sigma = \Sigma_1 \cup \Sigma_2$ is the set of all used colour types. The *basic colour types* are the elements of $\Sigma_1 = \{C_k \mid C_k \subseteq TN$, where $\{C_k\}$ is a partitioning of $TN\}$. Σ_2 is a set of cross products of elements of Σ_1, the *product colour types*,

   b) P and T are the finite, nonempty sets, the *places* and *transitions*, with $T \cap P = \emptyset$.

   c) $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation* or the set of *arcs*,

   d) $C: P \rightarrow \Sigma$ is the *colouring function* of the places,

   e) V is the set of *variable symbols*. Each variable symbol v has a type, denoted as Type(v), which should be a basic colour type.

   f) $E: F \rightarrow V^+$ is the *arc labelling function* that assigns to each arc a tuple (one or more components) of distinct variable symbols.

   g) $D_0: P \rightarrow \cup\{2^{Sk} \mid S_k \in \Sigma\}$ is the initial *token net distribution*, such that for each $p \in P$: $D_0(p) \subseteq C(p)$ holds, and each token net TN_j is assigned to exactly one place of the system net having proper colour type, i.e. either TN_j is an element of the basic colour type of the place or it is an element of one of the components of the product colour type of the place.

   There are some restrictions on the flow relation, on the colouring function and on the arc labelling function to assure that the system net behaves like a token moving net, no token net can be created or removed from the system. Thus, we have the following restriction on the arc labelling function, which implicitly restricts the flow relation and the place colouring function as well.

   The arc labelling function E has to assign a single variable $E(a) = (v)$ to an arc $a \in F$ whose adjacent place p is coloured by a basic colour type, i.e. if $C(p) = C_j$ then $Type(v) = C_j$ should hold. If the adjacent place p has the product colour type $C(p) = C_1 \times C_2 \times \ldots \times C_k$, then E should assign a tuple of different variable symbols with corresponding types to the arc a, i.e. $E(a) = (v_1, v_2, \ldots, v_k)$, $Type(v_j) = C_j$, and for all $i \neq j$: v_i and v_j are different variable symbols. It is required that the variable symbols occurring on the incoming arcs of a transition *t* are different, i.e. for all *p*, *p*' $\in$ F*t*: *p* $\neq$ *p*' implies that E((*p*, *t*)) does not share any variable symbol with E((*p'*, *t*)). Let V_t denote the set of variable symbols occurring on the incoming arcs of *t*. The set of variable symbols used to label the outgoing arcs of *t* should be precisely V_t, where each variable occurs exactly on one outgoing arc and only once (token flow principle).

h) The token nets $TN_j = (P_j, T_j, F_j, M_0^{(j)})$ are P/T nets. It is expected that for all j and k: $P_j \cap T_k = \emptyset$; and also for all j $\neq$ k: $P_j \cap P_k = \emptyset$ and $T_j \cap T_k = \emptyset$; and for all k: $T \cap T_k = \emptyset$, $T \cap P_k = \emptyset$, $P \cap P_k = \emptyset$ and $P \cap T_k = \emptyset$.

i) The set Y comprises the synchronised transition steps that "potentially" could occur in the object system. A synchronised step *s* always involves one system net transition and one or more token net transitions. The general structure of a step $s \in Y$ is given as follows: $(t, (v_1, t_{i1}), (v_2, t_{i2}), \ldots, (v_r, t_{ir}))$, where transition *t* is a system net transition, the symbols $v_1, v_2, \ldots, v_r$ are the variable symbols of V_t and t_{ij} are token net transitions. It is required that for each transition t_{ij} the token net TN_{ij} to which t_{ij} belongs is a member of Type(v_j); and the token nets TN_{ij} be different. It is also expected that each system net transition is a member of at least one step. We say that the token net transition *t* is *autonomous* if it is not included in any synchronised step *s*, otherwise we say that *t* is *synchronised*.

In the sequel we need the notion of a binding. Given a system net transition *t*. The function b: $V_t \rightarrow TN$ is a *binding of the variables of t*, if it is injective and assigns to each variable symbol $v \in V_t$ a token net from Type(*v*);. Observe however, that a synchronised transition step *s* uniquely identifies a binding. Namely, if the system net transition in *s* is *t* and the token net transitions in *s* are $t_{i1}, \ldots, t_{ir}$ whose respective token nets are TN_{ij}, then there exists a unique binding *b* which binds $v_j \in V_t$ to TN_{ij}. We denote this binding as b_s.

There are two difficulties to model object systems purely as coloured nets. The first is that we want to allow token nets to change their state without any transition occurrence of the system net. To model this in a pure coloured net version one needs to introduce more places and transitions in the system net what is not convenient for the representation. The second reason is that we want to introduce time on the token net level and it is probably syntactically difficult to lift it in a concise manner to a global coloured net.

Semantics of Object Systems

So let us now define the semantics of the object system. We first define its states. A state of an object system is given by (1) the distribution of the token nets in the system net, i.e. a function D: $P \rightarrow \{2^{S_k} \mid S_k \in \Sigma\}$, which has the same properties as the initial token net distribution D_0, and (2) the markings of all token nets. So, a state M of an object system is a tuple of the form $(D, m_1, \ldots, m_n)$. Note that this definition is inspired by [Valk 1995].

The firing rule of the object system is rather complex. Suppose that the actual object system state is M. We distinguish two cases

1) *Autonomous transition occurrences*. An autonomous transitions of a token net is enabled whenever there is at least one token on each of its input places. Autonomous transition occurrence affects only the corresponding token net state component of the actual object system state. System net transitions are supposed to occur not autonomously.

2) *Synchronised transition step occurrence*. A synchronised transition of a token net can occur only together with some other transitions. We define such a transition occurrence with respect to a fixed synchronised transition step *s*.

   a) The synchronised transition step *s* is *enabled*, if the token net transitions of *s* are enabled in their respective nets and its corresponding binding b_s is enabled in state M. Binding b_s is *enabled* in M, if for all $p \in$ F*t* and for all components v_k of E((*p*, *t*)): $b_s(v_k)$ is a token net TN_k such that $TN_k \in D(p)_k$, where *t* is the system net transition component of *s* and v_k is the *k*-th component of the arc inscription E((*p*, *t*)).

b) If s is enabled then it can *fire*. Firing of step s consists of the simultaneous firing of the system net transition t of step s and the firing of the token net transitions t_{ij} of s. When t fires it removes the token nets bound by b_s from its input places and puts them on the output places while respecting the labelling(s) of the output arc(s). So, the firing of a synchronised transition step always involves the concurrent firing of a set of transitions.

Now we have arrived to the introduction of time in the model. Here it turns out that it is possible to introduce a time concept by attaching time information only to the transitions of the token nets, whereas the system net transitions remain untimed. So we define

Definition 2.2 (Chameleon nets**)**

The tuple TOS = (SN, TN, Y) is a *Chameleon net*, if

1) TOS is an object system.
2) The token nets $\{TN_j\}$ are *Time Petri nets*, with the extension that some transitions of the token nets are untimed. That is $TN_j = (P_j, T_j, F_j, Eft_j, Lft_j, M_0^{(j)})$, where $(P_j, T_j, F_j, M_0^{(j)})$ is a P/T net and Eft_j and Lft_j are partial functions that assign the *earliest* and *latest* firing times to the transitions of TN_j. It is supposed that $dom(Eft_j) = dom(Lft_j) \subseteq T_j$ and for all $t \in dom(Eft_j)$ the inequalities $0 \leq Eft_j(t) \leq Lft_j(t) \leq \infty$ hold.
3) The following two syntactical restrictions are satisfied:
   a) In each step $s \in Y$ there is exactly one token net transition with time information.
   b) All autonomous transitions of all token nets must posses time information.

Our definition of Chameleon nets aims to introduce time to object systems in a coherent manner. Thinking of synchronised transition step occurrences it is to be avoided that in a step two or more transitions determine the timing of the step thus leading to some difficult semantics. On the other hand, there is no reason to allow autonomous token net transitions without time information. These are the motivations behind 3a) and 3b).

Semantics of Chameleon Nets

We start with the definition of the state of a Chameleon net. A state M is a tuple of the form (D, clock, $(m_1, clock_1), \ldots, (m_n, clock_n)$). In our definition we use informally the notion of Time Petri nets [Starke, 1995].

1) The first component D gives the token net distribution on the places of the system net.
2) The second component “clock” is a function defined on the set of synchronised steps Y. The range of clock is the set of non negative rational numbers and the extra element #. The value clock(s) is # if the synchronised step s is not enabled in the current system state. Whereas clock(s) ≥ 0 means that step s is continuously enabled for clock(s) time. For each synchronised step s we define Eft(s) and Lft(s) as the earliest and latest firing time, $Eft_j(t_j)$ and $Lft_j(t_j)$, inherited from the only time transition $t_j \in s$.
3) The components $(m_1, clock_1), \ldots, (m_n, clock_n)$ describe the state of the token nets, i.e. m_j is a marking of TN_j and $clock_j$ is a partial function from the transitions of TN_j to the non negative rational numbers and to the extra element #. The domain of $clock_j$ is the set of autonomous transitions of TN_j. If $t \in T_j$ is an autonomous transition, then $clock_j(t)$ measures the time elapsed from the time point when transition t got enabled last time, while $clock_j(t)$ = # means that t is not enabled. We assume that all clocks run at the same speed.

The initial state is $M_0 = (D_0, clock, (M_0^{(1)}, clock_1), \ldots, (M_0^{(n)}, clock_n))$ and all clock values are set either to # or 0 according to the enabledness of the respective transitions.

For the ease of defining the state changes, we extend the function clock($\cdot$) in such a manner, that it is defined on all synchronised steps s and on all autonomous token net transitions, too. Its value is defined by the respective clock values. We extend the domains and ranges of the functions Eft(·) and Lft(·) to all autonomous token net transitions in a similar way.

We define the state changes of a Chameleon system with respect to an object system state M. The state M changes if either (1) time passes or (2) some transitions fire.

1) In state M time $\tau > 0$ *can pass*, if for all enabled $s \in Y$: clock(s)+$\tau \leq$ Lft(s); and if for all enabled autonomous transitions t: clock(t)+$\tau \leq$ Lft(t). In this case the clocks are updated by increasing all clock values, different from #, by time τ; and the token net distribution of the system net and markings of the token nets do not change. Thus the new system state M' differs from M only in the clock values.
2) An enabled synchronised transition step $s \in Y$ can fire in state M if clock(s) $\geq$ Eft(s). Its firing consists of the concurrent firing of all transitions in their respective nets.
3) An enabled autonomous transition t can fire, if clock(t) $\geq$ Eft(t). Its firing consists of removing tokens from its input places and adding tokens to its output places.

   After firing of a transition t or a step s, the clock values of all transitions and all synchronised steps are to be updated according to the following rules:

   a) The clock value of all not enabled synchronised transition steps and of all not enabled autonomous transitions is #.
   b) The clock value for all newly enabled steps or transitions is set to 0. An enabled step or transition is *newly enabled*, if either its clock value before firing was #, or it fired and is enabled in the new system state.
   c) If a step or transition remains enabled and is not newly enabled then its clock value is unaffected.

Some semantic aspects of our system motivated the choice of the name *Chameleon net*. Namely, token nets can change their own state without interactions with each other and with the system net.

The definition of the reachability graph of a Chameleon net is a little bit more involved. Intuitively one should follow the reachability graph generation rules of a "standard" Time Petri net [Starke, 1990], but of course consider the state changes at both the system net and token net levels. However, in this paper we do not provide a computational method. Instead, we will give an unfolding algorithm that unfolds a Chameleon net to an ordinary Time Petri net. For the unfolded net one can compute the unfolded reachability graph in the usual way.

3. Unfolding of Chameleon nets

Our goal is now to give hints to an algorithm that takes a Chameleon net TOS as input and constructs a Time Petri net N that essentially encapsulates the same information as the input net. We do not provide our algorithm in a formal programming language, but we show the construction itself, instead.

So, given a Chameleon net TOS = (SN, TN, Y) as in Def. 2.1., then the *unfolded* Time Petri net is defined as N = (P', T', F', Eft, Lft, M_0') with the following properties:

1) The set of places P' of N is defined as the union of the sets $P_1, P_2, \ldots, P_n$ and $P^{(1)}, P^{(2)}, \ldots, P^{(n)}$, where P_j is the set of places of the token net TN_j and for all j there is a one to one correspondence between the elements of P and the elements of $P^{(j)}$, i.e. the latter ones are copies of the places of the system net. For all system net places $p \in P$ let $p^{(j)} \in P^{(j)}$ denote the copy of p in $P^{(j)}$. We will refer to these places as *token flow monitoring places*.

   Intuitively, we want to follow the movement of each token net in the system net. Since we have only black tokens in N, we must have as many copies of the places of the original system net as token nets. Then each copy $P^{(j)}$ is devoted to one token net to indicate its position in the system net SN.
2) The set of transitions T' is the disjoint union of the sets $T'_1, T'_2, \ldots, T'_n$ and T^*, where T'_j is the set of autonomous transitions of the token net TN_j; and $T^* = Y$, i.e. the set of synchronised transitions steps of

TOS. Elements of T^* are denoted as t^*. We are going to model each synchronised transition step explicitly in the Time Petri net N.

3) The flow relation $F' \subseteq (P' \times T') \cup (T' \times P')$ is defined with the aid of the flow relations $\{F_j\}$ of the token nets and the flow relation F of the system net.

   a) The arc $(p, t) \in F_j$ is an arc in F' iff t is an autonomous transition of TN_j and p is one of its input places. Symmetrically, the arc $(t, p) \in F_j$ is an arc in F' iff t is an autonomous transition of TN_j and p is one of its output places. Thus we want to have a copy of each token net, but without their synchronised transitions.

   b) If $t^* \in T^*$ is a transition corresponding to the synchronised step s of TOS, then we have the following arcs in F':

      i) For all synchronised token net transitions $t_j \in s$ if $(p, t_j) \in F_j$ then $(p, t^*) \in F'$ and if $(t_j, p) \in F_j$ then $(t^*, p) \in F'$.

      ii) Suppose that the system net transition in step s is t. The arc $(p^{(j)}, t^*) \in F'$ if the arc $(p, t) \in F$, i.e. p is an input place of t, and the token net TN_j (to which t_j belongs) is assumed on place p by step s. Similarly, if the arc $(t, p) \in F$ and the token net TN_j is to be put on place p according to s then $(t^*, p^{(j)}) \in F'$ holds.

   Transition t^* glues together the synchronised transitions of the synchronised step s and removes and adds tokens both from the token nets and the token flow monitoring places.

   c) The functions Eft and Lft are defined with the aid of Eft_j and Lft_j as follows

   d) For all autonomous transitions $t \in T_j$: $Eft(t) = Eft_j(t)$ and $Lft(t) = Lft_j(t)$.

   e) For all transitions $t^* \in T^*$: $Eft(t^*) = Eft_j(t_j)$ and $Lft(t^*) = Lft_j(t_j)$, where t_j is the timed member of synchronised step s, where s is the synchronised step represented by t^*.

   f) The initial marking M'_0 is defined by the initial token net distribution D_0 of TOS and the initial markings of the token nets of TOS.

   g) For places of the token nets, $p \in P_j$ implies that $M'_0(p) = M_0^{(j)}(p)$,

   h) For each $j \in [1..n]$: exactly one of the places in $P^{(j)}$ is marked. Namely if the token net TN_j is on place $p \in P$ in the initial token net distribution D_0, then the copy $p^{(j)}$ of p is marked, i.e. $M'_0(p^{(j)}) = 1$. For all other places $p' \in P^{(j)}-\{p^{(j)}\}$: $M'_0(p') = 0$.

We do not give a formal proof of the correctness of our construction, either. More importantly, we show how one can use our definition of Chameleon nets to describe a scheduling problem, and then analyse it after applying the unfolding algorithm.

4. Scheduling Example

Now we are concerned with a simple scheduling problem with 3 jobs and 2 machines. First we specify the jobs and machines by means of high level modelling techniques and then we analyse it to get the optimal schedule with respect to maximum completion time or make-span.

The specification of one job out of the three is depicted in Figure 1. The net essentially represents the *routing-plan* of the job. In this routing plan there are three *operations*, O_1, O_2 and O_3, but only two of them must be executed in order to complete the job. Transition "begin" signals the starting of the job by removing the token from place p1, but it has no resource requirements in terms of machines. Each operation O_j is represented by two transitions, s_j and f_j, that model the *starting* and *finishing* of the operation, respectively. The order of operations is as follows: first O_1 should be executed, then comes either O_2 or O_3 but only one of them. The completion of the job is signalled by putting a token on place p7 by firing transition "end".

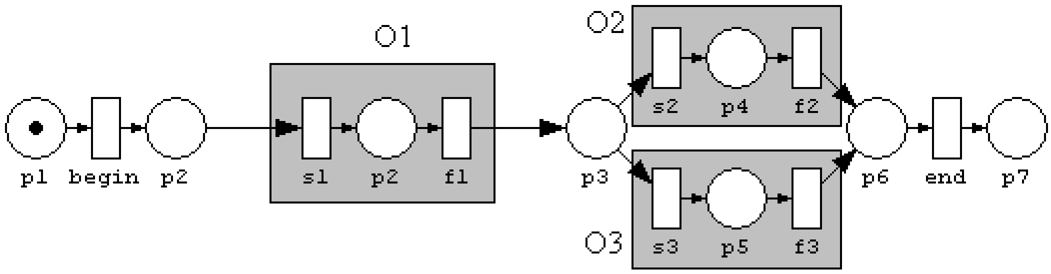

Figure 1. *The routing plan of the first and the second job.*

The timing attributes of the transitions are as follows. Transitions "begin" and "end" have the interval [0, 0]. All operation start transitions s_j have the interval $[0, c]$ where $c \geq 0$ is a common timing parameter to be discussed later, while to each finish transition f_j is attached the machining time t_j of the represented machining operation. The following table summarises all necessary information about these three operations.

Operation	Machine	Setup	Machining time
O_1	M_1	U_1	1
O_2	M_1	U_2	1
O_3	M_2	U_3	3

Table 1. *The specification of the operations of the routing plan in Figure 1.*

In our example two jobs have identical routing plan, while the third one calls for a different machining process depicted in Figure 2.

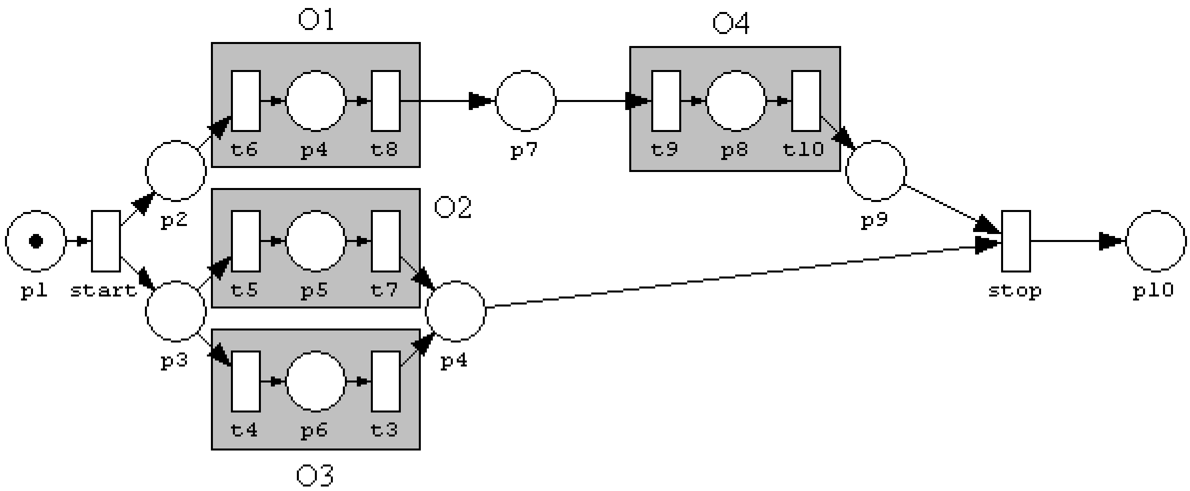

Figure 2. *The routing plan of the third job. It comprises four operations.*

The corresponding table of operation specifications is listed in Table 2. We will refer to these job nets as JN1, JN2 and JN3.

Now we turn to the specification of the capabilities of the machines. The first machine M_1 is capable to work in two different setups. This is coded in the form of a *setup-plan*. The setup plan in Figure 2 comprises four modules: two of them for using the setups and the another two are the setup changing operations, i.e. for changing from setup1 to setup2 and vice versa.

Operation	Machine	Setup	Machining time
O_1	M_1	U_2	1
O_2	M_2	U_2	2
O_3	M_2	U_1	2
O_4	M_2	U_3	3

Table 2. *The specification of the operations of the routing plan in Figure 2.*

The detailed description of these modules is as follows. The modules Setup-1 and Setup-2 represent actions and status of using setups 1 and 2, while modules Change 1->2 and Change 2->1 constitute the changing actions between setups. The structure of module Setup-1 is as follows. Place avail-1 is marked iff the first setup of the machine is available for use. The setup can be either continuously used modelled by transition con-1 or changed. The transition su-1 stands for starting some work on the machine using the first setup, while transition fu-1 is the respective finishing action. Each setup changing module has two transitions and one place in between. For instance in module Change 1->2 the transition s_{12} represents the start of setup changing, while transition f_{12} represents the finishing of setup change. In a setup net only the setup changing and continuing actions posses time information. Transitions con-j and s_{ij} have the interval [0, 0], whereas all of the transitions f_{ij} have common firing time 4 to change between Setup-i and Setup-j.

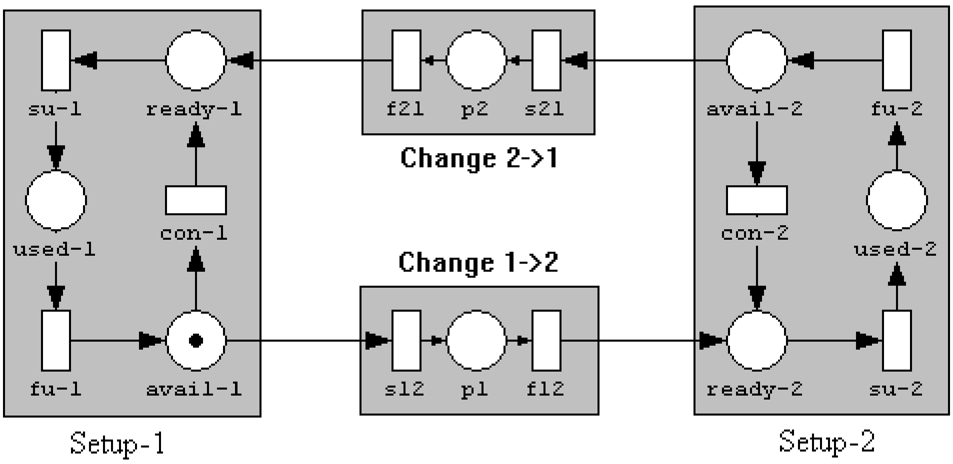

Figure 3. *The setup plan of machine M_1.*

The other machine M_2 is capable to work in three different setups, thus its setup plan comprises 3 setup modules and 6 setup changing operations.

These two types of nets, i.e. the job nets and setup nets, are the token nets of the system net, which reflects the workshop layout at a bird's-eye view (see Figure 4).

The system net has three modules, two machines and the common *Work-in-Progress* (WIP) place. Consider the module Machine-1 formed by the transitions M-1-S and M-1-F, and the places M-1-T and M-1-U. The work on Machine-1 starts with M-1-S which removes the setup net UN1 from the place M-1-U and an appropriate job net, in this case JN1 from the place WIP, and puts them together on the place M-1-T. The work is finished by M-1-F, which moves back the token net UN1 to the place M-1-U and also the job net to the place WIP.

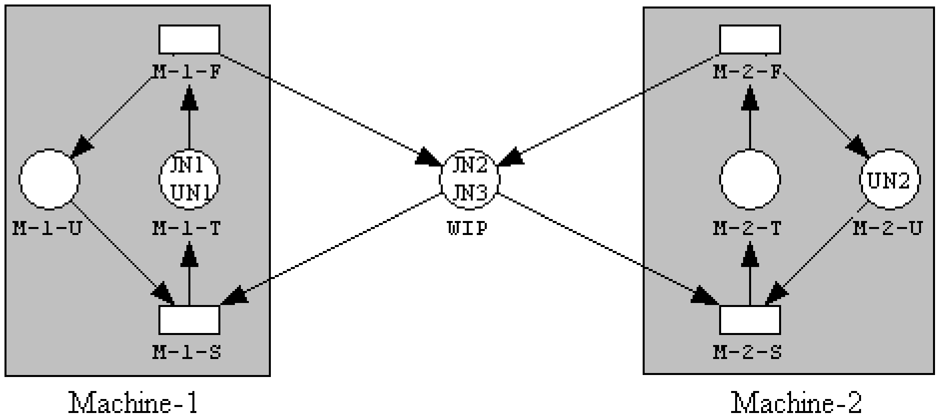

Figure 4*. A snapshot of the system net.*

The transitions of the system net lack any time information. Initially all token nets sit on place WIP, and setup net UN1 is on place M-1-U, while UN2 is on place M-2-U.

The synchronisation between the system net and the token nets aims to bring together the right job with the right machine which has the right setup to carry out the next operation on the job. That is, all synchronised steps comprise exactly one system net transition, one job net transition and one setup net transition. Our naming convention allows us to say that the start of a machining operation of a job net is synchronised with the start of using a setup and the start of using a machine. Symmetrically, the finishing of a machining operation is synchronised with the finish using a setup and machine. The only restriction of forming these triples comes from the content of machining operations: each machining operation prescribes a particular machine and a particular setup of that machine.

Now our next goal is to use the model for generating schedules that minimise the completion time of all jobs, i.e. the time needed to finish all of the jobs assuming that we begin them at time zero. We can unfold our high level net to a "flat" Time Petri net and then find a minimal time path from the initial state to a final marking. A final marking is characterised by the property that it has in all job nets a token on the output place of its "end" transition. Our computational experiments are briefly summarised in the sequel.

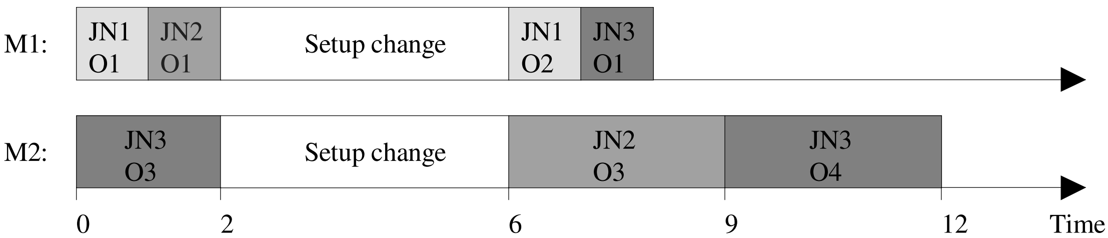

Figure 5*. The best schedule obtained with greedy scheduling.*

Assume that the machine operation times and setup changing times are given (they may come from standards or statistics made at the workshop). The only freedom is in the choice of parameter *c*. The initial guess is 0, which is a rather greedy heuristic, i.e. machines can not stay idle if there are some jobs that can be processed by them. In this way we model some sort of step semantics since the machining times are surely non-zero numbers. The best schedule computable so far is depicted in the form of a Gantt chart in Figure 5.

However, relaxing the greedy heuristic by letting parameter *c* to take a different value from 0 the firing obligation of synchronised steps is relaxed, too. The enlarged search space enables to find a better solution, the optimal one of the problem, which is depicted in Fig. 6.

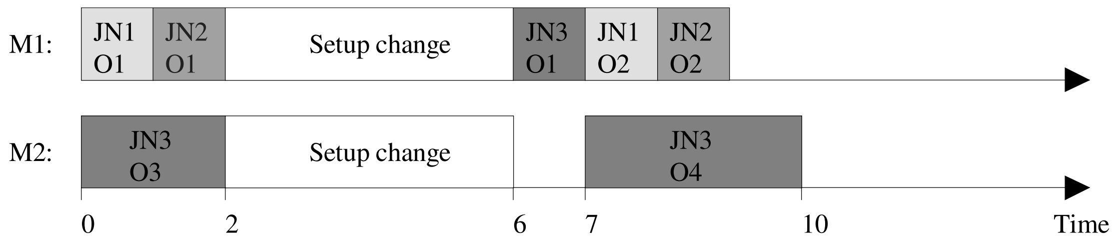

***Figure 6**. The optimal schedule.*

The most important lesson of our computational example is that a careless choice of the time model may hinder to find the best solutions of a scheduling problem.

5. Remarks Towards Analysis

The unfolding of Chameleon nets to Time Petri nets makes the wide range of known Petri net analysis techniques available to the analysis of Chameleon nets. So what remain are questions for possible generalisations of Chameleon nets and how to use the knowledge of the modular structure of a Chameleon net to ease its analysis. We summarise some answers to such questions without giving formal proofs in the following series of remarks.

Remark 5.1 A slightly *generalisation* of Chameleon nets can be given in the following way. The requirement in Definition 2.1 3) that each transition of the system net has to belong to a synchronised transition step can be relaxed so that also autonomous system net transitions are allowed. For an autonomous system net transition *t* that just moves a token net from its only input place to its only output place the unfolded Time Petri net contains a copy of *t* in each "token net position monitoring copy" of the system net places. But for more general transitions the number of copies needed in the unfolding increases much faster. So one needs for a transition with *n* input places where each place can hold *m* different token nets m^n copies of *t* in the unfolding. It seems therefore better to restrict the firing mode of autonomous transitions by other mechanisms to avoid the combinatorial explosion.

Remark 5.2 The *unfolding* of Chameleon nets defined in Section 3 gives not always a connected Petri net, some places may remain isolated. The general unfolding can be simplified by removing all isolated places in the unfolded net. Then only such monitoring places remain which have a chance to be marked or unmarked. Further reduction is often possible since the unfolding contains also sometimes parallel places, i.e. places with the same input and output transitions. For a setup net of our example only two monitoring places are enough (the places M-i-U and M-i-T) instead of the "full copy" of the five system net places.

Remark 5.3 The *symmetry* method to analyse Petri nets is well-known and can be combined with many other analysis techniques [Schmidt 93]. An advantage of Chameleon nets is that the symmetries given by identical token nets (with same synchronisation behaviour) are known in advance and can so be used together with the unfolding in all kinds of the further analysis. Symmetries in the system net are in general not present in the unfolding due to the disturbances of the synchronisation.

Remark 5.4 *Linear invariants* of Chameleon nets can be defined as usual for the unfolded untimed Petri net. The interesting question is how the invariants of the token nets and the system net are related to the invariants of the unfolding. The character of a Chameleon net as a token moving system ensures e.g. always that the system place vector $\mathbb{1}$ (all system places weighted by 1) is an invariant. A look on the incidence matrix of the unfolded Chameleon net shows that using a suitable imbedding of the places of the Chameleon token nets and the system net places to corresponding places in the unfolding one has the property that all linear invariants of the single components remain invariant. This property is similar as for Petri nets with step firing rule. The interesting question if there exist more invariants than those generated by the imbedded invariants of the components and also what kind of invariants depends on the structure of the Chameleon net and the used unfolding procedure. In our scheduling example the number of minimal support invariants is much greater than just the number of the minimal support invariants of the components. But the elimination of one element of a pair of parallel places in the unfolding eliminates of course also the half of all minimal support invariants with non-zero support according to that place pair. For examples and more details about the invariants phenomena we have to refer to a forthcoming paper elsewhere.

6. Acknowledgement

The authors are thankful to the referee for his valuable comments and helpful review.

7. References

Cho, H., Derebail, A., Hale, T. and Wysk R. A., 1994, A Formal Approach to Integrating Computer-Aided Process Planning and Shop Floor Control, *Transactions of the ASME*, Vol. 116, pp. 108-116.

MacCarthy, B. L. and Liu, J., 1993, Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling, Vol. 31, No. 1, pp. 59-80

Neuendorf, K.-P., Kiritsis, D., Kis, T., Xirouchakis P., 1997, Two-level Petri Net Modelling for Integrated Process and Job Shop Production Planning, ICAPTN‘97, proceedings of the workshop Manufacturing and Petri Nets, pages 135-150.

Schmidt, K., 1993, Symmetries of Petri nets. In Petri Net Newsletter No. 43, pages 9-25.

Starke, P., 1995, A Memo on Time Constraints in Petri Nets*, Informatik-Berichte der Humboldt-Universität zu Berlin* No. 46.

Valk, R., 1995, Petri nets as dynamical objects, *1st Workshop on Object-Oriented Programming and Models of Concurrency,* 27 June 1995, Turin, Italy.

Valk, R., 1996, *On Processes of Object Petri Nets*, University of Hamburg – Computer Science Department, Report No. 185/96.