Chapter 6

Workflow Modelling Based on Synchrony

Chongyi Yuan

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/48375

1. Introduction

Prof. Carl Adam Petri wrote: "In order to apply net theory with success, a user of net theory can just rely on the fact that every net which he can specify explicitly (draw on paper) can be connected by a short (≤ 4) chain of net morphisms to the physical real word; your net is, in a very precise sense, physically implementable." (Status Report On Net Theory, 1989, a forward for my book Petri Nets in Chinese[1]).

Why a net is physically implementable? The reason is, every concept in net theory is carefully chosen based on nature laws, and well defined in terms of precise mathematics and logic. For example, the concept of global time does not belong to net theory. Time measured with real numbers exists only in theories like theoretical physics. Logical time does not exist in the real world. For net theory, time is just "clock reading", a measurement of physical changes. Global time is not realistic for systems in which a shared clock is not available.

On the other hand however, it is easy to find in the literature, that many an author introduces new concepts into his or her Petri net with implementation totally forgotten. "Timed Petri Net" is just one of such examples.

As one of the chapters in this book on Petri nets, implementable concepts and only implementable concepts will be introduced.

We start with the definition of a directed net, which is the most fundamental concept in net theory. The next two sections serve to keep this chapter self-reliant.

This chapter is organized as below:

Sections 2 and 3 recall basic definitions of Petri Nets: The concept of directed net deserves a separate section since it is the foundation of the whole net theory. Section 3 is mainly about Place/Transition-systems, based on which workflow models are to be constructed.

Section 4 is an introduction of synchrony, a branch in net theory on transition synchronization that provides theoretical support to workflow modelling.

Section 5 talks about business processes, the subject of workflow research. A full understanding of the concept of business processes makes a good start.

Section 6 proposes the concepts of synchronizers and workflow logic. A synchronizer connects transitions in two consecutive steps in a business process, and workflow logic is obtained when all transitions in a business process are so connected. Properties and analysis methods of workflow logic are defined and proposed. A transition in workflow logic represents a business task while a synchronizer represents a task in workflow management. Workflow logic specifies all possible routes a business case may take when it is processed. An individual business case corresponds to a unique route among all routes given by workflow logic. This route is considered as the semantics of that case. Section 7 defines the concept of case semantics.

Section 8 is about business process management, or automatic management. The dual net of workflow logic is exactly the logic of management, based on which workflow engine conducts the process of individual cases.

Section 9 concludes this chapter with acknowledgement, and the last section is a list of references.

This chapter is about Petri nets and workflow modelling.

There are many ways in the literature to define nets and net systems. For example, the concept of flow relation, namely F, has been made implicit by many researchers. It is often combined with weight function W. Without the flow relation, the concept of directed nets would disappear; and without the concept of directed nets, the whole theoretical part of Petri nets would be without a foundation. Thus, this chapter starts from the definition of directed nets given by C. A. Petri himself.

Petri net systems have been considered as one of the adequate and promising candidates for workflow modelling. The concept of WF-nets proposed by Prof. Aalst from Holland has become popular in the last 10 to 20 years. A team in the Software Development Company of Peking University tried to use WF-nets as a formal model to develop software for a government organization at a time around the year 2000. The WF-nets didn't work, and they didn't know why. The author joint them, and we found problems of WF-nets, leading to failure. The concept of WF-nets was proposed without theoretical foundation. These problems were discussed in our paper titled "A Three Layer Model for Business process" [7]. The concepts of synchronizers, workflow logic and case semantics etc were defined in this paper for the first time. After so many years since then, people interested in workflow modelling remain sticking to WF-nets. Many people do not even know our work, it seems. The reason is, the author guess, General Net Theory (theoretical part of Petri nets) is not popular yet. This chapter shows how important "Synchrony" is to a successful application of Petri nets in the area of workflow modelling.

2. Directed net

Definition 1

A triple N=(S,T;F) is a directed net if $S \cup T \neq \emptyset \wedge S \cap T=\emptyset \wedge F \sqsubseteq (S \times T \cup T \times S) \wedge dom(F) \cup cod(F) = S \cup T$ Where $dom(F)=\{x \mid \exists y: (x,y) \in F\}$ and $cod(F)=\{y \mid \exists x: (x,y) \in F\}$.◆

A directed net used to be called "Petri net". We keep the term "Petri net" to mean "net theory", a term Carl Adam Petri used in *Status Report*. "Net" is often used to mean "directed net" if it causes no ambiguity.

$S \cup T \neq \emptyset \wedge S \cap T=\emptyset$ demands that a net consists of at least one element, and its elements are clearly classified.

$F \sqsubseteq S \times T \cup T \times S$ indicates that direct dependence does not exist between elements in the same class. $dom(F) \cup cod(F) = S \cup T$ excludes isolated elements from a net.

A directed net has a graphical presentation as shown in Figure 1, in which elements in S and T appear as circles and boxes respectively while elements in F appear as arrows (arcs). The arrow from x to y represents (x,y) in F.

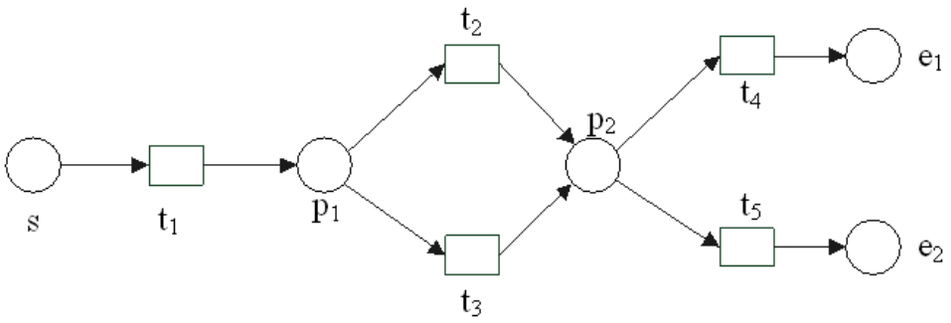


Figure 1. A Directed Net

Petri net (net theory) consists of Special Net Theory and General Net Theory. Special Net Theory focuses on system modelling and General Net Theory focuses on theories supporting system modelling. The concept of directed nets is their common foundation.

Definition 2

Let N=(S,T;F) be a directed net and $X=S \cup T$, For x in X, ${}^{\cdot}x=\{y \mid (y,x) \in F\}$ is the pre-set of x, $x^{\cdot}=\{y \mid (x,y) \in F\}$ is the post-set of x. For $t \in T$, ${}^{\cdot}t \cup t^{\cdot}$ is the extension of t.◆

Special Net Theory (SNT for short) and General Net Theory (GNT for short) are derived from Directed Net based on pre-sets and post-sets of elements as described below.

The concept of extensions of transitions leads to the principle of local determinism.

The S-complementation operation on a net leads to the removal of contact. The T-complementation operation on a net leads to the removal of differences between forward and backward flow of tokens (SNT).

The S-completion operation on a net leads to Synchrony while the T-completion operation on a net leads to Enlogy(GNT).

A directed net implies a unique undirected net that leads to Net Topology (GNT).

A special class of directed nets is the occurrence nets that lead to Concurrency (GNT).

Synchrony will be briefly introduced in Section 3, since it provides guidance to workflow modelling. It is impossible in this chapter to go any further on SNT and GNT. The point is, successful applications of Petri nets rely on both SNT and GNT, not only SNT.

3. Net system

Definition 3

A 6-tuple $\Sigma = (S,T;F,K,W,M_0)$ is a net system if (S,T;F) is a directed net and:

$K: S \to \{1,2,\cdots\} \cup \{\infty\} \wedge W: F \to \{1,2,\cdots\} \wedge M_0: \to \{0,1,2,\cdots\}$ such that $\forall s \in S: M_0(s) \leq K(s)$, where K,W,M0 are respectively the capacity function, the weight function and the initial marking. ◆

A mapping $M: \to \{0,1,2,\cdots\}$ is a marking if it satisfies

$$\forall s \in S: M(s) \leq K(s).$$

Conventionally, elements in S are called places and elements in T are transitions. A place is capable to hold certain kind of resources, with an allowance given by K. Infinite capacity does not mean infinite space for resources. It is just an indication that the capacity of that place is not a factor for a transition to become enabled (see definition below). A transition resembles a change of resource quantities (consumed or produced) and a change in kind. The precise change is given by transition rules. In the rest of this chapter, Σ is always assumed finite, i.e. Σ has a finite number of places and transitions.

Definition 4

1. Transition t is enabled by marking M if $\forall s \in {}^{\cdot}t: W(s,t) \leq M(s) \wedge \forall s \in t^{\cdot}: M(s)+W(t,s) \leq K(s)$. This fact is denoted by M[t>.
2. Transition t may fire, if M[t>, to yield a successor marking M′ given by $M'(s) = M(s) - W(s,t)$ for s in ${}^{\cdot}t - t^{\cdot}$, $M'(s) = M(s)+W(t,s)$ for s in $t^{\cdot} - {}^{\cdot}t$, and $M'(s) = M(s) +W(t,s)-W(t,s)$ for s in ${}^{\cdot}t \cap t^{\cdot}$, and $M'(s) = M(s)$ otherwise. This fact is denoted by M[t>M′.◆

It is easy to prove that the successor M′ is indeed a marking.

A marking represents a resource distribution on S. A transition firing causes the flow of resources along arcs in F, and thus F is called the flow relation of $\sum$.

It is easy to see that M[t> is determined by ${}^{\cdot}t \cup t^{\cdot}$, the extension of t, and the change from M to M′ is confined to ${}^{\cdot}t \cup t^{\cdot}$. This is the principle of local determinism. A marking is a global state, but the transition rules refer to only the extension of a single transition, not the

complete marking. Many system models take global states as a means of system control. But this is not always implementable, since a global state is not always instantly known. It takes time to know the current global state, and this delay may be significant to an effective real time control.

Figure 2 illustrates how to represent a net system graphically. The black dot inside place s is a token, denoting $M_0(s) = 1$. Empty places have no token (resource). Conventionally, M(s)=0, W(x,y) =1 and K(s) = ∞ are shown by default.

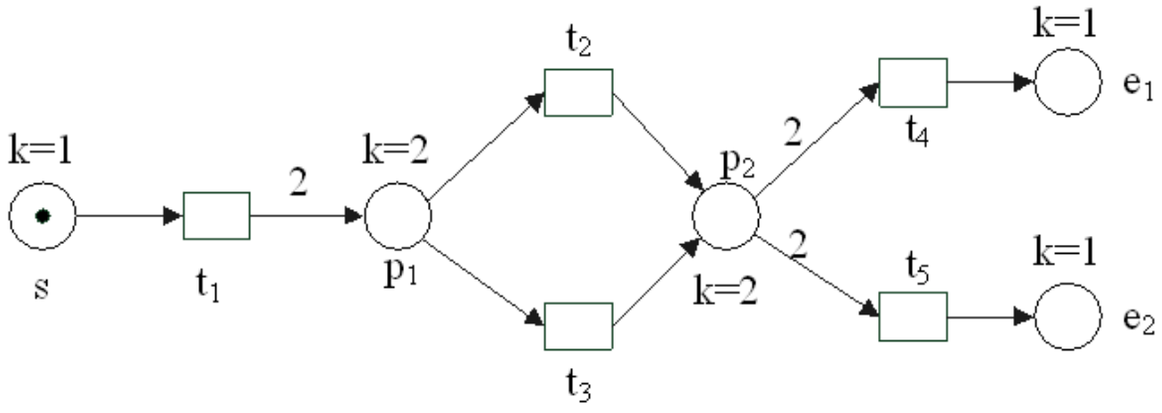


Figure 2. A Net System

Definition 5

Let Σ=(S,T;F,K,W,M0) be a net system, M is a marking and t1, t2 are transitions.

1. If $M[t_1>M' \wedge M'[t_2> \wedge \neg M[t_2>$, then t_1 and t_2 are in sequential relation at M, denoted by $M[t_1,t_2>$.
2. If $M[t_1> \wedge M[t_2>$, and $\forall s \in {}^{\cdot}t_1 \cap {}^{\cdot}t_2 : M(s) \geq W(s,t_1) + W(s,t_2)$ and $\forall s \in t_1{}^{\cdot} \cap t_2{}^{\cdot} : M(s) + W(t_1,s) + W(t_2,s) \leq K(s)$,then t_1 and t_2 are in concurrent relation at M, denoted by $M[\{t_1,t_2\}>$.
3. If $M[t_1> \wedge M[t_2> \wedge \neg M[\{t_1, t_2\}>$, then t_1,t_2 are in conflict at M, denoted by $cf(t_1,t_2,M)$.
4. If there is a transition t and a place b such that $\forall s \in {}^{\cdot}t_1 : M(s) \geq W(s,t) \wedge M(b) + W(t,b) > K(b)$, then t leads to a contact in b at M, denoted by ct(t,b,M). ◆

Two concurrently enabled transitions can fire either concurrently or one after another.

Theorem 1

The marking reached by concurrently fired transitions can also be reached by the transitions fired one after another.◆

This theorem guarantees that the next definition includes markings reached by concurrent transition firings.

Definition 6

The set of markings reachable from M_0 by consecutive transition firings is usually denoted by $[M_0>$.◆

The set $[M_0>$ may be infinite for a finite system. The algorithm for computing $[M_0>$ produces a finite tree, denoted by $T(\Sigma)$, and this tree can be re-structured to become a graph, denoted by $G(\Sigma)$. $G(\Sigma)$ will be used for the computing of synchronous distances later on. All these concepts and algorithms are in the category of techniques, we will go no further here.

So far we have not said a word about how to relate a net to the real world. This reflects an important aspect of Petri net, that is, a net is unexplained. This nature of nets has its good point and bad point. The good point is: the same net or net system may be explained in different application areas to solve different problems; the bad point is: unexplained transition firings lead to general analysis methods that are bound to be of low efficiency, since they cannot make use of application specific properties.

A net is "physically implementable" when every element in $S \cup T$ has an explaination for a fixed application problem, and every transition firing describes real changes in that application area. A net describes how real changes relate with each other. This chapter aims to show how to build, with the guidance of GNT, net systems for workflow modelling and how to find efficient analysis methods.

Some transitions are defined as "instant transitions" in Timed Petri Nets, for they fire instantly when they become enabled. What would happen when two instant transitions are in conflict? Conflict resolution takes time since it needs to be detected and it requires a decision from the system environment. Generally speaking, an enabled transition may be disabled (by others) without firing.

4. Synchrony

Careful observation reveals that sequential relation, concurrent relation and conflict relation are not relations between two transitions, but rather, they are relations between transition firings, i.e. two transitions may fall into one relation at a marking and fall into a different relation at another marking. Thus, a more precise way to denote these relations is: sq(M[t1>, M[t2>), cn(M[t1>, M[t2>), and cf(M[t1>, M[t2>) respectively. Note that sq(M[t1>, M[t2>) is asymmetry.

Synchrony is about how transitions themselves are synchronized. It describes laws exhibited in the course of transition firings. For example, the sunrise and the sunset are alternating "transitions" while a hand-clapping "transition" consists of simultaneous actions of the two hands. One may observe one more sunrise or one more sunset, depending on the times the observation starts and ends, but one always counts the same number of actions of the two hands for hand clapping. The laws exhibited by alternating transitions and simultaneous transitions are given by "the synchronous distance is 1" and "the synchronous distance is 0" respectively.

The concept of synchronous distance was originally defined in terms of events in a C/E-system, which is a model describing changes in the nature, like the changes of 4 seasons. A C/E-system has no initial marking. Instead, it has a current marking. This chapter is about

artificial systems. We have to redefine this concept of synchronous distance to serve our need.

4.1. Synchronous distance in a P/T-system

A net system as defined by Definition 3 and 4 is conventionally called a Place/Transition-system, P/T-system for short.

Definition 7

Let $\Sigma = (S,T;F,K,W,M_0)$ be a P/T-system. A sequence of transitions $\delta = t_1t_2\cdots t_n$ is called a transition sequence if they can fire one after another in the given order, starting from the initial marking. The length of δ is n.

An infinite sequence of transitions is a transition sequence if any of its finite prefix is a transition sequence. ◆

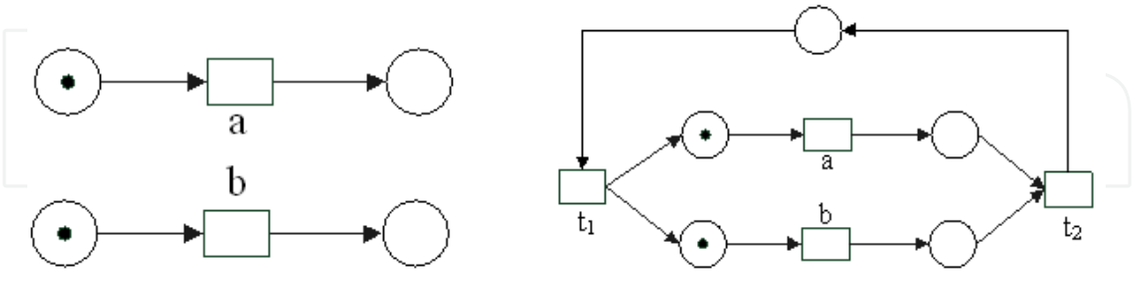
In what follows ρ denotes the set of all transition sequences of Σ.

Definition 8

Let $\Sigma=(S,T;F,K,W,M_0)$ be a P/T-system and T_1, T_2 be subsets of T, $T_1 \neq \emptyset$, $T_2 \neq \emptyset$. Let **δ** be a finite transition sequence, i.e. $\delta \in \rho$. Let $\#(\delta, T_1)$, $\#(\delta, T_2)$ denote the numbers of firings of transitions in T_1,T_2 respectively, and $\#(\delta, T_1,T_2) = \#(\delta, T_1) - \#(\delta, T_2)$. The synchronous distance between T_1 and T_2, denoted by $\sigma(T_1,T_2)$, is defined by

$\sigma(T_1,T_2) = \max\{\#(\delta, T_1,T_2) \mid \delta \in \rho\} - \min\{\#(\delta, T_1,T_2) \mid \delta \in \rho\}$ if exists, other wise $\sigma(T_1,T_2) = \infty$. ◆

Figure 3 (a) is P/T-system Σ1 and its set of transition sequences is {a, ab, b, ba}. We have, for T_1 = {a} and T2 ={b}, $\max\{\#(\delta, T_1,T_2) \mid \delta \in \rho\} = 1$ and $\min\{\#(\delta, T_1,T_2) \mid \delta \in \rho\} = -1$, so $\sigma(T_1,T_2) = 2$.



(a) Σ_1, (b) Σ_2

Figure 3. σ(a,b) =2

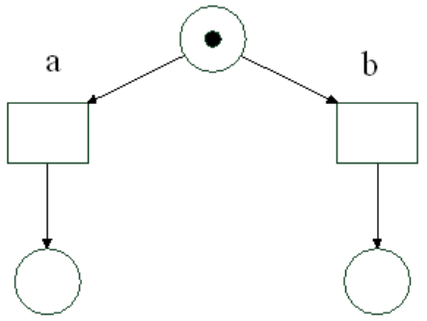
Note that we write σ(a,b) =2 instead of $\sigma(T_1,T_2)$ =2 in figure 3 by convention when both T_1,T_2 are singletons.

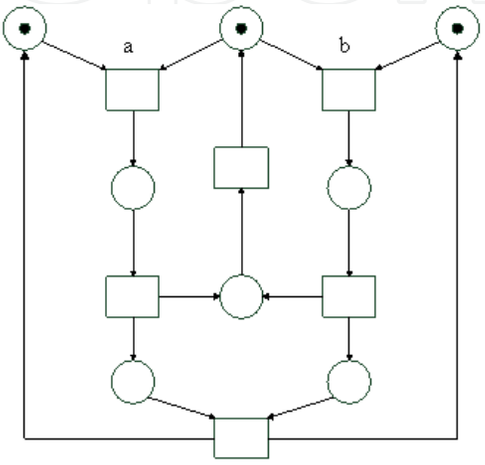
The set of transition sequences of Σ_2 is infinite and it contains infinite sequences. It is easy to check that any repeatable portion in a finite or infinite transition sequence contains the same

number of firings of transition b and transition a. This is why σ(a,b) is finite. The consecutive firings of transition a count at most to 2, so do the consecutive firings of transition b. This is the physical meaning of σ(a,b) =2 for Σ_2.

The distance (i.e. synchronous distance from now on) between left hand action and right hand action is 0, since it is impossible to see only an action of either hand in the course of hand clapping.

Figure 4 (a) shows another situation of σ(a,b) =2, where transitions a and b are in conflict at the initial marking. Figure 4 (b) is still another case of σ(a,b) =2.





(a) Σ_3 (b) Σ_4

Figure 4. σ(a,b) =2: a and b in conflict

Theorem 2

The synchronous distance defined by Definition 8 satisfies distance axioms:

$\sigma(T_1,T_2) = 0$ if and only if $T_1 = T_2$; $\sigma(T_1,T_2) \geq 0$; $\sigma(T_1,T_2) = \sigma(T_2,T_1)$; $\sigma(T_1,T_2) + \sigma(T_2,T_3) \geq \sigma(T_1,T_3)$. ◆

This theorem explains why the concept is called distance. It is assumed that $T_1 \cap T_2 = \emptyset$ when $\sigma(T_1,T_2) > 0$, since $\sigma(T_1,T_2) = \sigma(T_1-T_2, T_2 - T_1)$ by definition.

Theorem 3

$\sigma(T_1,T_2) < \infty$ if and on ly if for any repeatable portion δ of any sequence in ρ, $\#(\delta, T_1,T_2) = 0$. ◆

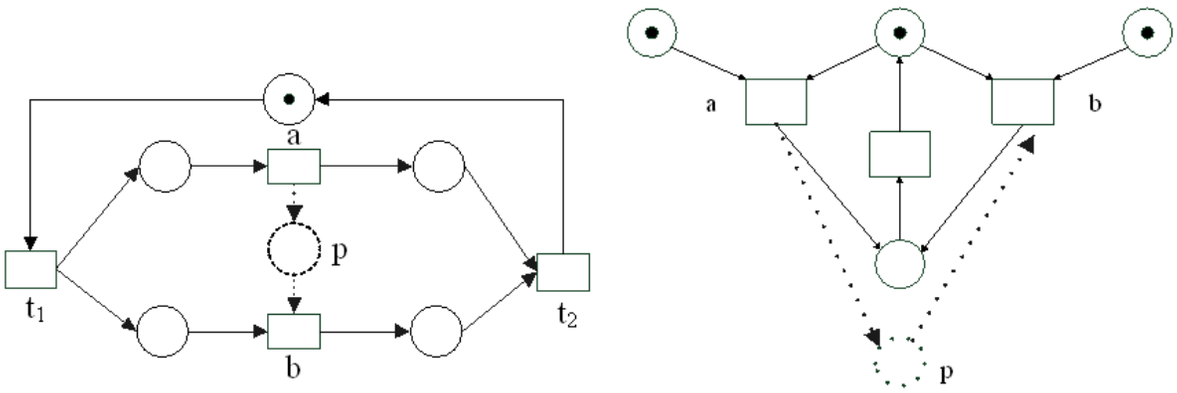
It is easy to prove the above two theorems, so omitted here.

4.2. Transition synchronization and place synchronization

People would think, based on experiences in daily life, that synchronization is something participated by different parties like hand clapping, or some events occurring at the same time like a live casting on TV with an on-going game. Such synchronization is characterized by distance 0, since one cannot see one hand clapping or see the TV show without the game.

We have said that hand clapping is a single transition consisting of actions of two hands; A live show and the on-going game would appear in a net as one transition as well since if they were separated as different transitions in parallel, ordered firings would produce the same effect , but this cannot be true. Synchronization characterized by distance 0 is "transition synchronization". Synchronization with $\sigma(T_1,T_2) > 0$ is "place synchronization" since such synchronization is achieved via places and it can be observed by taking an added place as an observation window.

System Σ_5 in Figure 5 (a) is the same system as shown in Figure 3 (b) with an added place p denoted by a dotted circle and connected to transitions by dotted arrows. This added place does not belong to Σ_5, it is to be used for observations.



(a) Σ_5 (b) Σ_6

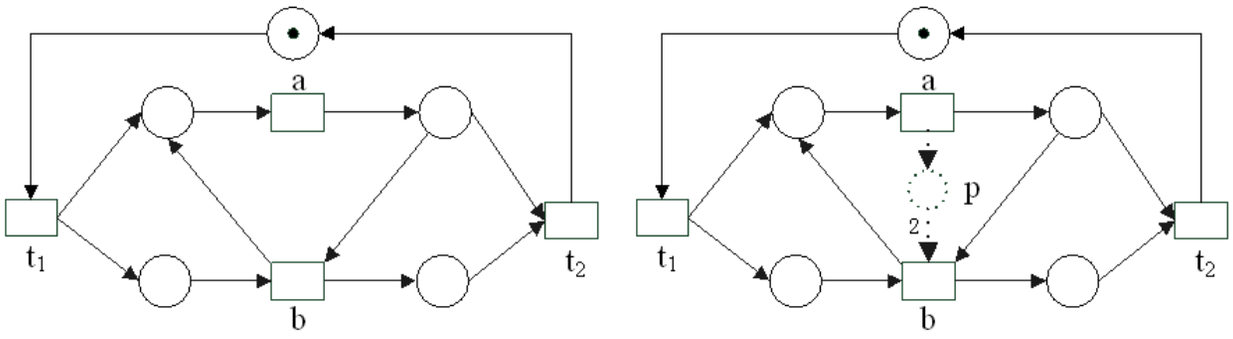
Figure 5. Added Place as Observation Window

An observer records what he finds through the window by putting a token into p when transition a fires and removing a token from p when transition b fires. It is assumed that place p has enough tokens, say n tokens, to start with, so the recording would not be interrupted. The maximum number of tokens in p is n+1 while the minimum is n-1. So the difference is 2 between transition b and transition a. The same observation applies to Σ_6 to get the same distance.

The observation window p for disjoint transition sets T_1 and T_2 is an added place whose input arcs are from T_1 and output arcs are pointing to T_2. Place p gets a token whenever a transition in T_1 fires and loses a token whenever a transition in T2 fires. Place p has enough tokens to start with to ensure a smooth observation. In case the number of tokens in p is not bounded, the distance between T_1 and T_2 is ∞. Otherwise, the difference between the maximum and the minimum is the distance.

It is easy to find that $\sigma(a,b) = \infty$ for the P/T-system in Figure 6 (a), since the repeatable sequence t_1abat_2 contains two firings of a and only one firing of b. The added place p as shown in Figure 6 (b) has weight 2 on the arc from p to transition b, it would have n+1 tokens after the first firing of a and n-1 tokens after the firing of b. The difference is 2.This is a weighted distance between b and a: $\sigma(a,2b) = 2$, the weight for a is 1 while the weight for b is 2. The concept of weighted synchronous distances makes a distinction when infinite

distance is encountered. For simplicity, the formal definition of weighted synchronous distances is omitted here.



(a) Σ_7 (b)Σ_8

Figure 6. Weighted Synchronous Distance

Generally speaking, T_1 and T_2 may have more than one transition. It is possible that all transitions in the same set share the same weight, it is also possible that different transitions need different weights. It is assumed that the weights, if exist, would take the smallest possible values. For simplicity, we write $\sigma'(T_1,T_2)$ for weighted distance between T_1 and T_2 when the weights for individual transitions are known. The restriction on weights to be smallest leads to uniqueness of the distance.

Theorem 4

For an arbitrary place s in S in P/T-system Σ, as long as s has disjoint pre-set and post-set, $\sigma'(T_1,T_2) = \max\{ M(s) \mid M \in [M_0>\} - \min\{ M(s) \mid M \in [M_0>\}$ if exists, otherwise $\sigma'(T_1,T_2) = \infty$, where T_1 and T_2 are respectively the pre-set and post-set of s and $\sigma'(T_1,T_2)$ denotes a weighted distance with W(s,t) as the weight of transition t in T_2 and W(t,s) as the weight of transition t in T_1.◆

This theorem is true since what the added place records is exactly what happens in s.

In case there are no weights that yield a finite distance between T_1 and T_2, T_1 and T_2 are asynchronous.

4.3. Computing synchronous distance

All transition sequences can be found on the graph G(Σ). Synchronous distances, weighted or not, are defined in terms of transition sequences. Thus, it is possible to design algorithms for the computing of distances. Firstly, if a ring exists on G(Σ) that contains different numbers of firings of T_1 transitions and T_2 transitions, the distance is ∞. All algorithms are designed for computing finite distances only. The book in Chinese entitled *Petri Net Applications* by the author will soon appear, in which an algorithm for the computing of synchronous distances in P/T-systems can be found. It is easy to design an algorithm for computing weighted distances based on this algorithm.

We go no further on synchronous distances in this chapter since what concerns us is workflow modelling and what has been said about synchronous distances is already sufficient.

5. Business process

A successful application of Petri nets requires a full understanding of the application problem.

The term "workflow" was used as a synonym for "business process" in the book *Workflow Management* by W. Aalst and K. Hee. The terminology was developed by the Workflow Management Coalition (WFMC), an organization dedicated to develop standard terminology and standard interfaces for workflow management systems components.

As the first step towards workflow modelling, we make a clear distinction between "workflow" and "business process".

5.1. Business process vs. workflow

A business process is a pre-designed process in an enterprise or an organization for conducting the manipulation of individual cases of a business. It existed even before the birth of computers. The manipulation of individual business cases consists of business tasks and management tasks. The concept of "workflow" aims at "computerized facilitation or automation of a business process, in whole or part" (WFMC). To this end, the separation of business tasks and management tasks is of first importance. In a way, this is similar to the separation of data processing from a program, leading to the concepts of databases and database management systems.

A single business task may be carried out by a computer program. But "computerized automation of a business process" focuses on management automation rather than task automation. Management automation relies on clearly specified management rules. Most of the rules apply to all cases while some of the rules apply to individual cases. The former is "workflow logic" and the latter is "case semantics". The purpose of workflow modelling is nothing but to establish a formal specification of management rules to serve as a guide in the design, implementation and execution of a computer system called "workflow engine". It is the execution of the engine that conducts the processing of business cases. Figure 7 illustrates how workflow is related to business process.

The workflow logic specifies how business tasks are ordered (for causally dependent tasks) and/or synchronized. But, ordering is also a way of synchronization. Thus, workflow logic specifies how business tasks are synchronized for all conceivable cases in a business. In other words, workflow logic specifies all possible routes that a business case may take when being processed. What workflow logic cares about a single business task is not what data it requires or what data it will produce, let along what exact values are inputted or outputted. In this sense, workflow logic is concerned with abstract business tasks only.

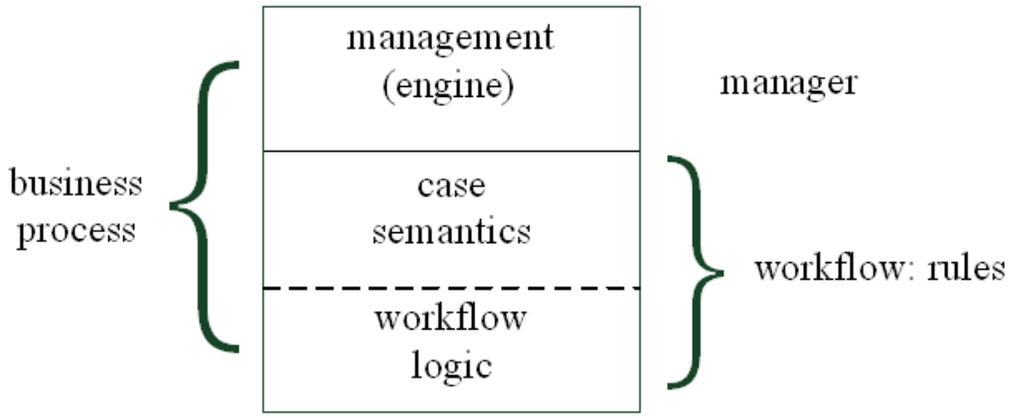


Figure 7. Business Process vs. Workflow

A selection is required at a crossing point of different routes. Concrete data of a given case determine the selection. The so selected route is the "semantics" of the given case. Thus, case semantics is concerned with those data that are used to decide whether a business task is on the route to be selected. These data are called explicit data since they should appear explicitly in the model of case semantics.

5.2. Management task vs. business task

A business must have a pre-designed form (paper form or electronic form) to record how an individual case is initiated and processed. Such a form is called a business form, B-form for short.

A business task can be characterized by

1. It requires professional knowledge, skill and/or experience related to the business.
2. It fills in at least one blank assigned to it on the B-form to tell what has been done.

A management task, on the other hand, requires knowledge on management rules. If it fills in blanks on the B-form at all, the written content is for management need only, e.g. a serial number for sorting and searching B-forms later on.

In the book *Workflow Management*, 16 tasks have been listed for "insurance claim" business in a (fictional) insurance company, among them, the first two tasks are:

1. Recording the receipt of the claim;
2. Establishing the type of claim (for example, fire, motor vehicle, travel, professional).

The first task may assign a serial number to the claim. Otherwise, it would leave no trace on a B-form. In fact, it is even not concerned with what B-form to be used. The second task must fill in the blank "type of claim" at least. Thus, the first task is a management task (may be shared by all businesses in a company) while the second task is a business task. In other words, the first task should be excluded from the workflow logic for "insurance claim", but the second task must be included.

5.3. Iteration vs. no iteration

The author of *Workflow Management* wrote: "This example of a process (i.e. insurance claim) also includes iteration or repetition — namely, the repeated assessment of an objection or the revision of the amount to be paid. In theory, this could go on forever."

As a management rule, an objection to a claim may need to be assessed more than once, but it must be a fixed number of times and by different persons; furthermore, the conclusion of each assessment must be recorded on the B-form. In other words, reassessment is a different task rather than a repetition of assessment.

Mathematically or logically, repetition could go on forever, but not for a theory of management. Endless repetition is not even thinkable in a management theory.

Iteration or repetition implies "redo" of the same task, i.e. what has been written on the B-form should be erased. "Redo" caused by wrong doings could, in theory, occur for every single task since human beings are erroneous by nature. No one would suggest a repetition for every task in a workflow model. "Redo" is nothing but a management measure to heal wrong doings, not a normal portion in a business process. Besides, workflow logic is about abstract tasks, but iteration is case dependent.

Endless redo could not occur since it is under control of the engine.

The execution of a single business task may include iteration. For example, when the amount of goods exceeds the truck load, the truck(s) would have to return after unloading. Such iteration is the detail of task execution, not a measure of management. Besides, it is case dependent. Iteration has nothing to do with workflow logic. What case semantics is concerned with is whether a task is on the route, not how a task is executed.

5.4. Set of tasks vs. set of blanks on B-form

Let $T = \{t_1, t_2, \cdots, t_n\}$ and $B = \{b_1, b_2, \cdots, b_m\}$ be the set of business tasks and respectively the set of blanks on the B-form for a give business process. The investigation here aims at properties of T and B for a well-defined business process.

There must be a correspondence between T and B, telling which blank(s) each task is responsible for. This correspondence should guarantee that no more than one task is responsible for the same blank in the course of processing a single case, and no blank is left empty upon termination of the case processing (those blanks that are not in relation with a given case are assumed to have been crossed out by the engine). Such correspondence has become common sense today. We will say no more about it.

Causal dependence among tasks in T leads to a combinatorial acyclic partial order on $T: < \subseteq T \times T$, and a sub-relation $<\cdot$ of $<$: $x <\cdot y \equiv x<y \wedge \forall z \in T: \neg(x<z \wedge z<y)$. $<\cdot$ is the "immediate successor" relation: y is the immediate successor of x. Since the ordering relation $<$ is derivable from $<\cdot$ by transitivity of $<$, we will talk about $(T, <\cdot)$ instead of $(T, <)$. No iteration among tasks in T leads to acyclic partial order.

Let u be a task in T, u is a start task if it is before every other task, i.e. $\forall t \in T: \neg(t <\cdot u)$; u is an end task if u has no immediate successor. It is assumed that there is a unique start task, i.e. all cases share a unique beginning to recognize and to accept a case.

Let T_1, T_2 be nonempty subsets of T. T_2 is called an immediate successor set of T_1 , denoted by $T_1 <\cdot T_2$, if $\forall u \in T1 \forall v \in T_2 : u <\cdot v$. It is easy to prove that T_1 and T_2 are disjoint, and tasks in T_1 (T_2) are not ordered. Synchronization for management need can only be introduced between T_1 and T_2. Here, the synchronization between T_1 and T_2 must be place synchronization since they are different transitions. In case T_1 and T_2 are both singletons, the synchronization can only be the immediate successor relation. As an example, let be T_1 = {a, b} and T_2 = {c, d}, $\sigma(T_1,T_2)=2$ and $\sigma(T_1,2T_2)=2$ are two different ways of synchronization. Taking into account the fact that every task, if it is included on a route, can only be (effectively, redo is not counted) executed once, $\sigma(T_1,T_2)=2$ requires that tasks a ,b being executed in parallel followed by tasks c, d being executed in parallel; $\sigma(T_1,2T_2)=2$ requires that tasks a, b being executed in parallel followed by either task c or task d being executed alone. Data from a given case will be used for the selection between task c and task d. It is clear that synchronous distances would be of no use if redo is taken into account.

Now the processing of a given case goes like this: it begins from the unique start task, each task on the route carries out its duty and fills in blank(s) it is responsible for on the B-form, and passes it to its immediate successor(s) via the engine. The engine takes care of quality checking, successor selection and appointing executer(s) for selected task(s); and finally, the engine passes the B-form to appointed executer(s).

6. Synchronizer and workflow logic

P/T-systems will be used for the modelling of workflow logic. A synchronizer is a special place to connect transitions (tasks) with immediate successor relation ($<\cdot$).

To capture the fact that every task (transition) is executed (effectively) at most once for a single business case and workflow logic contains no iteration, a concept of restricted P/T-systems, RP/T-systems for short, is proposed.

Definition 9

A P/T-system $\Sigma=(S,T;F,K,W,M_0)$ is a RP/T-system if it is acyclic ($F^+ \cap (F^{-1})^+ = \emptyset$) and any transition is restricted to fire at most once. ◆

The P/T-system in Figure 2 is a good example: as a normal P/T-system, transition t_2 can fire twice when p_1 has 2 tokens; but as a RP/T-system, t_2 can fire only once at the same marking. In fact, transitions t_2 and t_3 would fire when p_1 has 2 tokens for a RP/T-system, though t2 remains enabled after its first firing.

Note that RP/T-systems are not a new class of net systems. If we introduce a control place c for every transition t such that c has an empty pre-set and {t} is its post-set, and initially it has one token. Apparently, with an added control place for every transition, every transition can fire at most once by conventional transition rules.

6.1. Synchronizer

Definition 10

A place p in a RP/T-system is a synchronizer if T_1 and T_2 are the pre-set and post-set of p and they are not empty; and $\forall t \in T_1$: W(t, p) = b and $\forall t \in T_2$: W(p, t) = a where b and a are integers such that $0 \le a \le n$ and $0 \le b \le m$; n, m are respectively the numbers of transitions in T_1 and T_2; and K(p) = ab.◆

The RP/T-system in Figure 2 has two synchronizers p_1 and p_2 since $K(p_1) = 2$ and $K(p_2) = 2$.

The synchronizer defined by Definition 10 is usually denoted by p = (T_1, T_2, (a, b)) or p = ($^{\cdot}$p, p$^{\cdot}$, (a, b)). Figure 8 (a) is the graphical presentation of a synchronizer p and (b) is the detail of p, explaining how synchronization is achieved. Theorem 4 concludes that $\sigma(bT_1, aT_2) = ab$, i.e. the weighted synchronous distance between T_1 and T_2 are ab where b is the weight for all transitions in T_1 and a is the weight for all transitions in T_2.

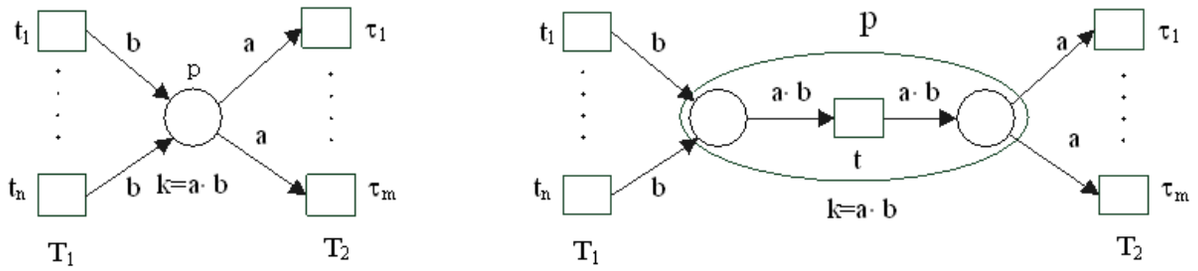


Figure 8. A Synchronizer and its Detail

Definition 11

Let p = (T_1, T_2, (a, b)) be a synchronizer and n, m be the numbers of transitions in T_1 and T_2. P can be classified as below:

n=m=1, p is sequential synchronizer,
m>1, p is a split,
n>1, p is a join,
a>1 and a=n, p is an ALL-join,
1<a<n, p is an OR-join,
a=1 and a<n, p is a XOR-join,
b>1 and b=n, p is an ALL-split,
1<b<m, p is an OR-split,
b=1 and b<m, p is a XOR-split.

Here we have deliberately chosen terms used by Prof. Aalst for WF-net. The difference is, our synchronizers are places while synchronization is achieved via transitions in WF-net. Synchrony in Section 3 has made it clear that synchronization with a distance greater than zero is place synchronization. We will see what advantages the concept of synchronizers brings with it when we approach to management logic.

Figure 9 shows how to achieve mixed synchronization where p_1 is an XOR-split and p_2 is an ALL-split. What follows task t would be one of t_1 and t_2 plus both t_3 and t_4.

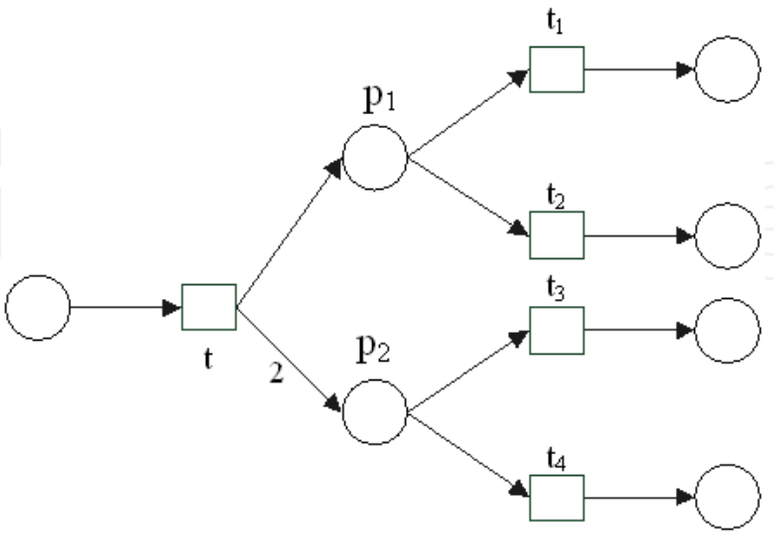


Figure 9. ALL-split and XOR-split mixture

Definition 12

Let x, y be variables of integer type. The place $p_1 = (T_1, T_2, (a, y))$, $p_2 = (T_1, T_2, (x, b))$ and $p_3 = (T_1, T_2, (x, y))$ are variations of synchronizer $p = (T_1, T_2, (a, b))$ as long as $0 < x \leq n$ and $0 < y \leq m$. p_1, p_2 and p3 are flexible synchronizers, since x and y can take different values.◆

Figure 10 explains how flexible synchronizers work: x may take 1 or 2 as its value to achieve different synchronization.

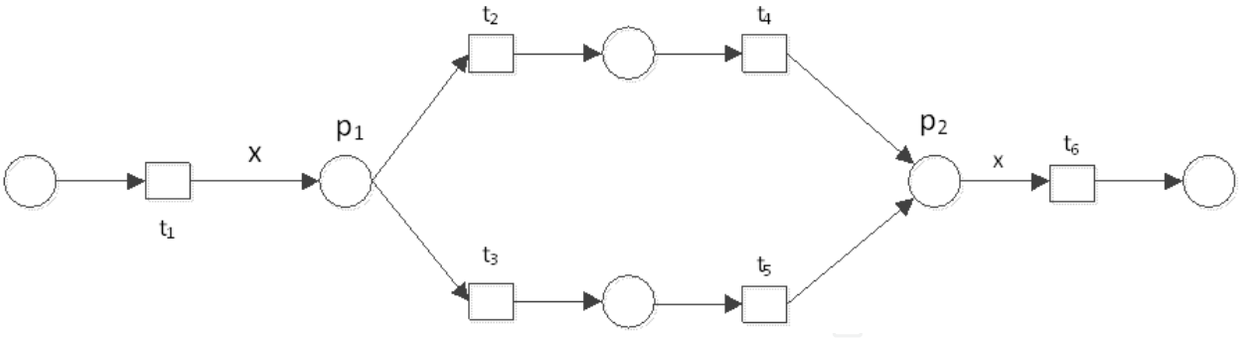


Figure 10. Flexible Synchronizers p_1 and p_2

We have not found a way to express flexible synchronization in terms of WF-net since an OR-split or OR-join transition in WF-net does not tell the exact number "OR" represents.

6.2. Workflow logic

RP/T-systems are candidates for modelling workflow logic, but not every RP/T-system is suitable.

Definition 13

For RP/T-system $\Sigma = (S,T;F,K,W,M_0)$, relation next$\subseteq T\times T$ is given by next = $\{(t, t') \mid t^{\cdot} \cap {}^{\cdot}t' \neq \emptyset\}$. We write next(t, t') for (t, t') ∈ next. Σ is well ordered if, for any t, t' in T, next(t, t') implies $\forall u \in T: \neg(\text{next}(t,u) \wedge \text{next}(u, t'))$.

The system in Figure 11 is not well-ordered, since we have next(t_1, t_3) and next(t_1, t_2)∧ next(t_2, t_3).

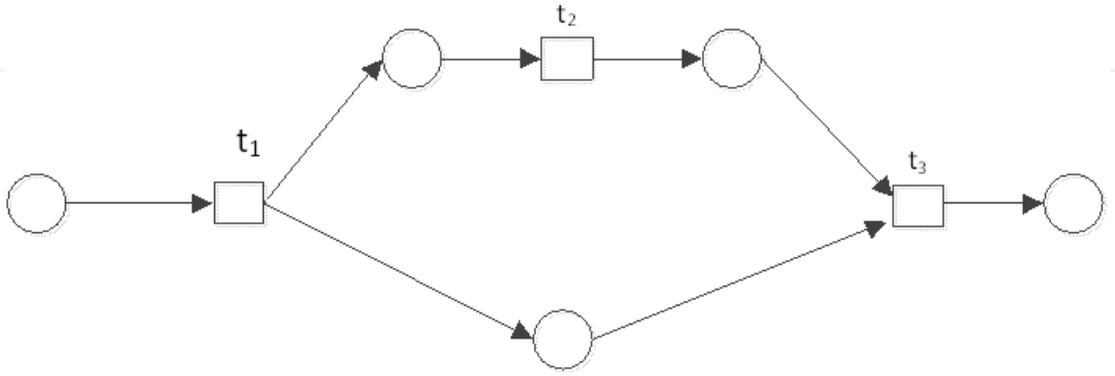


Figure 11. A System not Well-ordered

Remember that we are seeking for net systems that are suitable for modelling workflow logic. If t_1, t_2 and t_3 are tasks, then the B-form would be passed from t_1 to t_3 twice: one directly and one via t_2. This redundancy leads to structure complexity and analytical difficulty. Besides, there might be semantic inconsistency for this or that reason.

Definition 14

A well-ordered RP/T-system $\Sigma = (S,T;F,K,W,M_0)$ is a workflow logic if :

Every transition has nonempty pre- and post- sets; there is a unique place s_0 such that it has an empty pre-set and its post-set is a singleton $\{t_0\}$ and $K(s_0) = 1$, $W(s_0, t_0) = 1$; every intermediate place p is a synchronizer; every end-place, i.e. a place with an empty post-set, has a capacity 1 and its input arc(s) has 1 as its weight; The initial marking has a unique token in s_0.◆

A task is initiated upon receiving a B-form, and passes it over to its successor upon termination. Thus, its pre-set and post-set are nonempty. The unique place s_0 is the place to receive initiated B-forms one by one and one at a time. Transition t_0 is the unique start task. An end-place is to receive completed B-forms, one by one and one at a time. All tasks must be well synchronized, so intermediate places are synchronizers. The weight on an input arc (t, p) of synchronizer p represents that W(t, p) immediate successors need the B-form completed by t, and the weight on an output arc of p represents that B-forms completed by W(p, t) different immediate predecessors of t are needed by t. The unique token in s_0 represents an abstract single case. It is an abstract case since no data is needed. An abstract case resembles all conceivable cases.

The system in Figure 2 is workflow logic. The conventional transition rules would apply to a workflow logic if a control place is added to each of the transitions (so, no transition can fire for the second time for one case) and the detail of every synchronizer as given in Figure 8 is also made explicit. But, this would make workflow logic too complicated for practical use. The following is the transition rules for workflow logic.

Definition 15

Let $\Sigma = (S,T;F,K,W, M_0)$ be a workflow logic, t in T is a transition and M is a marking (i.e. the initial marking M_0 or reachable from M_0 by rules given here).

1. t is enabled by M (or at M), denoted by $M[t\gg$, if $\forall s \in {}^{\cdot}t: M(s)=K(s)$,
2. Once $M[t\gg$ is true, it remains true until either t occurs or t is not enabled according to conventional transition rule.
3. The successor marking, when t fires, is computed according to the conventional rule.◆

It is recommendable to compute the set of reachable markings for the workflow logic given in Figure 2. We denote with $[M_0 \gg$ the set of reachable markings.

Redundancy is always closely connected to complexity. The property of being well-ordered has removed certain redundancy. The next definition reveals different redundancy.

Definition 16

A workflow logic is well-structured if for every synchronizer p and a reachable marking M, M(p) = K(p) implies the fact that either all transitions in the post-set of p are enabled, or none of them is enabled.◆

Figure 12 illustrates this definition: the workflow logic in (a) is not well- structured, since t_3 is enabled but t_4 is not when p_2 has a token (the capacity of a synchronizer is known without saying). On the other hand, transitions t_3, t_4 and t_5 are all enabled when p_2 has 2 tokens in the system in (b).

Careful readers may have noticed that p_2 in (b) is not a synchronizer. No, it is not. It is a variation of a synchronizer. The two transitions t_3 and t_5 in (b) are viewed as twins: either both of them to be on a route, or none of them to be selected since t_4 requires 2 tokens. Thus, as far as route selection is concerned (workflow logic aims at the description of all possible routes), twin transitions could and should be combined as shown in Figure 13. Variations of synchronizers may be allowed in practice for convenience. But for the analysis of workflow logic, combined twins are preferred.

If the system in Figure 12 (b) is taken as workflow logic with a synchronizer variation, it reveals another problem: there would be two tokens upon the completion of one case when t_3 and t_5 are on its route. This is inconsistent with common knowledge: A single case cannot have two separated conclusions. This observation raises a question: what properties a well-defined workflow logic should have, in addition to being well-ordered and well-structured?

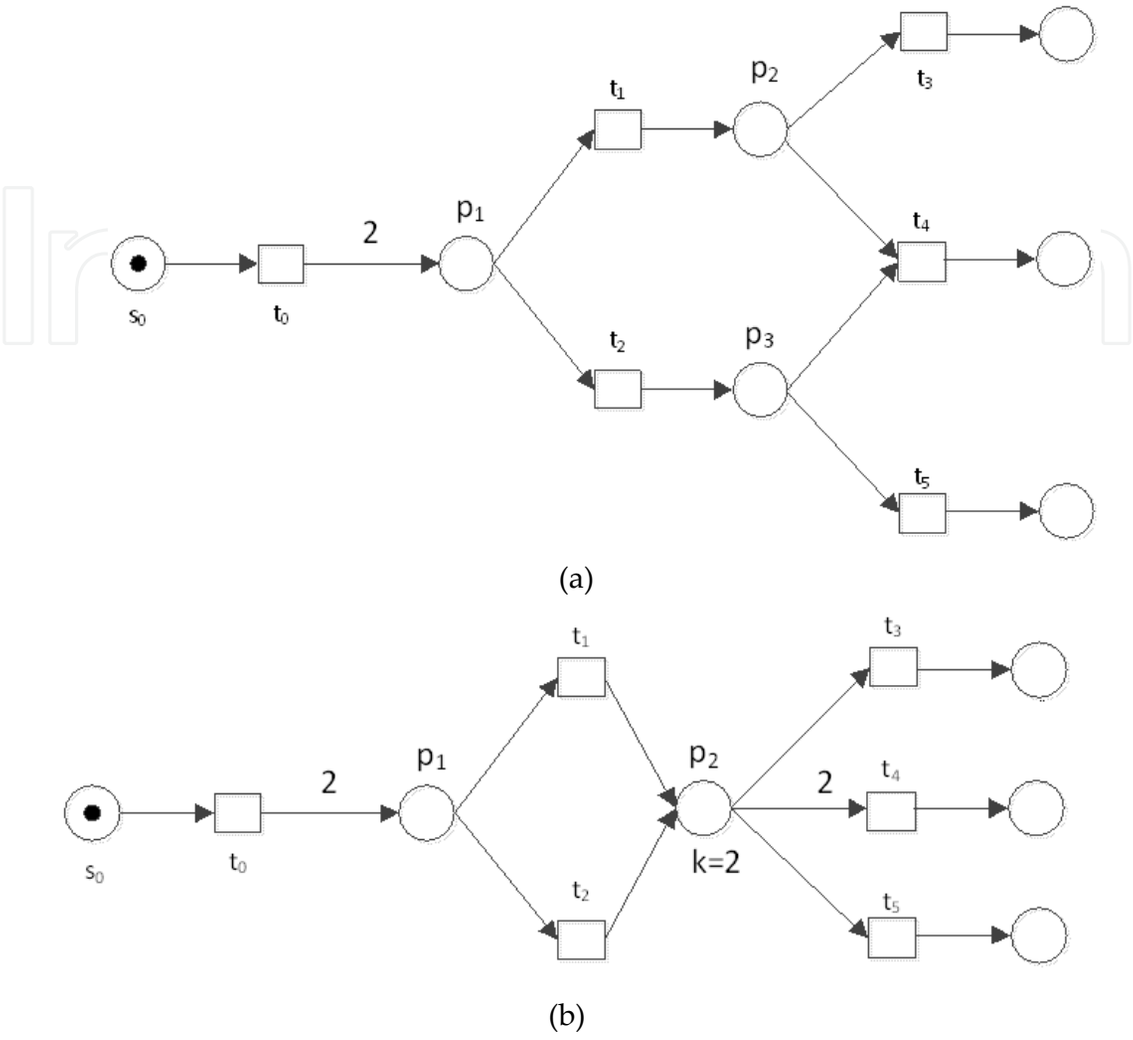


Figure 12. Not well- structured workflow logic

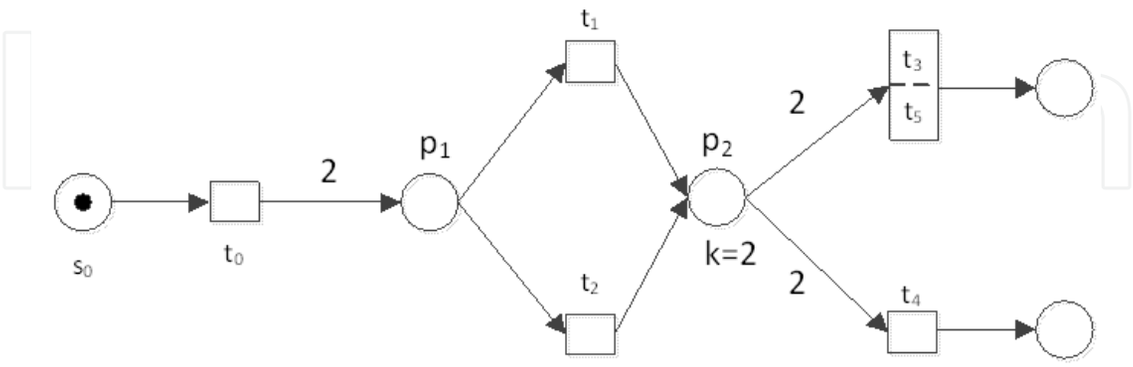


Figure 13. Combined Twins

Soundness! This is the answer from WF-net. Soundness requires:

"A process contains no unnecessary tasks and every case submitted to the process must be completed in full and with no references to it (that is case tokens) remaining in the process."

What we propose is throughness: the token in s_0 (an abstract case) will be passed over by transition firings (tasks) via the engine, to a unique end-place. All properties required by soundness are covered by being through.

In what follows, workflow logic is always assumed well-structured.

6.3. Throughness and reduction rules

Let $\Sigma = (S,T; F, K, W, M_0)$ be a well-structured workflow logic, and M is a reachable marking.

Definition 17

1. M is a termination marking if it enables no transition; M is an end-marking if there is a unique end-place e such that M(e) = 1 and M(s) = 0 for all places s other than e.
2. Σ is through if every termination marking is an end-marking and every end-marking is also a termination marking. ◆

The workflow logic in Figure 2 is through.

Theorem 5

If workflow logic Σ is through, then for every transition t, there is a reachable marking M that enables t and there is a route to which t belongs.◆

From the fact that Σ is acyclic and s_0 is unique, we know that there is a directed path from s_0 to t. By mathematical reduction on the length of this path, this theorem is easy to prove.

Soundness of a WF-net is proved via computing T-invariant by adding an extra transition between its unique place o and its unique initial place i. This method applies to workflow logic as well. But we do it differently.

Keep in mind the principle of local determinism. Our attention focuses on local structures of workflow logic rather than global properties like T-invariants.

Characteristics of local structures bring up the following reduction rules for proving throughness of workflow logic.

Reduction Rule 1

Synchronizer $p_1 = (T_1,T_0,(a_1, a_2))$ and $p_2 = (T_0,T_2,(b_1,b_2))$ can be reduced to synchronizer $p = (T_1,T_2,(a_1,b_2))$ if $a_2 = b_1$. ◆

Figure 14 illustrates this rule.

Let Σ be the workflow logic to which rule 1 is applied and Σ′ is resulted by replacing p_1, T_0, p_2 with p in Σ. We have

Theorem 6

Σ is through if and only if Σ′ is through.◆

Synchronizer p_1 requires that a_2 transitions from T_0 to fire after the firings of a_1 transitions from T_1 while synchronizer p_2 requires that b_1 transitions from T_0 to be fired before any transition firing from T_2. Given $a_2 = b_1$, p_1 and p_2 are consistent with each other on T_0: a_1 transitions from T_1 followed by b_2 transitions from T_2. Thus, the theorem is true. (p_1,T_0, p_2) is in fact the detail of p. This reduction of omitting consistent detail is a net morphism in net topology. No wonder the property of being through is reserved. All reduction rules are in fact to conceal local details, and as such, they reserve throughness. But there will be no more theorems and proofs to be given below for simplicity.

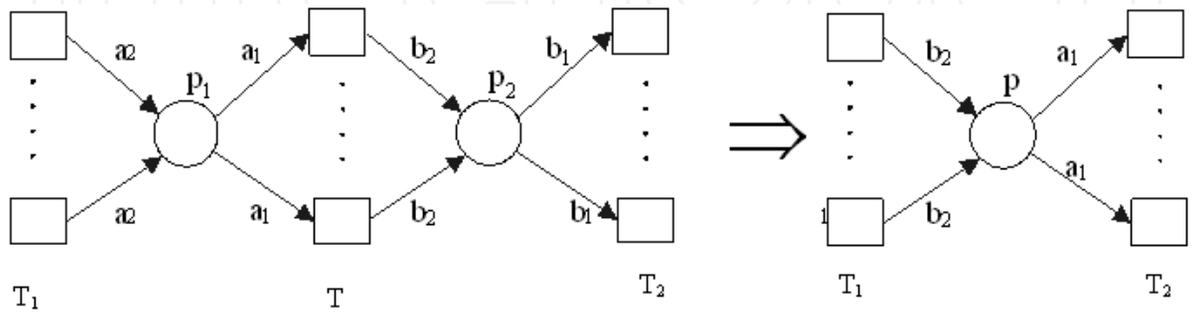


Figure 14. Reduction Rule 1

Reduction Rule 2

For synchronizer p = ({t_1}, {t_2}, (1,1)), if $t_1 \cdot \cap \cdot t_2 = \{p\}$, then ($t_1$, p, t_2) can be replaced by a single transition t with $\cdot t = \cdot t_1 \cup \cdot t_2 - \{p\}$ and $t \cdot = t_1 \cdot \cup t_2 \cdot - \{p\}$. All weights on remaining arcs remain.◆

Figure 15 illustrates this rule.

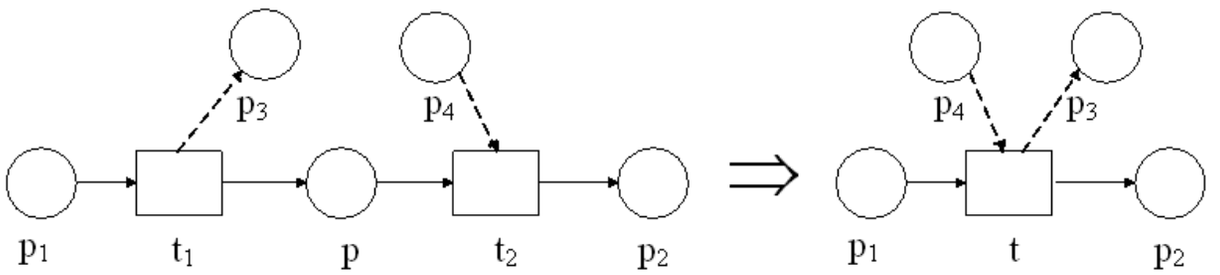


Figure 15. Reduction Rule 2

It is easy to see that the combined effect of t_1 and t_2 on all places other than p is the same as the effect of t. The dotted circles and arcs may exist, but not necessarily.

Reduction Rule 3

For transition t and places p_1, p_2, if $p_1 \cdot \cap \cdot p_2 = \{t\}$ and $W(p_1, t) = W(t, p_2) = 1$, then ($p_1$,t,$p_2$) can be replaced by p, $\cdot p = \cdot p_1 \cup \cdot p_2 - \{t\}$ and $p \cdot = p_1 \cdot \cup p_2 \cdot - \{t\}$. All weights on remaining arcs remain. ◆

Note that places p_1 and p_2 are not necessarily synchronizers, i.e. it is possible $\cdot p_1 = \emptyset$ and/or $p_2 \cdot = \emptyset$.

Figure 16 (a) illustrates this rule.

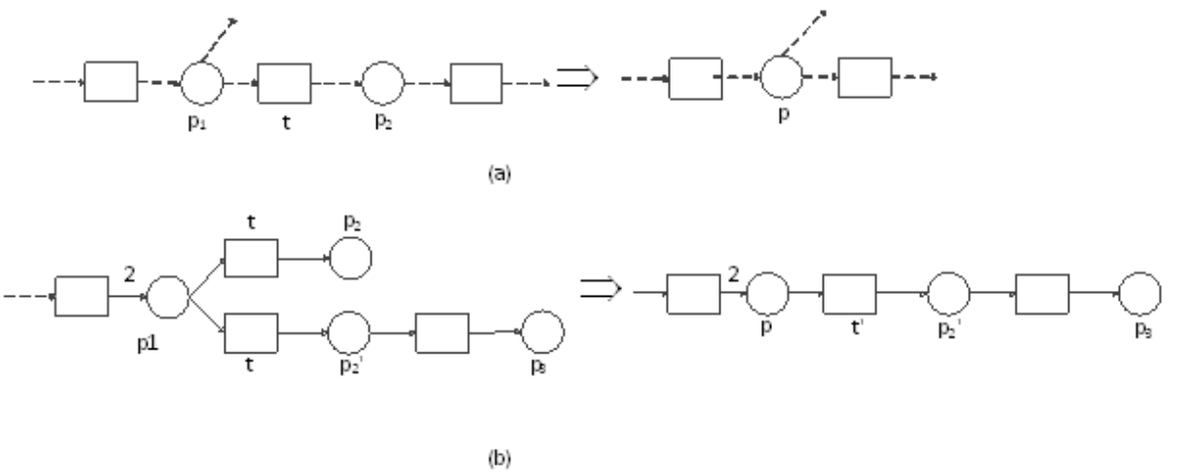


Figure 16. Reduction Rule 3

The workflow logic in Figure 16 (b) is not through since the termination marking would have two tokens at two different end-places. Rule 3 can be applied to reduce it in size, but the resulted place p turns out to be inconsistent between its pre-set and its post-set.

Reduction Rule 4

For $T_1 = \{t_1, t_2, \cdots, t_a\}$ and $T_2 = \{t_1', t_2', \cdots, t_b'\}$, if $p_i = (\{t_i\}, T_2, (1, b))$ is a synchronizer, i = 1, 2,⋯, a, then p_1, p_2,⋯ pa can be reduced to a single synchronizer $p = (T_1, T_2, (a, b))$; If $p_i' = (T_1, \{t_i'\}, (a,1))$ is a synchronizer i = 1,2,⋯, b, then p_1', p_2',⋯p_b' can be reduced to the same single synchronizer $p = (T_1, T_2, (a, b))$.◆

Figure 17 illustrates this rule where a = 3 and b = 2.

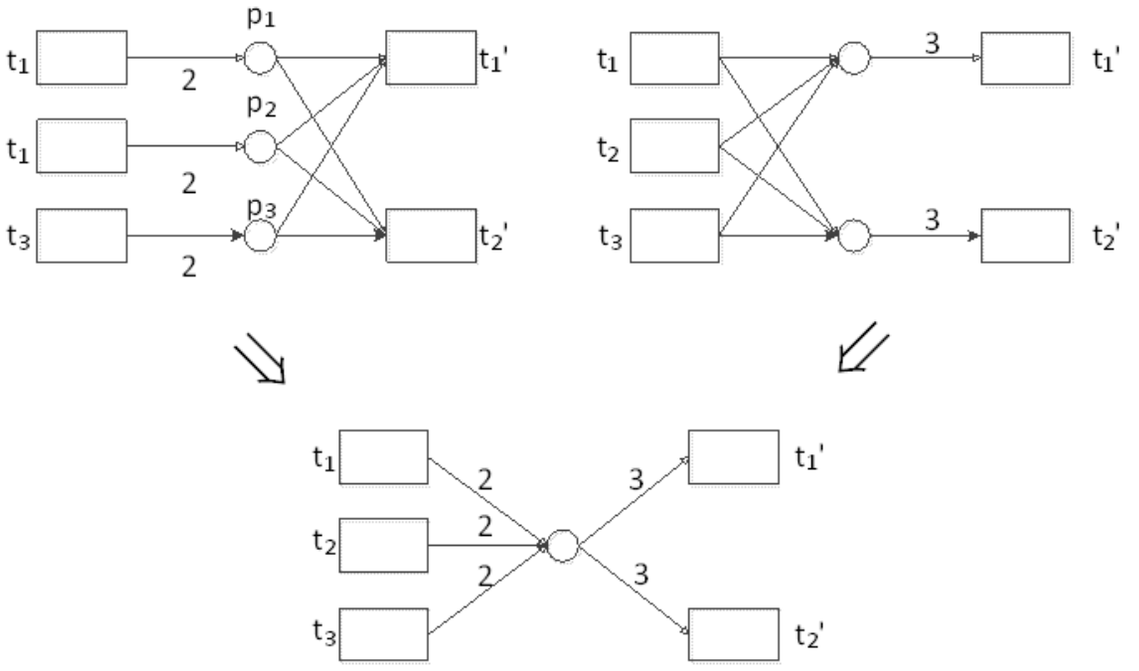


Figure 17. Redaction Rule 4

We will see the significance of this rule and the rule next when management logic is discussed, since separated managements before the rule is applied become centralized management after it.

Reduction Rule 5

If $p_i = (\{t\}, \{t_i\}, (1, 1))$ is a synchronizer for every i, i = 1, 2, ⋯, a, then these synchronizers can be reduced to a single synchronizer $p = (\{t\}, \{t_1, t_2, \cdots, t_a\}, (1, a))$.

If $p_j = (\{t_j\}, \{t\}, (1, 1))$ is a synchronizer for every j, j = 1, 2, ⋯, b, then these synchronizers can be reduced to a single synchronizer $p = (\{t_1, t_2, \cdots, t_b\}, \{t\}, (b, 1))$.◆

Figure 18 illustrates this rule, where a = b = 3. .

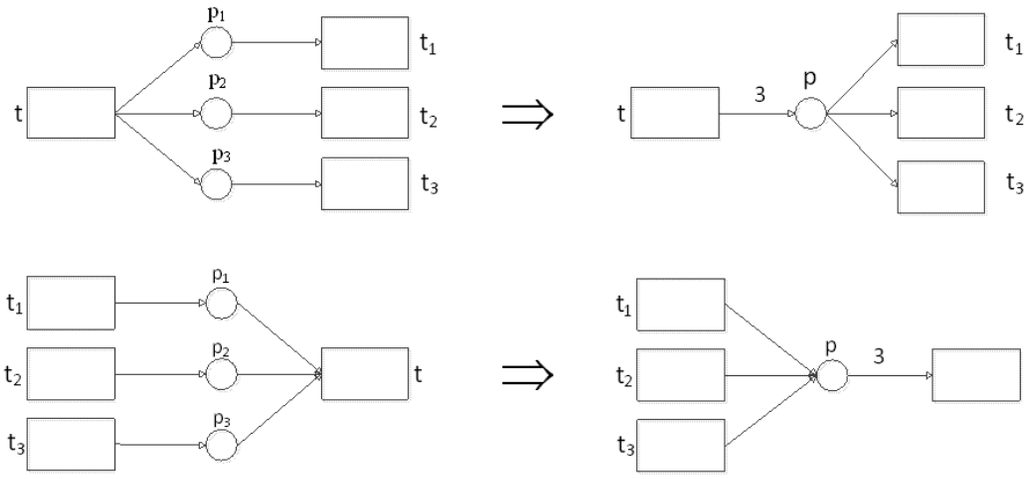


Figure 18. Reduction Rule 5

The next reduction rule reserves throughness, but a place element rather than a synchronizer is used for reduction.

Reduction Rule 6

In this rule, i = 1, 2, ⋯, a and j = 1, 2, ⋯, b. For transition sets $T = \{t_1, t_2, \cdots, t_a\}$ and $T_i = \{t_{i1}, t_{i2}, \cdots, t_{ib}\}$, if $p_i = (\{t_i\}, T_i, (1, b))$ is a synchronizer for every i, then these synchronizers can be reduced to the place p: $^{\cdot}p = T$, $p^{\cdot} = \bigcup_{i=1}^{a} T_i$, K(p) = ab, $W(t_i, p) = b$ and $W(p, t_{ij}) = 1$, If $p_i' = (T_i, \{t_i\}, (b, 1))$ is a synchronizer for every i, then these synchronizers can be reduced to the place q: $^{\cdot}q = \bigcup_{i=1}^{a} T_i$, $q^{\cdot} = T$, K(q) = ab, $W(t_{ij}, q) = 1$, $W(q, t_i) = b$.◆

Figure 19 illustrates this rule, where a = 3 and b = 2.

A noticeable fact is: places p and q are different from a synchronizer at two aspects. Firstly, their capacities are both ab instead of the product of the input weight and the output weight. Secondly, transitions in T_i and T_j , i ≠ j, maybe enabled at different times before reduction, but they will be enabled at the same time by p; similarly, transitions in T will be enabled at the same time by q, but maybe not before reduction. The important point is, the number of transitions to be selected for a route remain unchanged (real selection is to be

determined by concrete data from a practical case later by case semantics.) due to the fact that all synchronizers in question are either ALL-split or ALL-join. This means that the property of being through is reserved.

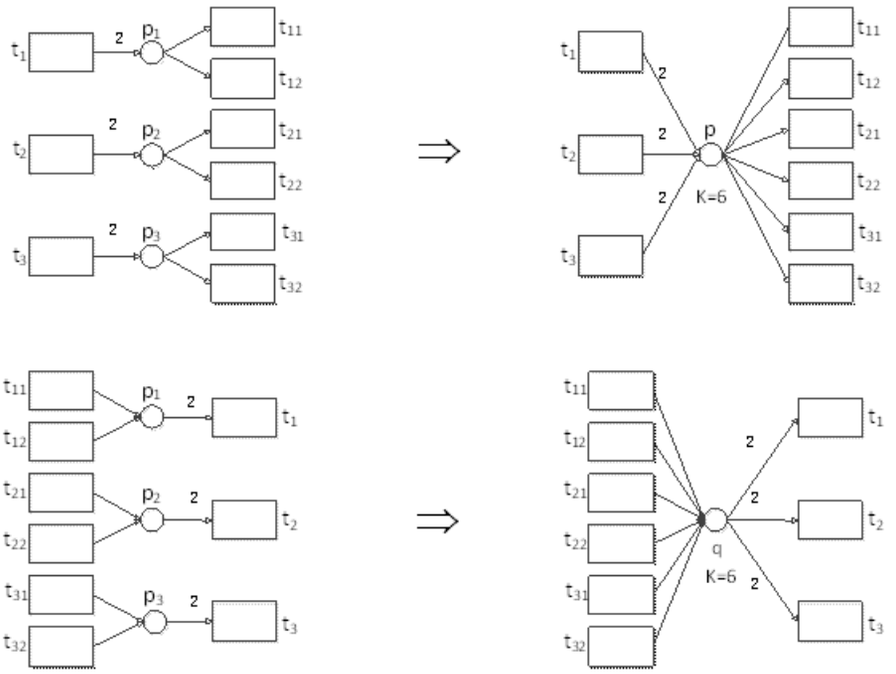


Figure 19. Reduction Rule 6

Definition 18

A place p with ${}^{\cdot}p = T_1$ and $p^{\cdot} = T_2$, $T_1 \neq \emptyset$, $T_2 \neq \emptyset$, is called a virtual synchronizer if there exist integers a and b such that $\forall t \in T_1: W(t,p)=b, \forall t \in T_2: W(p, t)=a$, $K(p) \neq ab$, but $K(p)/b = n$, $K(p)/a = m$, where n, m are respectively the numbers of transitions in T_1 and T_2. For virtual synchronizer p, we also write $p = ({}^{\cdot}p, p^{\cdot}, (a,b))$.◆

The two places p and q in Figure 19 are virtual synchronizers. It is easy to check that the two places used for reduction by Rule 6 are virtual synchronizers in general.

For further reduction when a virtual synchronizer is involved, we have the revised version of Reduction Tule 1:

Reduction Rule 1':

Let each of $p_1 = (T_1, T_0, (a_1, a_2))$ and $p_2 = (T_0, T_2, (b_1, b_2))$ be a synchronizer or a virtual synchronizer. As long as $K(p_1)/a_1 = K(p_2)/b_2$, (p_1, T_0, p_2) can be reduced to $p = (T_1, T_2, (a, b))$, where $a = K(p_1)/b_1$ and $b = K(p_2)/a_2$, and $K(p) = ab$, $\forall t \in T_2$: $W(t, p) = b$ and $\forall t \in T_2$: $W(t, p) = a$. ◆

This rule is applicable when the involved virtual synchronizer is of ALL-join and ALL-split nature. Otherwise, local consistence should be checked since a virtual synchronizer is more flexible than a synchronizer. Figure 20 explains: the workflow logic on top is not through since just one of p_2 and p_3 may have 2 tokens while p_4 requires the two of them to have two

tokens each. As shown in the figure, a virtual synchronizer p_5 is obtained when rule 6 is applied. The resulted system after rule 6 remains being not through. If rule 1′ is further applied to p_5 and p_4, we get a system that is through. Inconsistence between p_1 and p_4 is concealed. The virtual synchronizer p_5 does not tell whether it is ALL-split or OR-split. It inherits property of being ALL-split or OR-split from p_1.

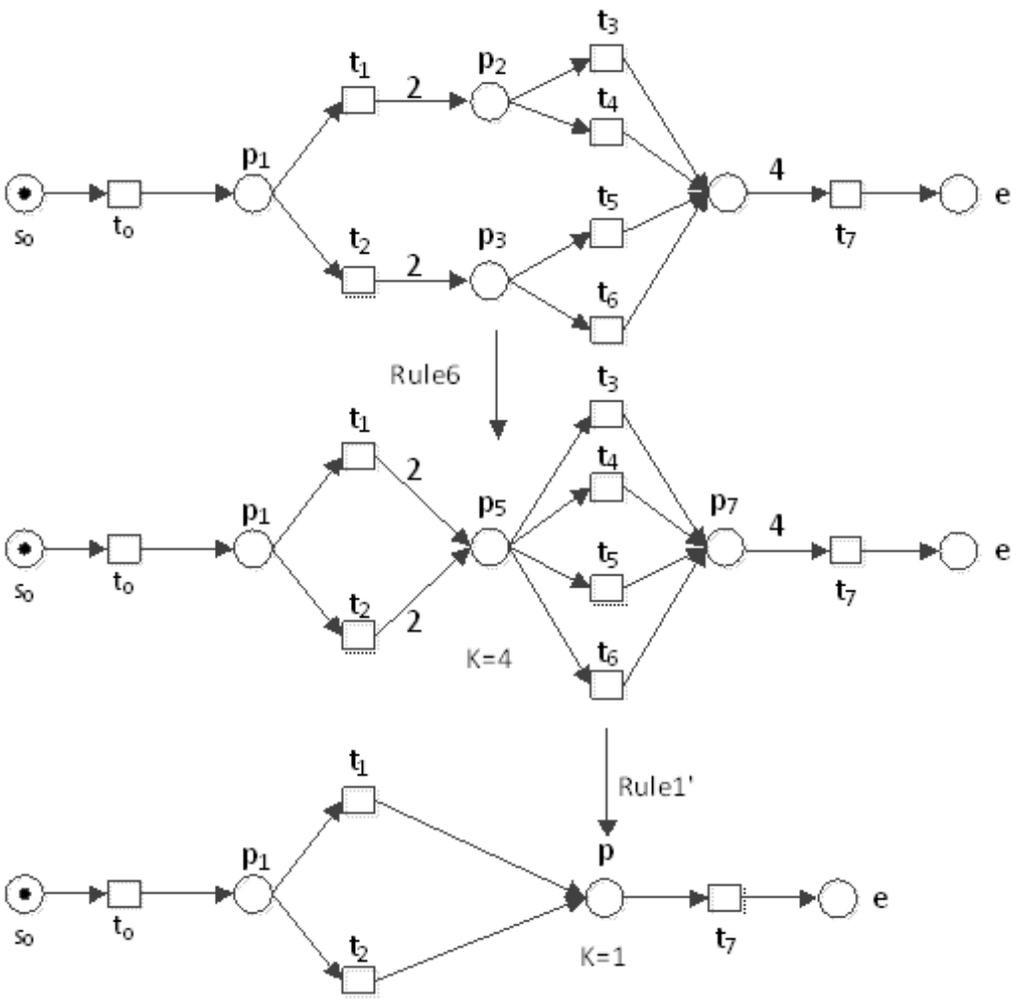


Figure 20. Reduction Rule 1′ leads to Error

Figure 21 illustrates Reduction Rule 1′, in which the workflow logic on top is apparently through. After twice applications of rule 6, the resulted p_6 and p_7 are virtual synchronizers both. By reduction rule 1′, these two virtual synchronizers are replaced by p, which is a synchronizer. This time the property of being through is reserved.

Reduction Rule 6 is suggested not to be used as long as there is a different reduction rule applicable.

Theorem 7

A well-structured workflow logic has the property of being through if it can be reduced to a single isolated place, i.e. a place whose pre-set and post-set are both empty.◆

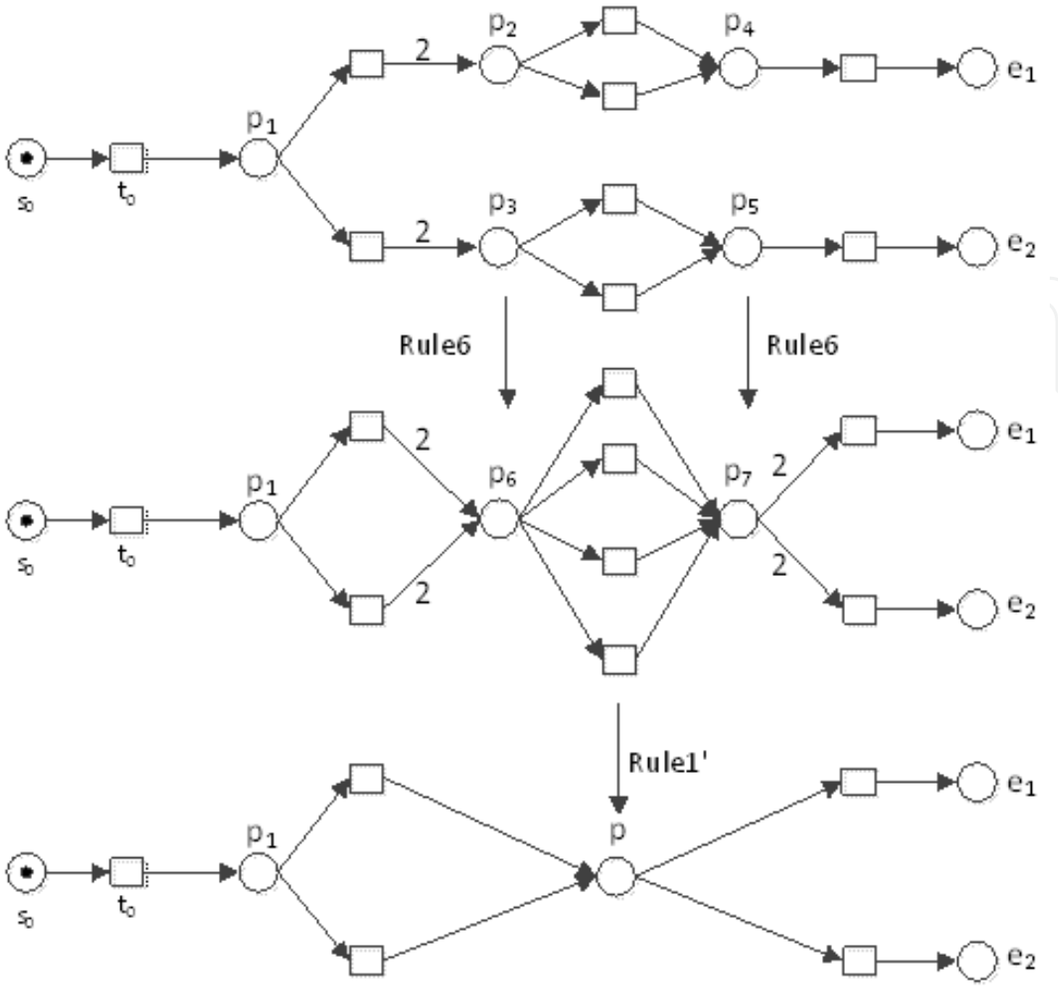


Figure 21. Reduction Rule 1′

This theorem is true since all reduction rules (including carefully used rule 1') reserve throughness . The isolated place is the start-place and the end place at the same time, thus the concealed detail has the property of being through.

This isolated place represents "harmony" as far as workflow logic is concerned. An interesting fact is, the isolated transition represents "contradiction" when it is resulted by applying the Resolution Rule and the Expansion Rule in Enlogy (a branch in GNT, net logic) for logical reasoning. A pair of dual elements, i.e. a place and a transition, lead to opposite concepts in philosophy, could this happen by chance? In fact as we will see soon, the dual of workflow logic is exactly the logic for workflow management.

We do not claim the completeness of this set of reduction rules. Whenever a user finds workflow logic not reducible to "harmony" though it is indeed through, please try to figure out new reduction rules and to let us know. Reduction rules would be enriched this way, we hope.

Figure 22 shows how to reduce to harmony the workflow logic for insurance claim where t_0, t_1, t_2, t_3 and t_4 are respectively the tasks "accept", "check policy", "check claim", "send letter" and "pay" (See *Workflow Management*).

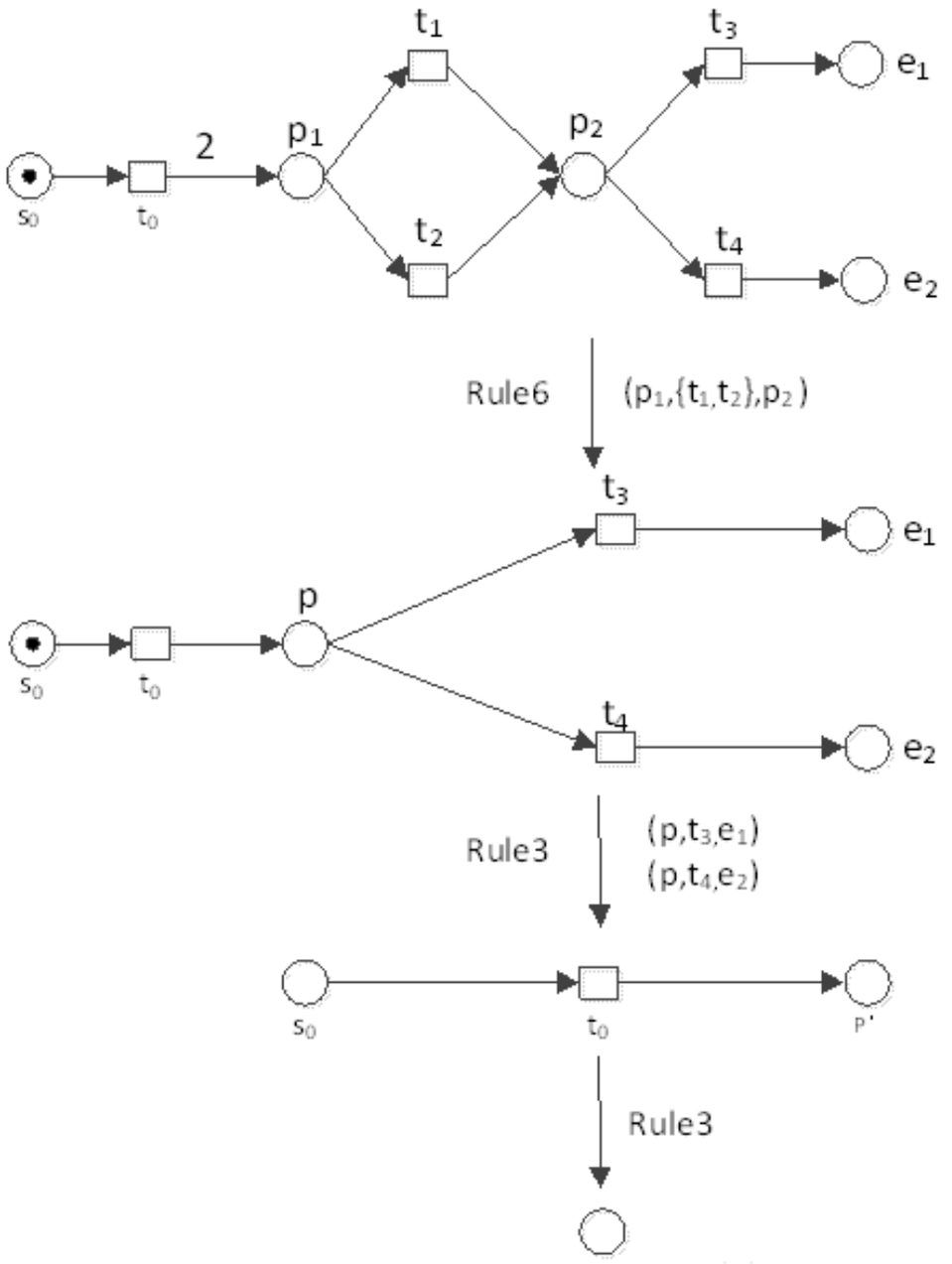


Figure 22. Reducing Workflow to Harmony

Figure 23 is a more complicated net from Dr. Aalst during his visit to Tsinghua University (Beijing, China) in 2004, where (a) is the original net in which every place has the capacity 1 and all weights on arcs are also 1, i.e. a WF-net. The author would suggest to take (b) as the workflow logic for whatever business in mind. Note that (c) contains virtual synchronizers.

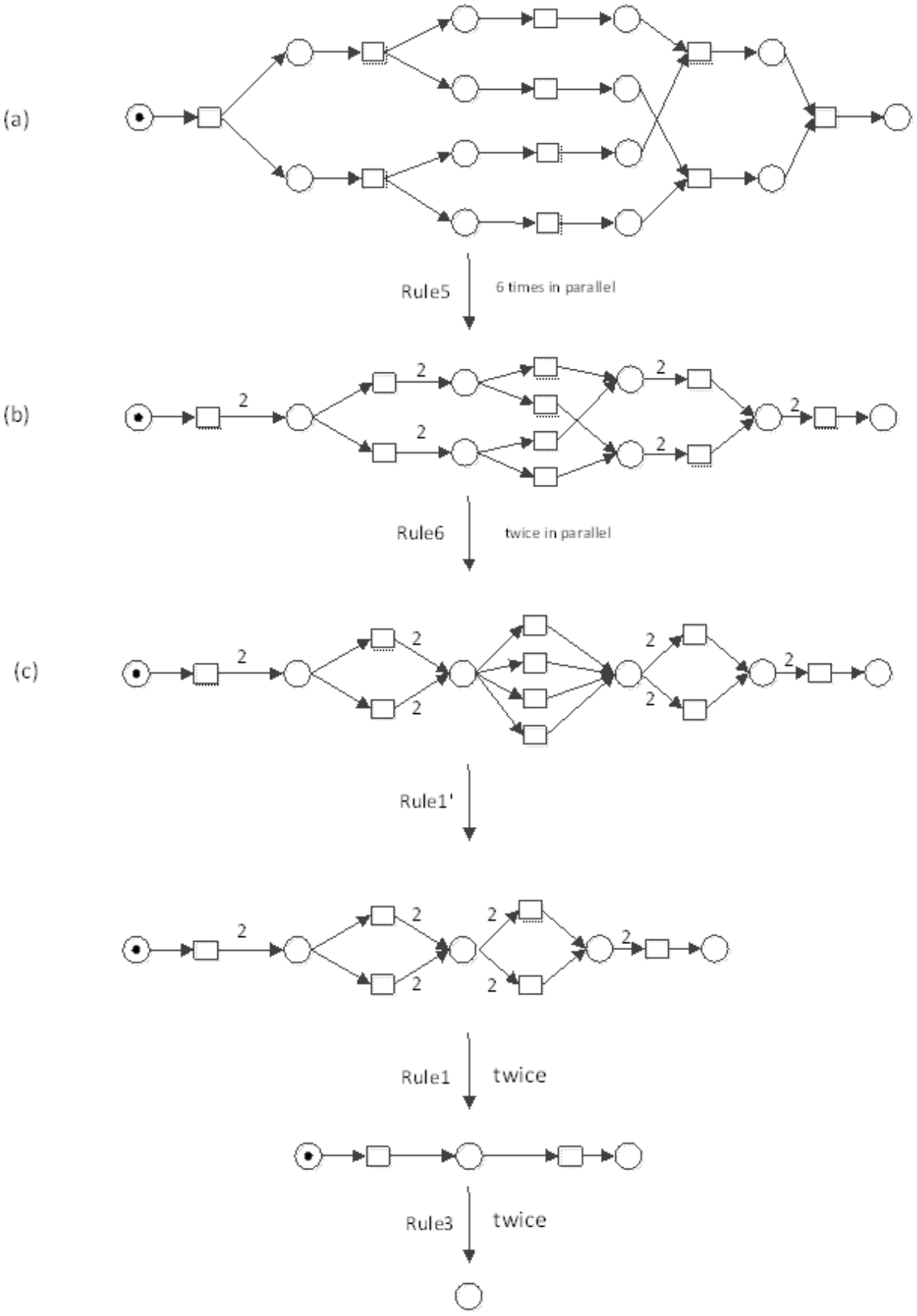


Figure 23. Reduction Process

6.4. Develop a workflow logic for a given business

Let T = {t_1, t_2,⋯t_n} and B = {b_1, b_2, ⋯b_m} be respectively the set of business tasks with $<\cdot$ as its immediate successor relation, and the set of blanks on the B-form for a well-designed business. There should be a fixed correspondence between T and B to make clear responsibilities for each of the tasks in T. This correspondence will not be mentioned any more.

Let Σ = (S,T;F,K,W,M_0) be the net system such that T is the task set and S = {p_1, p_2, ⋯p_k}∪{s_0} ∪E where for each i, i = 1, 2, ⋯, k, k is the number of non-end tasks t_1, t_2, ⋯t_k in T, $\cdot p_i$ = {t_i}, $p_i\cdot$ is the set of immediate successors of t_i. Let ai be the number of tasks in this immediate successor set, a_i' is the intended number of tasks to be executed after t_i, then K(p_i) = a_i', and W(t_i, p_i) = a_i' for every immediate successor t of t_i, W(p_i, t) = 1. It is easy to see that p_i is a synchronizer: p_i = ({t_i}, $p_i\cdot$ ' (1, a_i')).

The place named s_0 is the unique start place: it has an empty pre-set and its post-set contains the unique start task with arc weight 1.

For every end-task in T, there is a unique end place e in E with 1 as the arc weight between them. E contains nothing else.

$M_0(s_0)$ = 1 and all other elements in S have no token.

As a workflow logic of a business process, transition rules for RP/T-systems are assumed for Σ.

Σ must be well-ordered since $<\cdot$ is exactly its next relation among transitions.

Σ may be not well-structured as shown by Figure 12 (a). In this case, measures must be taken as suggested from Figure 12 (a) to (b), then to Figure 13.

Redundant synchronizers may be removed by Reduction Rule 3.

As an example, Let T = {t_1, t_2, t_3, t_4, t_5} and $<\cdot$= {(t_1, t_2), (t_1, t_3), (t_2, t_4), (t_2, t_5), (t_3, t_4), (t_3, t_5)}. There are 3 non-end tasks in T, namely t_1, t_2 and t_3. So there 3 synchronizers: p_1, p_2 and p_3. Figure 24 shows the developed workflow logic.

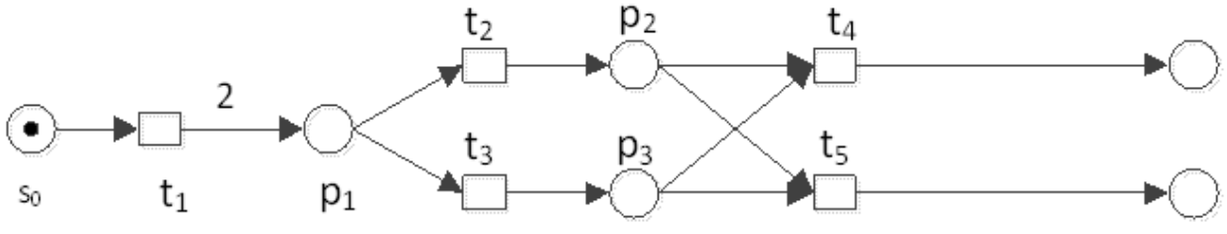


Figure 24. A developed Workflow Logic

There is redundancy in the developed system at p_2 and p_3. By Reduction Rule 3, these two synchronizers are reduced to a single synchronizer p = ({t_2, t_3}, {t_4, t_5}, (2, 1)). This is the workflow logic for insurance claim, as shown in Figure 22, there it was reduced to harmony.

7. Case semantics

Let $\Sigma = (S,T;F,K,W,M_0)$ be the workflow logic developed for a business process. Case semantics is to select a route for the processing of a given case.

A healthy enterprise should have new cases arrive every day. It is impossible to have different routes for different cases. Cases must be classified so that one route for one case class.

Let $C= \{c_1, c_2, \cdots c_k\}$ be the set of conditions for case classification. For example, let Boolean variable x, y be used to record conclusions given by tasks "check policy" and "check claim" respectively for insurance claim i.e. x and y are blanks on the B-form. One way to specify case classes is: $c_1 = x \wedge y$, $c_2 = \neg x \vee \neg y$. This classification is consistent with the workflow logic developed in Figure 24. Another way of case classification is to have 4 classes: $c_1 = x \wedge y$, $c_2 = \neg x \wedge y$, $c_3 = x \wedge \neg y$, $c_4 = \neg x \wedge \neg y$. This classification leads to a different task set: there have to be 4 end tasks. Thus the workflow logic is also different. The B-form should as well be changed accordingly.

Before going on from conditions to routes, a definition of routes is needed.

Definition 19

A route on a workflow logic $\Sigma = (S,T;F,K,W,M_0)$ is a RP/T-system $\Sigma' = (S',T';F',K',W',M_0')$ such that S',T', and F' are subsets of S,T and F respectively:

S 'contains the start place s, a unique end place e, and all synchronizers in S that have a directed path to e

T' consists of the start transition (task), the transition in the pre-set of e, and for every synchronizer $p= ({}^{\cdot}p, p^{\cdot}(a, b))$ in S', a transitions from ${}^{\cdot}p$ and b transitions from $p^{\cdot}$.

$F' = \{(x,y) \mid x,y \in S' \cup T' \wedge (x,y) \in F\}$.

For $x \in S'$, $K'(x) = K(x)$; For $(x, y) \in F'$, $W'(x, y) = W(x, y)$; $M_0'(x) = M0(x)$ for all x in S' .◆

The difference between $\Sigma = (S,T;F,K,W,M_0)$ and $\Sigma' = (S',T';F',K',W',M_0')$ appears when a synchronizer p is an OR-join and/or OR-split. Conditions $c_1, c_2, \cdots c_k$ will be used to select a transitions from ${}^{\cdot}p$ and b transitions from $p^{\cdot}$. Thus, variables (blanks in B) used in these conditions are explicit variables, since they will appear explicitly in the system model for case semantics.

7.1. C-net: Net with variables [5,8]

C-net was originally defined for programming, i.e. for computation and communication. Variables are essentially different from conventional place elements of directed nets. A conflict between transitions in a net system is caused by either the lack of tokens in a shared place, or by the limited capacity of a shared place; a conflict between operations on a variable is read-write conflict: simultaneous read-write on the same variable is impossible

by nature regardless of the exact value of that variable. Besides, there is a rich variety of data types and operations for data types. Conditions for case classification cannot be formally expressed without variables and operations on variables. The example of insurance claim explains: $c_1 = x \wedge y$, $c_2 = (\neg x \wedge y) \vee (x \wedge \neg y)$, $c_3 = \neg x \wedge \neg y$ provide a third way of case classification.

When variables are introduced into a net as another kind of state elements, a net becomes a C-net. A simplified version of C-net is introduced next, just to serve the need of modeling case semantics.

Definition 20

$\Sigma = (S,V,T;F,K,W,R,Wr, M_T, M_0)$ is a C-net system if

S,V,T are the sets of places, variables and transitions respectively; F,R,Wr are respectively the flow relation, the read relation and the write relation; K,W are the capacity function and the arc weight function; M_T is a marking on T and M_0 is a marking (initial marking) on S.◆

A transition may read and/or write a variable, but not must. The marking on a transition has a Boolean expression as its guard and assignments as its body when it writes. The guard and the body must be consistent in relation to read and write. For example, the guard can only refer to variables that are to be read by the transition.

If a transition has nothing to do with variables, conventional transition rules apply. If a transition has nothing to do with places, it is enabled when its guard is true. If a transition is related with places as well as variables, it is enabled if it is enabled by places and its guard is true.

Isolated transitions are excluded in a C-net.

An enabled transition may fire to produce a successor marking on S, the marking on T remains like the weight function and the capacity function; variables in write relation with this transition are assigned values by the assignments.

A c-net transition in general has a figure called "status" as part of M_T. Transition firings may change "status". But this does not concern us.

To save space, detailed formal definitions are avoided. Figure 25 illustrates them, including graphical presentation of C-nets.

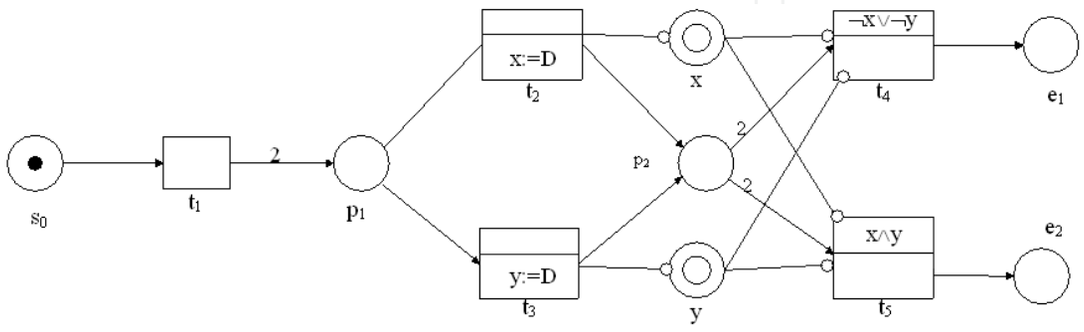


Figure 25. A C-net System: Case Semantics for Insurance Claim

The case semantics of insurance claim given in Figure 25 divides conceivable cases into 2 classes. Variables like x and y are drawn as double circles, write relation is represented by a zero-headed arc from transition to variable, reversed zero-headed arc represents read relation. For example, t_2 writes x and t_4 , t_5 read x.

The start transition t_1 is in no relation with x and y, it has no guard, no body. The guards of both t_2 and t_3 are "true", denoted by default. The write operations of these two transitions are to be performed by their executers, denoted as x:= D where D is the abbreviation of "default". Transitions t_4 and t_5 have conditions for case classification as their guards. They have empty body since they write no variables.

Apparently, the case semantic contains two routes: one for each class. As long as the classification of cases is complete and consistent (every case falls into exactly one class), there would be a unique route for each case.

7.2. Workflow logic and its case semantics

Let $(T,<\cdot)$, $B=\{b_1, b_2, \cdots b_m\}$ and $C = \{c_1, c_2, \cdots c_k\}$ be the main constituents of a business process. There is a precise correspondence between tasks in T and blanks (on B-form) in B; Condition set C is complete and consistent; the immediate successor relation $<\cdot$ is implied by a combinatorial acyclic partial order on T. With all these requirements satisfied, $((T,<\cdot), B, C)$ is a well-designed business process. We will not put all these into formal definitions since that would be cumbersome and uninteresting.

Definition 21

The C-net system $\Sigma'=(S,V,T;F,K,W,R,Wr,M_T,M_0)$ is a case semantics of the workflow logic $\Sigma=(S,T;F,K,W,M_0)$ developed for well-designed business process $((T,<\cdot), B , C)$, if the marking M_T (which is constant) satisfies that for those t in T that have a guard, the guard is a condition in C, and every condition in C is the guard of some transition.◆

Definition 22

Let C-net system $\Sigma'=(S,V,T;F,K,W,R,Wr,M_T,M_0)$ be a case semantics of the workflow logic $\Sigma=(S,T;F,K,W,M_0)$ developed for well-designed business process $((T,<\cdot), B , C)$. If Σ is through, and for every OR-split synchronizer $p = (\cdot p, p^{\cdot}(a, b))$, $b<|p^{\cdot}|$, for every condition ci in C, i = 1,2, ⋯ k, $|\{t|t\in p\cdot \wedge \text{guard}(t) = c_i\}|= b$, then Σ' is consistent with Σ.◆

Theorem 8

If Σ is a through workflow logic and Σ' is a case semantics consistent with Σ, then there is a unique route for every conceivable case. ◆

For a case in class c_i, the route it will take is obtained by deleting from Σ all transitions whose guard is not c_i together with their successor elements.

Figure 26 shows another case semantics of the insurance claim business process where cases are divided into 3 classes by conditions $c_1 = x\wedge y$, $c_2 = (x\wedge\neg y)\vee(\neg x\wedge y)$ and $c_3 = \neg x\wedge\neg y$. The task set has one more end-task for this classification.

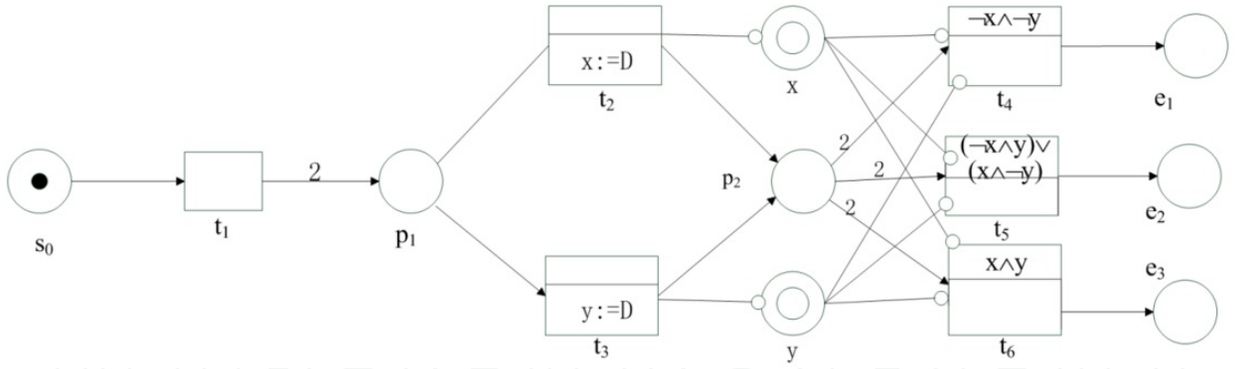


Figure 26. Another Case Semantics for Insurance Claim

The only OR-split synchronizer in Figure 26 is p_2. It has 3 tasks in its post-set and each of which bears a different condition as its guard. This is consistent with p_2: p_2 is a XOR-split synchronizer.

8. Business process management

So far we have proposed a formal model as workflow logic and a formal model as case semantics for a well-designed business process. Case semantics in embedded in a Workflow logic when they are consistent with each other. There is a unique route for every conceivable case as the semantics of that case. Thus, a firm foundation has been laid for management automation. The computer system that conducts the processing of each given case based its semantics is the workflow engine.

8.1. Management logic

As suggested by Figure 8 (b), every synchronizer $p = ({}^{\cdot}p, p^{\cdot}, (a, b))$ requires management (transition t in the detail of the synchronizer) to keep tasks in ${}^{\cdot}p$ well synchronized with transitions in $p^{\cdot}$. The start-place requires management, so do all end-places. At the start-place, the engine must recognize the proper business process when a case (B-form) arrives, and to have the corresponding workflow logic ready, and to appoint an executer for the start-task and passes the B-form to the executer. At an end-place, the engine must check whether the case is well processed and to put the completed B-form in place.

All in all, places and synchronizers are exactly where the engine does its duty. In other words, the dual of workflow logic is the management logic, and the dual of case semantics with given condition c_i in C produces the route for all cases classified by c_i.

Definition 22

1. The directed net $N' = (T,S;F)$ is the dual net of directed net $N = (S,T;F)$.
2. $\sum' = (T, S; F,K',W', \mathbf{M}_0)$ is the workflow management logic for workflow logic $\sum = (S, T; F,K,W, M_0)$, where $K'(t) = \sum W(p,t)$ for every t in T, i.e. the capacity of t is the sum of all weights on its input arcs; $W'(x, y) = W(x, y)$ for all (x, y) in F; $\mathbf{M}_0(t) = 0$ for t in T.
3. $(\sum,\sum')$ is the logic pair for the business process in question.◆

Figure 27 is the logic pair for insurance claim with two case classes.

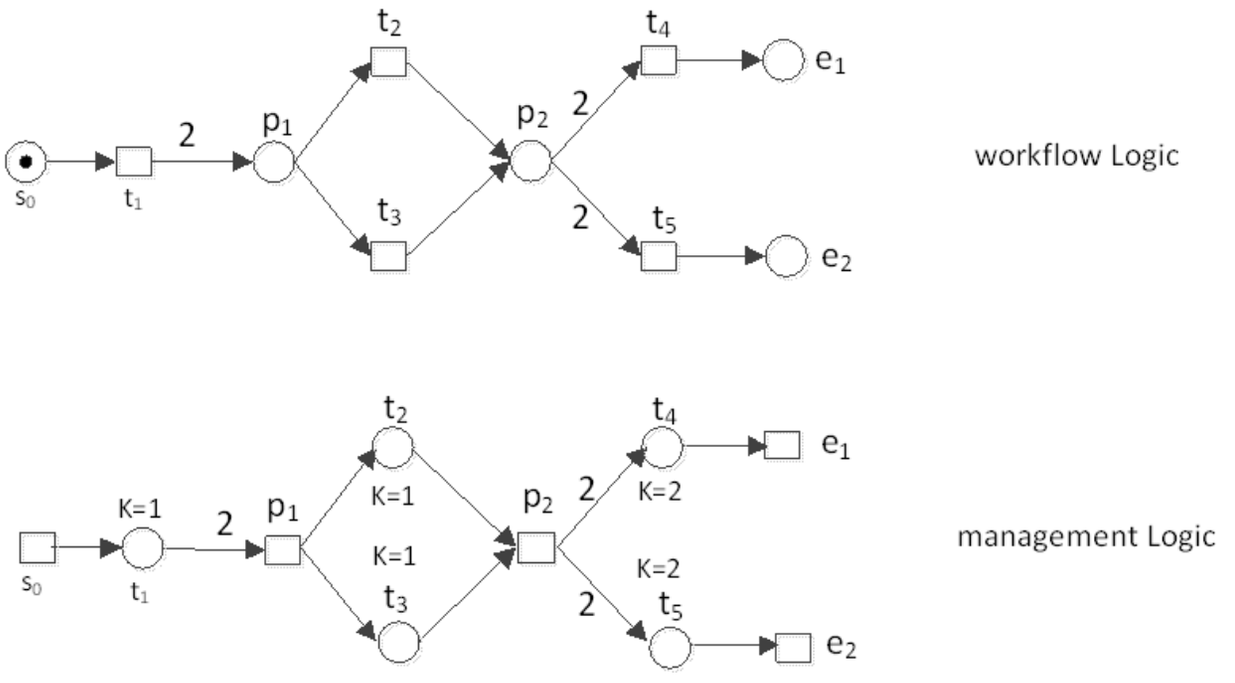


Figure 27. Logic Pair

Management tasks (transitions in Σ') and business tasks (transitions in Σ) are executed in turn: whenever a place (including synchronizers) in Σ has a full capacity of tokens, the transition with the same name in Σ' is enabled. The enabled transition fires to put tokens into its output places in Σ', whenever a place in Σ' has a full capacity of tokens, the transition with the same name in Σ is enabled. This interactive transition firings continue till a transition named with an end-place is fired to move the case token out of the pair. This is the way how management tasks and business tasks interact with each other.

8.2. Route for cases in the same class

A logic pair specifies, for a given business process, how individual cases are processed under the conduction of the engine (the manager). Figure 28 shows a case semantics and its dual, that is a semantic pair, a pair of C-net systems. The guard on a transition (task) will be used by the engine (transitions named after a synchronizer) for the selection of routes. We will not give a formal definition of this pair, since it is clearly specified by the figure. The interaction between this pair is the same with the logic pair, except details given by guards.

Figure 27 and Figure 28 serve as the main part of a formal specification for a workflow engine. The rest of the engine is enterprise specific (personal, organizational division etc.).

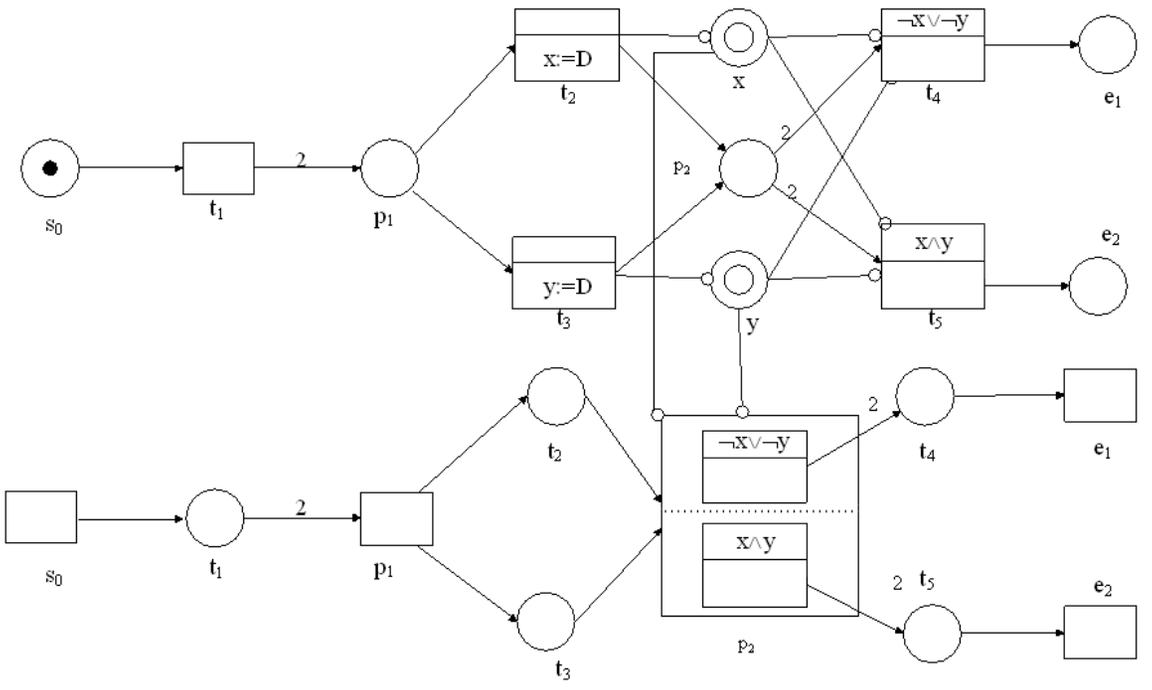


Figure 28. Semantic Pair

8.3. Duty of the engine

The main duty of the engine is to conduct the manipulation of individual cases based on the logic pair (before the class to which the case belongs is known) and the semantic pair (transitions named after an OR-split synchronizer), including appointing executers for selected tasks.

In addition to this major duty, other duties include (not exclusive):

- Set a time upper bound for each task to be executed next, and to remind before expiration, check whether it is done on time;
- Check whether a task is properly done (e.g. whether the content filled in a blank matches its type);
- Decide, based on given rules, what to do when an expiration or quality failure is detected (e.g. redo for quality failure, bad record on the due person for expiration);
- Other thinkable duties, especially enterprise specific ones.

An engine carries out its duty properly if and only if it has predefined management rules to follow. In addition to global rules specified by the logic pair and the semantic pair, local rules are also necessary, i.e. rules for every management task (every place in workflow logic). Remember that all places in workflow logic are where the engine can and must put a hand on.

Some management rules are closely related to individual enterprises, some are even secrete to outsiders. Current and historical business data, business partners data, conventions or traditions etc. All rules related to these must be carefully established and checked together with a well-designed business process, a well-structured workflow logic of the process, a

case semantics consistent with the logic. The two pairs are derivable from the logic and semantics. We go no further here in this chapter.

9. Conclusions and acknowledgement

Prof. Carl Adam Petri wrote: "In order to apply net theory with success, a user of net theory can just rely on the fact that every net which he can specify explicitly (draw on paper) can be connected by a short (≤ 4) chain of net morphisms to the physical real world; your net is, in a very precise sense, physically implementable".

We have seen this quotation at the very beginning of this chapter. This chapter tries to make clear how to specify a net that is "in a very precise sense physically implementable". A successful application of net theory starts from a full understanding of the application problem. A well designed business process ((T,<·), B ,C) leads to the discovery of workflow logic, case semantics and management pairs, a tree-layer model for workflow modeling.

Without theory, full understanding of application problems becomes hard. Well-structured business process ((T,<·), B , C) starts from a clear distinction between business process and workflow, a clear distinction between business tasks and management tasks, and a clear distinction between transition synchronizations and place synchronizations. Business process and workflow are not synonym of each other.

A full understanding of application problems and a good grasp of theories in Petri nets that build a road to success.

The author is grateful to the editors of this book. It is a great honor to me as well as a good chance for me to exchange ideas on Petri nets with friends outside my country.

Author details

Chongyi Yuan
School of Electronics Engineering and Computer Science, Peking University, Beijing, China

10. References

[1] C.Y. Yuan: *Petri Nets* in Chinese, Southeast University Press,(1989).
[2] W. Brauer (editor): Net Theory and Applications, Springe LNCS Vol.84.
[3] C. A. Petri: Nonsequential Processes GMD-ISF Report 77, (1977).
[4] C.A. Petri: Concurrency incl. in 2 (in Theoretical Computer Science).
[5] C.Y. Yuan: Principles and applications of Petri Net. Beijing, China: Publishing house of electronics industry, 2005.
[6] W. Aalst, K. Hee: *Workflow Management*, The MIT Press (2000).
[7] C.Y. Yuan, W. Zhao, S.K. Zhang, Y. Huang: A Three Layer Model for Business Process —Process Logic, Case Semantics and Workflow Management, Journal of Computer Science & Technology, Vol.22 No.3, 410-425 (2007).
[8] C.Y. Yuan: *Petri Net Application*, to appear .