Chapter 3

Structured Business Process Design of Supply Chain Processes

In this chapter, a structured process modeling algorithm for supply chain business processes using CIMOSA (Computer Integrated Manufacturing Open System Architecture) behavioral rules and free-choice Petri nets is presented. The well-behaved properties such as liveness and boundedness are critical for supply chain business processes. They enable the avoidance of possible deadlock, endless loops and dangling tasks in process models. However, for complex systems, the checking of the well-behaved property is a NP-hard problem. In this chapter, a structured process modeling algorithm is presented. This algorithm employs six well-behaved control structures, the chaining rule and the modified nesting rule. The resulting process models from this algorithm are free-choice Petri nets whose well-behaved property can be decided in polynomial time. The process models resulted from this structured process modeling algorithm also have the advantages of modularity, readability and maintainability. A partial model of shop floor domain of a supply chain process built by using the proposed structured process modeling algorithm has been presented as an illustration example.

This chapter is organized as follows. Section 1 gives the basic constructs of the *function view* of CIMOSA. Section 2 and 3 provide a structured process modeling algorithm and show that structured supply chain process models from this algorithm are free-choice process-nets. In Section 4, an industrial application is used to illustrate this algorithm. Section 5 extends the free-choice process-nets to include events. Summaries and important conclusions are provided in Section 6.

3.1 Constructs of the Function View

3.1.1 Process Models of CIMOSA

CIMOSA views the manufacturing enterprise as a set of functional entities processing enterprise objects (associated with information), operations of which are governed by a set of communicating concurrent processes (Vernadat 1996). Essential constructs of the CIMOSA function view are *event*, *domain*, *domain process*, *business process*, *enterprise activity* and *functional operation*. They are used to model enterprise functionality and behavior.

3.1.2 Business Process Entity Definition

CIMOSA uses a set of constructs to define the problem and decompose the enterprise model into manageable modules. These modules are called *domains*. Domains interact with one another by the exchange of *events* and results. Domains comprise a set of core processes called *domain processes* that represent stand-alone processes of the enterprise. In other words, a *domain process* is a sequence of activities of an enterprise with well-defined starting conditions and providing a defined, measurable or quantifiable, end-result.

3.1.3 Manufacturing Enterprise Behavior Specification

Once domains have been established and their relationships made explicit, events and domain processes of each CIMOSA domain can be defined. A domain process can be decomposed into lower-level processes, which further decomposed into sub-processes, and so on using the functional decomposition principle. This will result in a tree structure of functions and sub-functions. The root of each functional decomposition is a *domain process*, the intermediate nodes are *business processes*, and the leaves are *enterprise activities*. This naturally defines three different types of constructs to represent functions. Domain processes are triggered by events only. They indicate the sequence of enterprise activities to be executed to realize the

desired enterprise behavior. They can be defined as a network of activities connected by so-called *behavioral rules*.

Business processes are similar to domain processes except that (1) they cannot be triggered by events but must be called upon by a parent structure (domain process or higher-level business process), and (2) their termination conditions must be defined as *ending statuses*. Ending statuses are values defined by users and characterizing the execution status of the process.

Behavioral rules define the flow of actions describing the behavior of a domain process. They govern the sequence of execution of enterprise functionality according to the system state (event-driven control flow). They are control structures relevant to supply chain activities and cover sequential, parallelism, rendezvous and iterative controls. Therefore, the process behavior of a business process can be defined as a set of behavioral rules. Generally, a behavioral rule can be written as a set of conditions and a related action:

WHEN (conditions) DO action

CIMOSA provides nine rules to describe behavior of structured business processes, which are presented in Table 3.1 (in which *EF* denotes enterprise function and can represent an enterprise activity or a business process, and *ES* is the ending status of an enterprise function).

3.1.4 Manufacturing Enterprise Functionality Specification

Enterprise activities define enterprise functionality. An *enterprise activity* is defined in CIMOSA as a set of elementary actions to be considered as a whole, i.e., requiring resources and time for its full execution. Resources required by enterprise activities are *functional entities*. Functional entities provide *functional operations* or services used to realize the functionality of the enterprise activity. *Functional operations* used in the activity behavior are units, or atoms, of

functionality. Rotating a robot arm can be regarded as a functional operation. Functional operations cannot be decomposed. There must be at least one functional entity in the system that can provide or perform them. This research focuses on the function view of systems.

Table 3.1 CIMOSA behavioral rules for process workflow (ESPRIT Consortium 1993)

Types	Syntaxes	Descriptions
Process triggering rules	WHEN (START WITH evnet_1) DO EF1	They are used to start a process by means of one or more events.
Forced sequential rules	WHEN (ES(EF1)=any) DO EF2	They cause the next enterprise function to be activated independent of any status information.
Sequential rules	WHEN (ES(EF1)=end_stat_1) DO EF1	They cause the next enterprise or business process to be activated according to the value of an ending status.
Conditional sequential rules	WHEN (ES(EF1)=end_stat_1) DO EF2 WHEN (ES(EF2)=end_stat_2) DO EF3 WHEN (ES(EF3)=end_stat_3) DO EF4	They are used to represent branching conditions in the flow of control. They are equivalent to an exclusive OR.
Spawning rules	WHEN (ES(EF1)=value_1) DO EF1 & EF2 & EF3	They are used to represent the parallel execution of functions in a flow of control, & is the parallel operator.
Rendezvous rules (Logical AND)	WHEN (ES(EF1)=value_1 AND ES(EF2) =value_2 AND ES(EF3)=value_3) DO EF4	They are used to synchronize the end of spawning rules.
Convergence rules (Logical OR)	WHEN (ES(EF1) = value_1 OR ES(EF2) = value_2 OR ES(EF3)=value_3) DO EF4	They cause the next one or more enterprise functions to be activated after one of the paths activated by a defined conditional procedural rule is completed.
Loop rules	WHEN ES(EF1)=loop_value DO EF1	They are used to repeat the same enterprise function.
Process completion rules	WHEN ES(EF2)end_stat_1 DO FINISH	They are used to finish the process.

3.2 Supply Chain Business Process Design

3.2.1 Motivation

During the1960s, many large software development efforts encountered severe difficulties. Software schedules were typically late and the finished products were unreliable. People began to realize that software development was a far more complex activity than they had imagined. Research activity in the 1960s resulted in the evolution of *structured programming* — a disciplined approach to writing programs that are clearer than unstructured programs, easier to

test and debug, easier to understand, and easier to modify. Just like structured programs (e.g., a structured program can be written in C language) can be written in terms of only three control structures (i.e., sequence structure, selection structure and repetition structure) by following some rules, structured processes can be designed in terms of well-behaved control structures by following some modeling algorithms.

3.2.2 Well-behaved Routing Structures in Supply Chain Processes

In this section, nine behavioral rules in CIMOSA are consolidated into six well-behaved constructs that can then be used to build structured supply chain process models. Figure 3.1 gives nine supply chain process routing structures and their corresponding Petri net control structures.

To design a structured process model, well-behaved control structures are used instead of direct use of nine behavioral rules. In general, for Petri nets, arbitrary two of behavioral rules can be combined together. But in practice, when a process is constructed, no arbitrary two types of control structures can be combined together directly. This is because that some combinations lack practical meanings or result in errors. Two pragmatic combination rules are given in the following.

If several concurrent activities initiated by spawning rules are joined by convergence rules, from the view of Petri nets, then only one of the completions of concurrent activities is needed to fire the succeeding process. That means other finished activities are useless. These non-value-adding activities should be eliminated from a well-behaved process. Combination rule 1 is used to avoid this situation.

Combination Rule 1: Concurrent activities initiated by spawning rules (or AND-splits) should not be joined by convergence rules (or OR-joins).

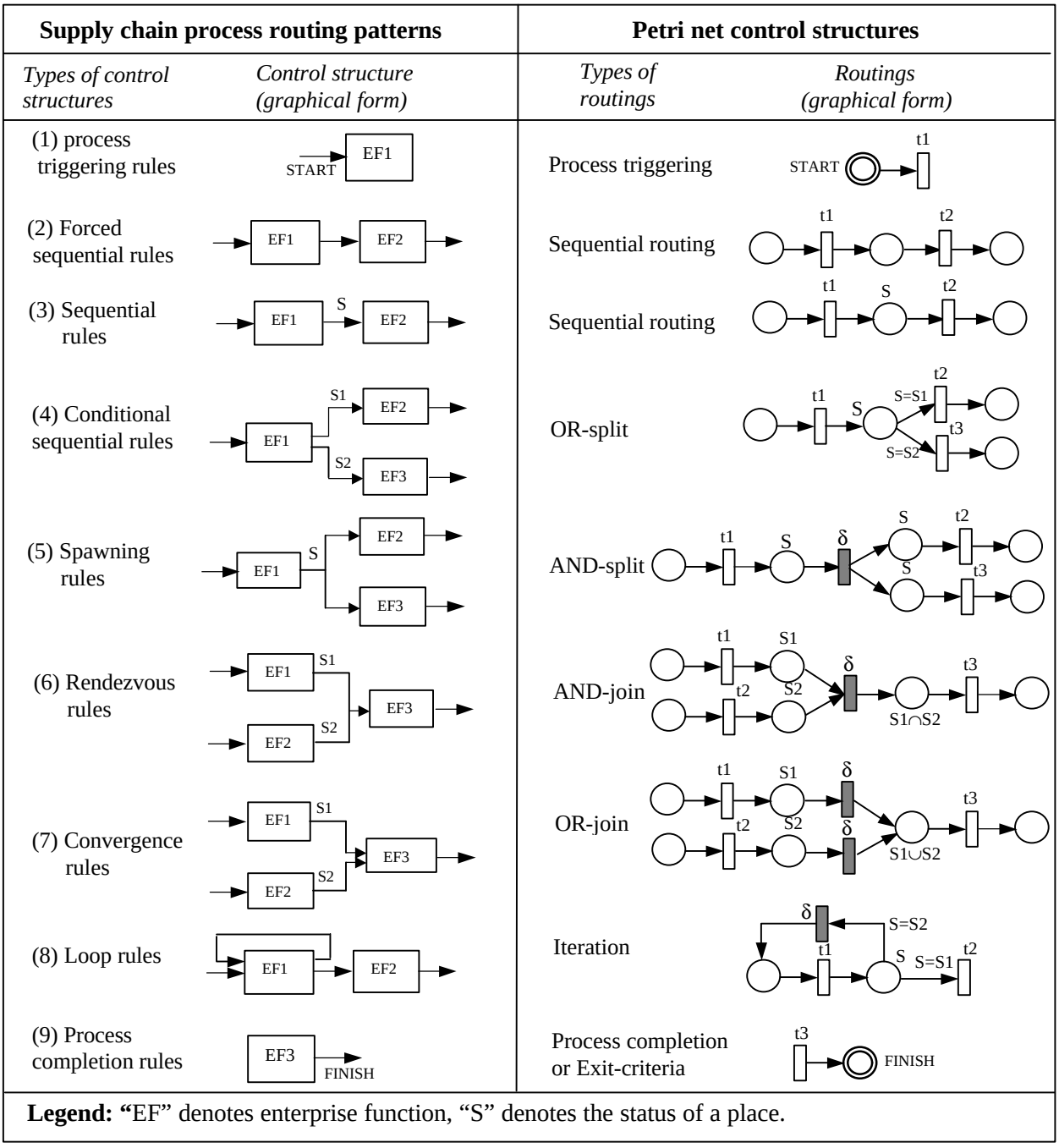


Figure 3.1 Supply chain process routing structures and Petri net representations

If several alternative activities initiated by conditional sequential rules are joined by rendezvous rules, from the view of Petri nets, then a deadlock will occur because only one of the alternative activities will be finished. Such a deadlock should be eliminated from a well-behaved process. Combination rule 2 is used to rule out this situation.

Combination Rule 2: Alternative activities initiated by conditional sequential rules (or OR-splits) should not be connected by rendezvous rules (or AND-joins).

By applying these two combination rules, two well-behaved constructs can be obtained.

(1) **Parallel structure:** AND-split is used to represent the starting of a concurrent structure and AND-join is used to indicate the ending of the concurrent structure. So their combination becomes the parallel structure.

(2) **Selection structure:** OR-split is used to represent the starting of a selection structure and OR-join is used to indicate the ending of the selection structure. So their combination becomes the selection structure.

To define a process model, other four control structures are needed.

(3) **Sequence structure:** Forced sequential rules can be treated as a special case of sequential rules. Thus, sequential rules can be used as sequence structures.

(4) **Repetition structure:** Loop rules in supply chain process modeling can be directly used as repetition structures.

To model a process, starting and ending points are also needed. Process triggering rules and process completion rules are for such purpose.

(5) **Starting point:** process triggering rules.

(6) **Ending point:** process completion rules.

Above six well-behaved control structures can be modeled by using Petri nets (see Figure 3.2). In Figure 3.2, an *Entry* place and an *Exit* place are defined for each of sequence, selection, parallel and repetition structures. *d* is a dummy transition with execution time equal to *0*. It is easy to see that all of them are free-choice Petri nets (with reference to the definition of free-choice Petri nets).

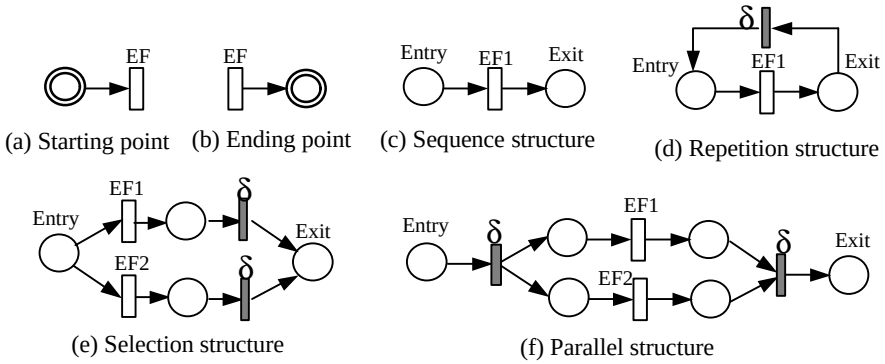


Figure 3.2 Free-choice Petri net representations of six well-behaved constructs of supply chain

3.2.3 Structured Supply Chain Business Processes

Similar to workflow-nets in Van der Aalst (1997), a process net can be defined as follows.

Definition 1 (Process-net). A Petri net $PN=(P, T, F)$ is called a process net if it meets the following two conditions:

(1) PN has an input place i and an output place o. i.e., $\bullet i = \phi$ and $o\bullet = \phi$.

(2) If a transition t^* is added to PN which connects place o with i (i.e., $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$), then the resulting Petri net is strongly connected.

(3) In this definition, input place stands for the starting of a business process and output place stands for the completion of a process. The second condition eliminates dangling tasks (those tasks may never be implemented).

Definition 2 (extended process-net). An extended process-net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$ is obtained by adding an extra transition t^*, which connects o and i. Its definition is as follows:

$$\overline{P} = P, \overline{T} = T \bigcup \{t^*\}, \overline{F} = F \bigcup \{< o, t^* >, < t^*, i >\}.$$

A structured supply chain business process has some well-behaved properties. Some desired properties are stated as follows.

Free-deadlock: A process should not have any deadlock situations. The liveness property in Petri nets can be used to detect the possibility of deadlock.

Boundedness: This property refers to the limited capacity of a resource (no overflow).

Endless loops: A process should not have any infinite loops. Otherwise, some task will be executed infinitely.

Proper termination: A process that starts from *i* should be able to reach the ending status *o* by executing a series of enterprise functions. When a process terminates, *o* is the only state reachable from state *i* with at least one token in place *o*.

No dangling tasks: A process should avoid any dangling tasks.

These properties are related each other. For example, proper termination property can be achieved if the corresponding extended process net is live and bounded. Above properties correspond to the correctness properties of a process. In addition to correctness properties, a good business process modeling approach should consider the modularity, readability and maintainability.

Modularity, readability and maintainability: A good business process modeling should employ the principles of modularity and top-down design, and a good process model should be easy to understand and modify.

Therefore, a structured process model can be defined as follows.

Definition 3 (Structured supply chain process models). A supply chain process model is said to be structured if (1) its extended process net is live and bounded; and (2) it has the properties of modularity, readability and maintainability.

3.3 Structured Supply Chain Business Process Modeling Algorithm

The modeling algorithm for designing structured supply chain business processes is as follows.

Step 1. Each enterprise function (business process or enterprise activity) of the supply chain process model becomes a transition in the Petri net model and each ending status or condition becomes a state place.

Step 2. START in the process triggering rules becomes source place (represented by "Starting point"). FINISH in the process completion rules becomes sink place (represented by "Ending point").

Step 3. Apply the chaining rule and the modified nesting rule in any times and in any order (dummy transition δ can not be replaced).

In this structured process modeling algorithm, if the modified nesting rule is replaced by the nesting rule in step 3, then we call this algorithm as the **simple-process-net algorithm** because the resulting net from it is a simple net.

Chaining rule, nesting rule and modified nesting rule are briefly defined below.

Definition 4 (Chaining rule). Any routing structures (sequence, selection, parallel and repetition structures) can be chained with "Starting point" and "Ending point" control structures by connecting corresponding transitions and places. Any control structures (sequence, selection, parallel and repetition structures) can be chained together by merging "Exit" place of the previous one and "Entry" place of the current chained one.

Definition 5 (Nesting rule). (1) Begin with a simple sequence structure (execute only once). (2) Any transition (except dummy transitions) can be replaced by any control structures (sequence, selection, parallel and repetition structures). Because dummy transitions cannot be

replaced, each replaced transition has only one input place and only one output place (see Figure 3.1). Then its input place will merge with "Entry" place of the replacing control structure and its output place will merge with "Exit" place of the replacing control structure.

In the nesting rule, the execution time of a dummy transition equals to *0* and it does not represent any enterprise functions. The introduction of dummy transitions does not change the semantics of original nets. Dummy transitions do not change the states of their input places, they just propagate the states of their input places into their output places.

In the following, we will prove the resulting Petri net models from the chaining rule are free-choice Petri nets, and the ones from the nesting rule are simple Petri nets (its definition is given in the Appendix A, see Descrocher et al 1995). Because free-choice Petri nets have strong theoretical results and efficient analysis techniques such as Rank Theorem, it is desirable to restrict process-nets to free-choice Petri nets. The modified nesting rule is presented for this.

Definition 6 (Modified nesting rule). (1) Begin with a simple sequence structure (execute only once). (2) Any transition can be replaced by any control structures (sequence, selection, parallel and repetition structures). For each replaced transition, its input place will merge with "Entry" place of the replacing control structure or its output place will merge with "Exit" place of the replacing control structure. (3) When a transition *t* in parallel structures is replaced by a repetition structure, arcs are created to connect places $({}^{\bullet}(t^{\bullet\bullet}))$ to transitions $(({\it Exit})^{\bullet})$.

In the modified nesting rule, step 3 is introduced to eliminate "asymmetric confusion" situation (Figure 3.3 gives an asymmetric confusion example). Confusion is a mixture of concurrent and conflict.

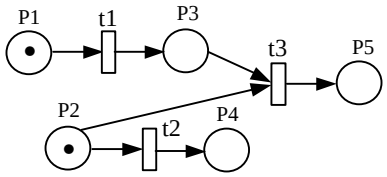


Figure 3.3 An example of asymmetric confusions

In Figure 3.3, t_1 and t_2 are concurrent (both enabled and firable), and if t_1 fires first, then t_3 and t_2 will be in conflict. One characteristic of this case is that there is one place (p_2) that is input to two transitions and a place (p_3) that is input to a subset of $p_2^\bullet$.

In the following we will prove the resulting Petri-net-based process models from the modified nesting rule are free-choice Process-net. First of all, some concepts are given below.

Definition 7 (Clusters) (Desel et al 1995). Let x be a node of a net. The cluster of x, denoted by $[x]$, is the minimal set of nodes such that (1) $x \in [x]$, (2) if a place s belongs to x then $s\bullet$ is included in $[x]$, and (3) if a transition t belongs to $[x]$ then $\bullet t$ is included in $[x]$.

Theorem 1. A process net resulted from the "simple-process-net algorithm" is a simple net; a process net resulted from the "structured business process modeling algorithm" is a free-choice process net.

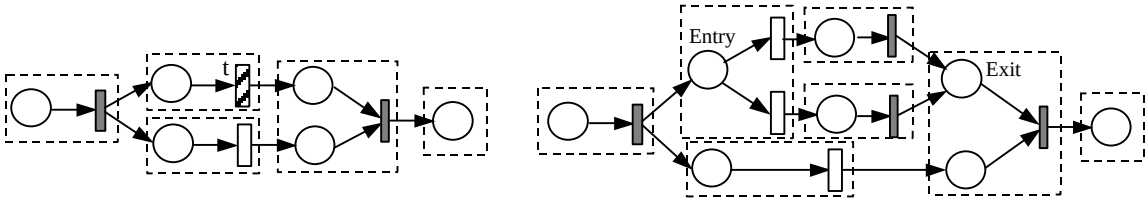
Proof. First, we prove a process model resulted from the "simple-process-net algorithm" or "structured business process modeling algorithm" is a process-net. (1) It is easy to see such a process model has a source place and a sink place, so first requirement in process-net definition is met. (2) If a transition t^* is added to each of sequence, selection, parallel and repetition structures which connects place *Exit* with *Entry* (i.e., $\bullet t^* = \{Exit\}$ and $t^*\bullet = \{Entry\}$), then each resulting Petri net is strongly connected. In other words, the extended Petri nets of sequence, selection, parallel and repetition structures are strongly connected. (3) Because the chaining rule simply connects six well-behaved structures and the extended Petri net of each structure is

strongly connected, a process model resulted from the chaining rule is still strongly connected. (4) Since the replaced transition has only one input and output place, its input or output place will be merged with "Entry" and "Exit" place of the replacing structure in terms of the nesting rule or the modified nesting rule, this situation does not change connectivity.

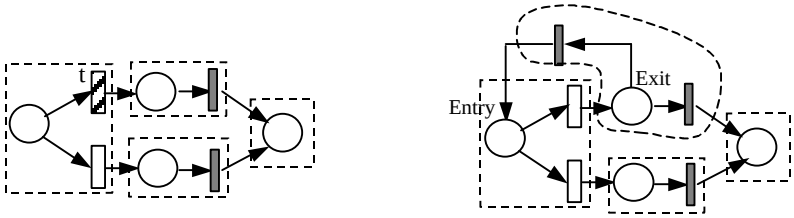
Second, we prove a process model resulted from the "simple-process-net algorithm" is a simple net and a process model resulted from the "structured business process modeling algorithm" is a free-choice process net. (1) Because chaining is done by merging "Exit" place of the previous structure and "Entry" place of the current chained one, the chaining rule does not change the cluster structure of each control structure and chaining points do not create any new arcs or connections. Therefore, a process model derived from chaining those free-choice Petri net control structures is still a free-choice Petri net. (2) The clusters of a free choice Petri net have a particular structure: each place *s* of a cluster *c* is connected to every transition t_c of *c* by an arc (*s*, t_c). It follows that all the transitions of a cluster have the same set of input places, and all the places of a cluster have the same set of output transitions. Sequence, selection, parallel and repetition structures are free-choice Petri nets. So they will inherit this property. In the following, *t* is used to denote the replaced transition. In the nesting rule, $[{}^{\bullet}t, t, t^{\bullet}]$ will removed from the original net, and the replacing control structure will be brought into this original net to replace the $[{}^{\bullet}t, t, t^{\bullet}]$. There are two interfaces between the original net and the replacing structure, i.e., one is Entry interface and the other one is Exit interface. (3) For Entry interface, a new cluster will be created as: $Entry \cup (Entry)^{\bullet} \cup [({}^{\bullet}t)^{\bullet} - t]$. $Entry \cup (Entry)^{\bullet}$ stands for *Entry* place and its output transitions in the replacing structure. $({}^{\bullet}t)^{\bullet}$ stands for those transitions whose input places are ${}^{\bullet}t$. While $[({}^{\bullet}t)^{\bullet} - t]$ takes *t* out of $({}^{\bullet}t)^{\bullet}$. In this new cluster, there is only one place (i.e., *Entry*), and it is the only input place of transitions $(Entry)^{\bullet} \cup [({}^{\bullet}t)^{\bullet} - t]$. In other

words, all the transitions of this cluster have the same input place *Entry*. Therefore, the new cluster in the Entry interface satisfies the characteristic of free-choice Petri nets. (4) For Exit interface, another new cluster is created as: $Exit \cup (Exit)^{\bullet} \cup t^{\bullet\bullet} \cup [{}^{\bullet}(t^{\bullet\bullet}) - t^{\bullet}]$. $Exit \cup (Exit)^{\bullet}$ stands for *Exit* place and its output transitions in the replacing structure. $t^{\cdot\cdot}$ denotes the output transitions of *t*'s output places. ${}^{\bullet}(t^{\bullet\bullet})$ stands for those places that have $t^{\cdot\cdot}$ as output transitions. $[{}^{\bullet}(t^{\bullet\bullet}) - t^{\bullet}]$ takes $t^{\cdot}$ out of ${}^{\bullet}(t^{\bullet\bullet})$. Because the *Exit* places of sequence, selection and parallel structures do not have any output transitions, $(Exit)^{\cdot}$ is empty. Notice that $t^{\cdot\cdot}$ in each control structures (sequence, selection, parallel and repetition structures) contains at most one transition (see Figure 3.2). In this new cluster, *Exit* will replace $t^{\cdot}$ and becomes the input places of $t^{\cdot\cdot}$ along with $[{}^{\bullet}(t^{\bullet\bullet}) - t^{\bullet}]$. This cluster meets the condition of free-choice Petri nets: all the transitions of a cluster have the same set of input places (in this case, the cluster has only one transition). Figures 3.4(a) and 3.4(b) are used to illustrate these cases. The *Exit* place of the repetition control structure has one output transition, *d*. When transition *t* in the repetition structure is replaced by sequence, selection, parallel or repetition structure, same conclusions as above can be reached. When transition *t* in the sequence or selection structures is replaced by the repetition, the same conclusions are kept. The only exception is when the transition *t* in the parallel structure is replaced by the repetition structure. This situation will result in an asymmetric confusion (see Figure 3.4(c)). The reason is that $(Exit)^{\cdot}$ is not empty (*Exit* has one output transition) and $[{}^{\bullet}(t^{\bullet\bullet}) - t^{\bullet}]$ in the parallel structure contains at least one place. (5) Except Entry and Exit interfaces, the remaining clusters in the replacing structures will be brought into the original net directly. Therefore, the resulting net by the nesting rule is a simple net. (6) In practice, iteration is often considered to be undesirable form of routing, because it corresponds to

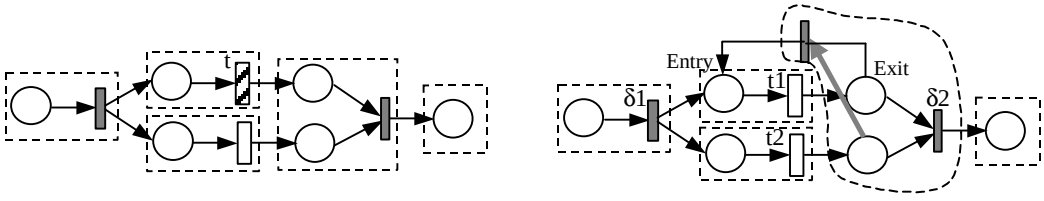
the repeated execution of the same task without making any real progress. But sometimes iteration cannot be avoided. In Figure 3.4(c), there are two concurrent sub-tasks *t1* and *t2* joining together into a task (corresponding to an assembly work in the context of manufacturing) in transition *d2*. The asymmetric confusion is unusual here, because when one sub-task completes, it has to wait another sub-task to finish its non-deterministic iteration. To avoid the situation, arcs from $[^{\bullet}(t^{\bullet\bullet}) - t^{\bullet}]$ to *(Exit)*$^{\cdot}$ are introduced. That means whether an iteration continues or not will be decided by the ending statuses of all sub-tasks. This is much more reasonable in practice. The introduction of new arcs makes the asymmetric confusion become a free-choice structure. These new arcs are called as *control arcs*. Therefore, a process net resulted from the "structured business process modeling algorithm" is a free-choice process net.



(a) A transition in a parallel construct is replaced by a selection structure



(b) A transition in a selection construct is replaced by a repetition structure



(c) A transition in a parallel construct is replaced by a repetition structure

Legen Transitions which are refined or replaced Dummy transitions Clusters Control arcs

Figure 3.4 Illustrations for Theorem 1

Free-choice Petri nets can model both concurrent and conflict situations. It is a class of Petri nets for which strong theoretical results and efficient analysis techniques exist. In the following, we introduce a very useful result.

Theorem 2. *An extended process net resulted from the "structured business process modeling algorithm" can be checked for liveness and boundedness in polynomial time.*

Proof. This is a direct result of Theorem 1, Theorem A1 (i.e., Rank Theorem) and Corollary A1 (see Appendix A or Desel et al 1995). Therefore, to check whether an extended process net of a structured process model is live and bounded or not can be solved in polynomial time by using existing Petri net analysis techniques.

3.4 An Application Example

The CIMOSA *partial reference model* can be used to guide and help enterprises in building their particular enterprise model, which satisfies the enterprise requirement. Then, this *particular model* can be implemented in the enterprise environment. In this chapter, we use a partial model of shop floor domain as an example to illustrate the supply chain process design by using the structured process modeling algorithm.

This partial model is based on surveys from several Chinese manufacturing enterprises (Chen and Deng 1993). According to the production and organization structures in these enterprises, enterprise functions can be classified into three domains: Production Management & Business Strategy, Enterprise Engineering (CAD/CAPP/CAM), and shop floor. This paper is focused on the shop floor domain. In order to manufacture parts satisfying quantity, quality and due time requirements, the shop floor should include the following domain processes: DP1-Shopfloor Planning & Management, DP2-Shopfloor Manufacturing, DP3-Shopfloor Storage &

Transportation, and DP4-Shopfloor Support. The relationships between them are described in Figure 3.5 (EV stands for Event).

The behavior analysis diagram of the shop floor domain process (DP2) is shown in Figure 3.6.

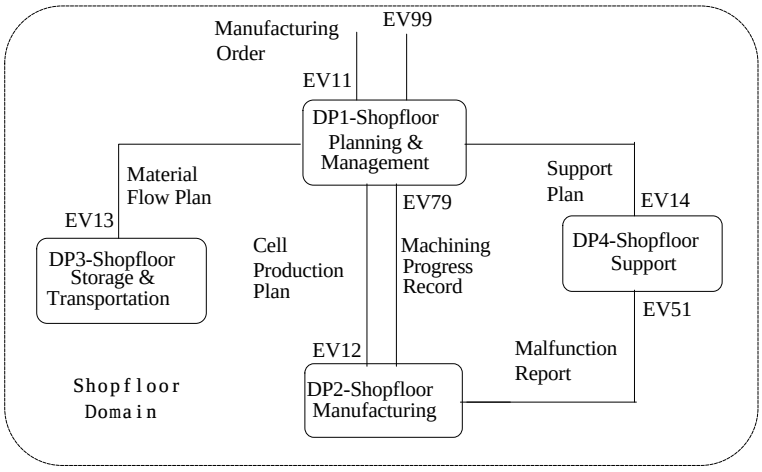
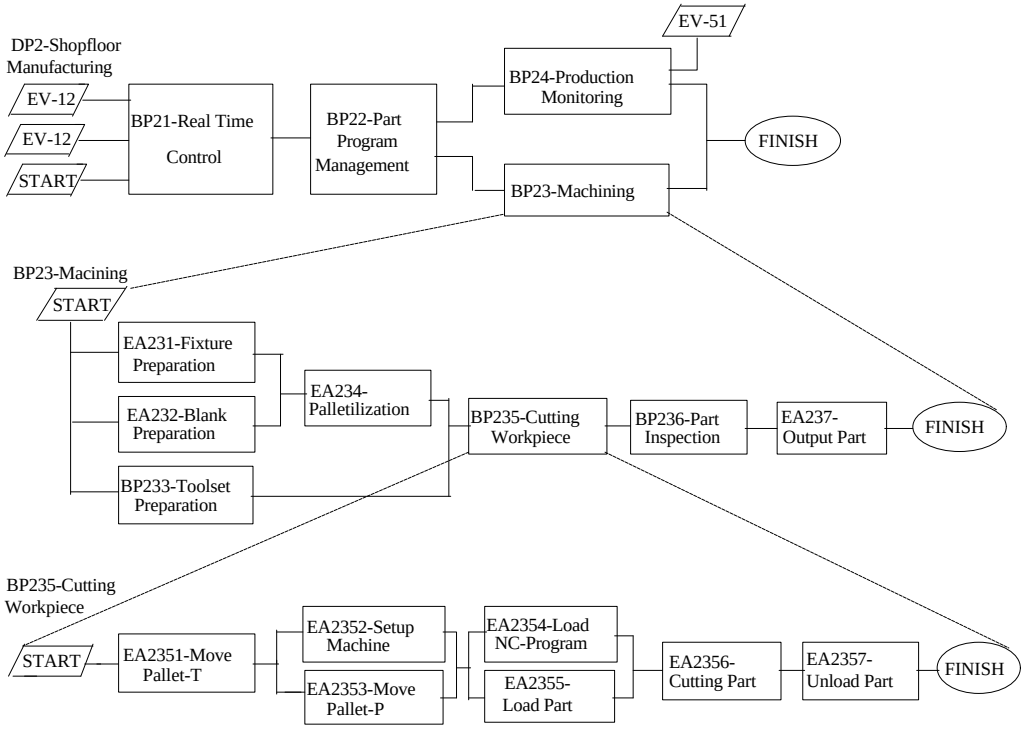


Figure 3.5 The four domain processes in a shop floor domain



Legent: EV-12: Arrival of cell production plan; EV-51: Arrival of malfunction analysis report.

Figure 3.6 Behavior analysis diagram of shop floor domain process (Functional decomposition hierarchy)

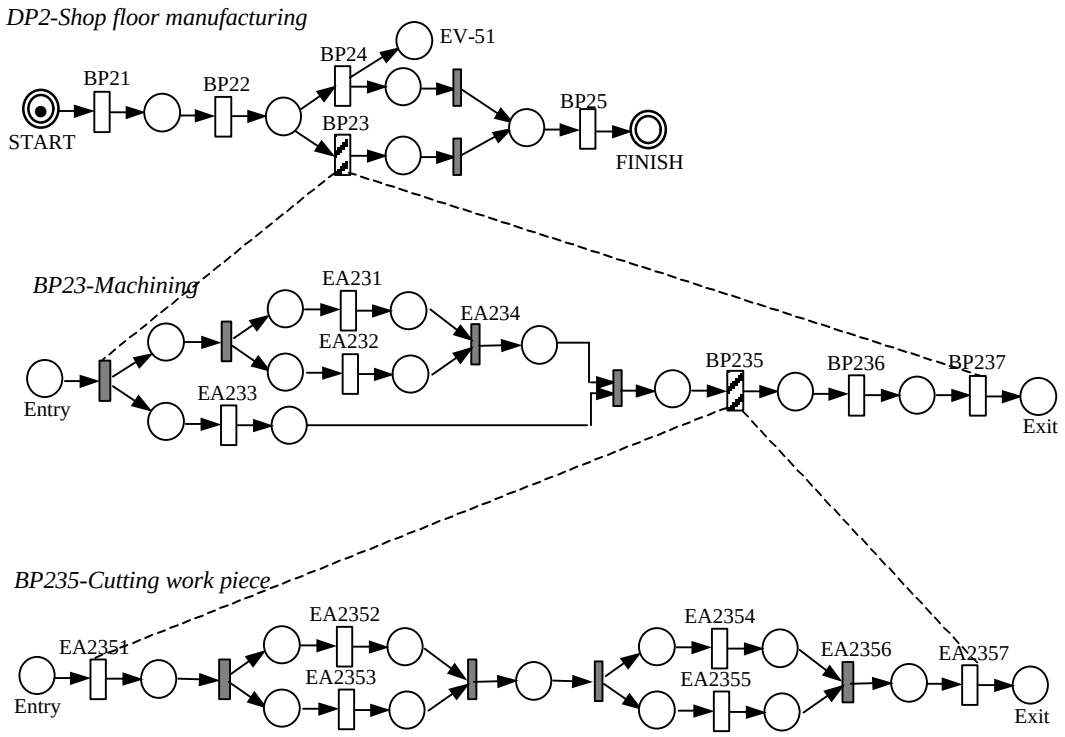


Figure 3.7 Structured process model development procedure for a shop floor manufacturing domain process

The modeling algorithm in Section 3 can be used to design a structured business process model for this application.

Step 1. Use six well-behaved control structures and the chaining rule to build the domain process “DP2-Shop floor manufacturing” (see Figure 3.7).

Step 2. Use four well-behaved control structures (sequence, selection, parallel and repetition structures), the chaining rule and the modified nesting rule to build the business process “BP23-Machining” (see Figure 3.7).

Step 3. Use four well-behaved control structures (sequence, selection, parallel and repetition structures) and the chaining rule to build the business process “BP235-Cutting work piece” (see Figure 3.7).

Step 4. Use “BP23-Machining” net to refine the transition BP23 in “DP2-Shop floor manufacturing” by means of the modified nesting rule.

Step 5. Use “BP235-Cutting work piece” net to refine the transition BP235 in “BP23-Machining” by means of the modified nesting rule.

Because this shop floor domain process is developed by the structured process modeling algorithm, the resulting net is a free-choice process-net and its well-behaved properties can be checked in polynomial time. For this shop floor domain process, it is easy to see its extended Petri net is live and bounded. In addition, from this example we can find the advantages of employing the principles of modularity and top-down design, the process model designed by the structured process modeling algorithm is easy to understand, and easy to modify.

3.5 Free-choice Process-nets with Event Extension

In supply chain business processes, some enterprise functions may create new events. For example, in Figure 3.6, business process “BP24-Porduction monitoring” will produce an event “EV-51.” In the corresponding free-choice Petri net (see Figure 3.7), an event place is added as one of output places of transition BP24. We call this kind of net as free-choice Process-nets with event extension. For this event extension, the following theorem exists.

Theorem 3. *A free-choice process-net with event extension remains as a free-choice process-net.*

Proof. First, the event place is one of output places of the corresponding transition. In other words, this event place is connected to a transition of a free-choice process-net. Therefore, the net with extension of event places is still strongly connected, and is still a process-net.

Second, an event place does not have any output transition, so it will not be included in any clusters of the original net. In fact, this place itself will form a cluster. Thus, the net with event extension is still a free-choice Petri net.

3.6 Summary

Current BPR and workflow tools do not provide an effective approach to design a supply chain business process. In this chapter, a structured process modeling algorithm is presented. This algorithm employs six well-behaved control structures, chaining rule and modified nesting rule to develop structured process models in a top-down way. Some non-value-adding activities are avoided by using these rules. Live and bounded properties are very important for well-formed process models, but they are very difficult to verify because of time and space complexity. The structured process models can be checked for these logic properties in polynomial time and have the advantages of modularity, readability and maintainability. An industrial application case is used to illustrate the usefulness of the proposed method.

Chapter 4

Verification Analysis and Reengineering of Supply Chain Workflows Using Object-Oriented Petri Nets

In previous chapter, the structured supply chain process design is addressed. However, for the quantitative and qualitative workflow analysis, other elements in supply chain processes such as resources and activities are also need to be considered. And except the structured processes, the semi-structured and ill-structured processes are also need to be modeled.

This chapter is organized as follows. The basic elements of supply chain workflows are given in Section 1. Section 2 gives a description of an example of manufacturing supply chain network. Section 3 includes the definition of OPTNs, the top-level and object-level process modeling, the approaches for calculating the system P-invariants from object models, and the structural analysis by P-invariants. Sequencing analysis of supply chain process models by net unfolding is presented in Section 4. Conclusions are given in Section 5.

4.1 Basics of Supply Chain Workflows

Two main concepts of supply chain workflows can be derived from the term "workflow": "work" and "flow." Work such as order and product has to be performed, besides, there is some flow of work. A place where work is performed must be identified. It is called *work unit*. Thus, work flows between work units, i.e., work is exchanged between work units (Jablonski 2000). Examples of work units in a supply chain are retail stores, where customers' order work is performed, or supplier sites, where manufacturers' order work is performed, or shop floors, where manufacturing work is performed. Since work is flowing between work units, there are connectors between them, called *channels*.

4.1.1 Components of Supply Chain Work Units

Observing the real world examples, one can find that a supply chain work unit consists of three components: *people* who perform work at a work unit, *tools* that are needed to carry out work and *operations* that are executed at a work unit. Thus, operations form logical units of execution, tools are deployed within operations in order to provide the required functionality.

4.1.2 Components of Supply Chain Workflow Channels

Consider a sale office within a supply chain, an incoming order (through Internet, Email, Fax or Telephone) indicates a order work for the office clerk. For an manufacturing site within the supply chain, delivered pieces that have to be assembled indicate an manufacturing work. In an abstract view, orders and pieces are nothing but information (or data) and material. That is, information and material flow along channels, i.e., *information channels* and *material channels* are introduced. Things that arrive at a work unit are *work indicators*. Combined with the people who have to carry out work, the tools they will use and operations that must be performed, these work indicators become work.

Further consider the following example: an assembly part is delivered which was painted in the work unit before. To become dry, this part must lie for at least two days untouched. Thus, further information must flow along the channels to indicate this temporal constraint. This situation can be interpreted as if a control token is flowing between the work units, i.e., a *control channel* is defined. Usually, control tokens bear temporal and causal information.

The well-known concept of events also flows along the control channel since events are nothing but control tokens that indicate, mostly, that some piece of work can be worked upon.

In principle, the following rule holds: a work unit can start to perform if all incoming channels are set. This means that data has arrived, material is provided and control flow (e.g., in form of an incoming control event) permits the work unit to start.

4.2 A Manufacturing Supply Chain Case

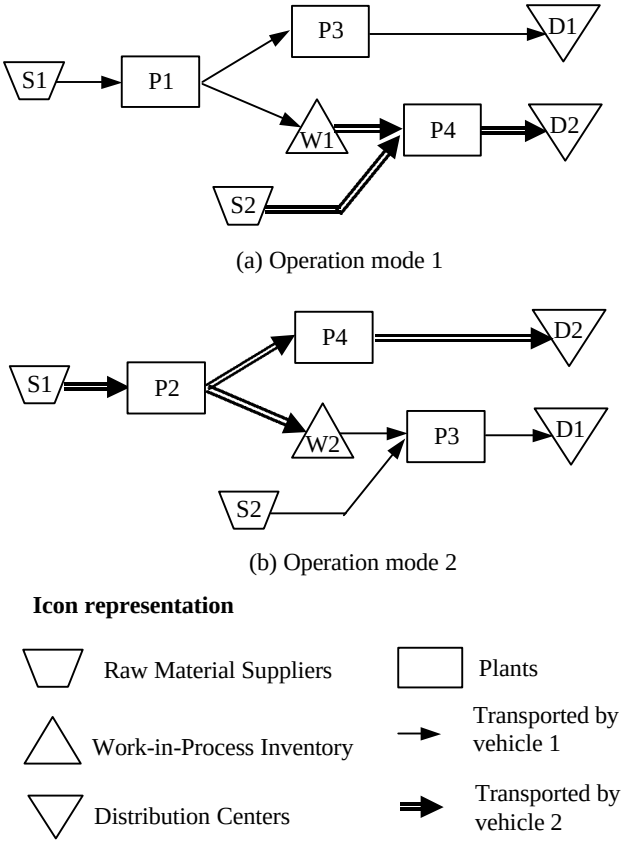


Figure 4.1 An manufacturing supply chain case

In this chapter, an manufacturing supply chain network (shown in Figure 4.1) is used as a case to illustrate the effectiveness of a new OPTN (Object-oriented Predicate/Transition Nets) based modeling and analysis approach. As shown in Figure 4.1, the system consists of two raw material suppliers (*S1* and *S2*), four plants (*P1*, *P2*, *P3* and *P4*), two work-in-inventories (*W1* and *W2*), and two distribution centers (*D1* and *D2*). According to the production capacity, locations and facility features of plants, the manufacturing supply chain can operate under two modes: mode 1 and mode 2. For this supply chain, a fleet consisting of two vehicles is responsible the transportations of material flows between sites in the manufacturing supply chain. This manufacturing supply chain can supply two types of products (product line 1 and product line 2) from raw materials supplied by *S1* and *S2*, respectively. Under operating mode 1, since plant 1 is

closed to plant 3, we keep zero inventory between them (therefore, it is JIT production strategy). The processing of product 2 needs the facilities in plant 3 or plant 4. Because supplier 2 is close to the plant 4 and distribution center 2, we choose plant 4 and distribution center 2 as the supply chain nodes of product line 2. Operating mode 2 has a similar interpretation as operating mode 1.

4.3 OPTN (Object Predicate/Transition Net) Model

4.3.1 Top-level Model and Object-level Model

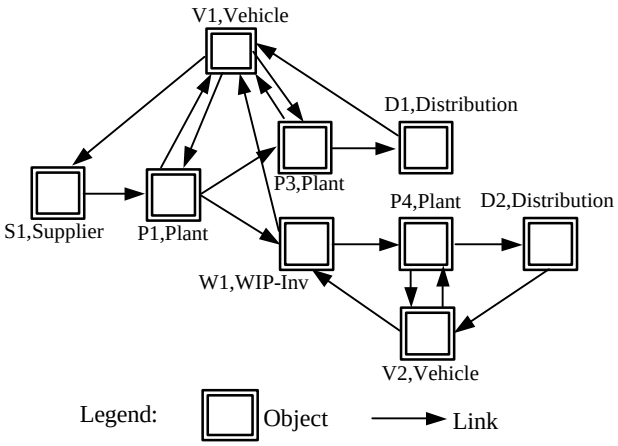


Figure 4.2 Top-level object model for product line 1 under mode 1

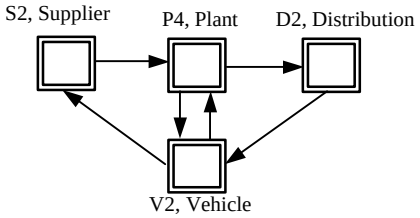


Figure 4.3 Top-level object model for product line 2 under mode 1

From an object-oriented perspective, a system is composed of a number of related objects, each object has its own behavior and attributes. The communication among objects is achieved by message passing. In terms of process routing, the top-level object models for supplying products 1 and 2 are given in Figures 4.2 and 4.3, respectively.

1. *Structure of OPTNs:* Each *OPTN* consists of two parts: internal structure and external structure. The internal structure comprises state places and activity transitions. The functionality

of the *OPTN* is achieved by the sequences of activity transitions. Treating each object as a single server queue, each object can only perform one function at a time. So no concurrency can be included in each object. In other words, the transitions in an object are sequential. However, an object may have alternative sequences of transitions (e.g., multi-functional facilities). So alternative routings of transition sequences inside object are possible. The external structure comprises of a set of ports (places used to receive and send messages), which form the interface of the *OPTN*.

2. *Relationship of input ports and output ports inside OPTNs*: In each object, input ports are used to enter input parameters and output ports are used to show results. Relations of input ports and output ports can be used to synchronize objects when they are synthesized. Let the sequence of transitions of object *A* be $S=\{t_1, t_2, \ldots, t_n\}$. For single-function objects, an approach to establish functional relations between input ports and output ports can be described as follows.

Step 1. Find the first transition t in the S;

Step 2. For t, find $IP_t = {}^{\bullet}t \cap IP$ and $OP_t = t^{\bullet} \cap OP$ (IP_t and OP_t denote the input ports and output ports of t, respectively);

Step 3. Remove t from S;

Step 4. Repeat Step 1 to Step 3 until all transitions are removed from S.

Step 5. For the construction of relations of $t_2, \ldots, t_n$, the input ports, transitions and places that proceed them are added.

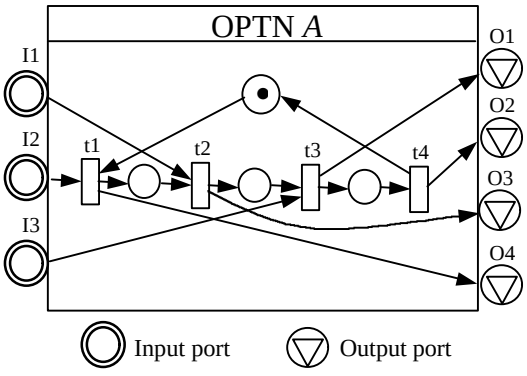


Figure 4.4 Input ports and output ports of OPTN A

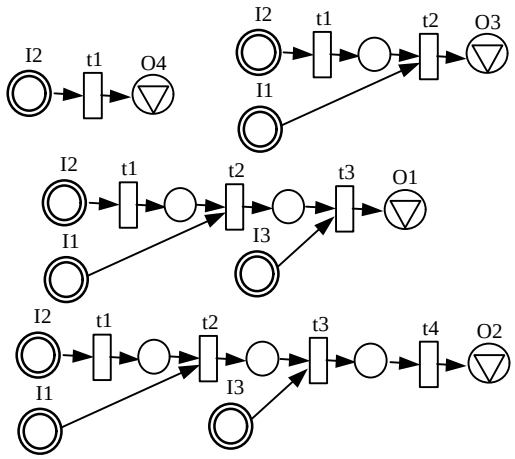


Figure 4.5 Relations of input and output ports of OPTN A

For example, in Figure 4.4, $IP_{t1}=\{I_2\}$, $OP_{t1}=\{O_4\}$, $IP_{t2}=\{I_1\}$, $OP_{t2}=\{O_3\}$, $IP_{t3}=\{I_3\}$, $OP_{t3}=\{O_1\}$, $IP_{t4}=f$, $OP_{t4}=\{O_2\}$. The corresponding input ports and output ports relationships are shown in Figure 4.5. For multi-function objects, alternative transition sequences are used to describe different functions. Predicate models will be established to decide which routing to choose. Then the same algorithm is employed to establish functional relations between input ports and output ports.

3. *Interactions among object-level models and structural analysis by P-invariants*: Objects interact each other through message passing. The connections among objects are based on the specification of scheduling and control. Consider the manufacturing supply chain described in Figure 4.2, the process routing for product 1 specifies the connection of objects as follows: if the materials for product 1 are ordered from supplier 1, vehicle V1 transports them to plant P1 for the first operation of product 1. After the first operation is finished, if the facility in plant P3 is idle, the semi-products will be transported from P1 to P3 by V1. If the facility P3 is busy (or the capacity of plant P3 is full), the semi-products will be transported from P1 to WIP inventory W1. For components in W1, if P4 has some available capacity, the components may be picked up and transported to P4 by V2 to finish their second operation. After all operations of a

product 1 are finished, it can be transported to either distribution center D1 by V1 or distribution center D2 by V2. This process routing can be used to guide the connection of objects.

4.3.2 Verification Analysis of Supply Chain Workflow Processes

For large and complex Petri net models, analysis of them becomes very difficult, if not impossible. However, for each object, its analysis is much easier because it has much smaller number of places and transitions, and much simple structures (no concurrency inside objects). The difficult point of modular analysis is how to obtain the system properties from objects' properties. Lee and Park (1993) use reachability tree to analyze the precedence relation of gates (transitions for message passing) and construct an *interface equivalent* (IE) net, then analysis is made by synchronizing the gates of combined objects. This approach can only detect the deadlock among gates. In Wang (1996a and 1996b), a deadlock analysis by P-invariants is presented. The object communication net (OCNet) is constructed in which except the object being studied, other related objects are represented by hiding the internal actions. Although the calculation effort of P-invariants is reduced since only interfaces are included for other objects, the computation burden is still large because the OCNet becomes large when the number of objects increases. Another shortcoming of this method is: once system configuration changes, these calculations become useless. In this research, the P-invariants for each object are calculated. As long as the structure of an object does not change, the corresponding P-invariants can be reused. To obtain the P-invariants of the entire system that consists of interacted objects, a Petri net synthesis technique presented by Narahari and Viswanadham (1985) is adopted, see Appendix A. Unlike the gates are used to connect the ports of communicating objects (Lee and Park 1993, Wang 1996a and 1996b), the place fusion approach is used. The message sending ports of an object will be merged with the message receiving ports of the corresponding objects. Since the internal structure is used to describe the local actions of objects, it is possible to define objects' internal behaviors such that they do not have common actions. In Petri net language, if $T_1 \in OPTN_1$ and $T_2 \in OPTN_2$, then $T_1 \cap T_2 = \phi$. To obtain the system's P-invariants, the interaction mechanisms among objects are classified. For two objects interaction, the above theorem can be applied directly. For the communications of more than

two objects, three situations are categorized: (1) sequential interaction, (2) *one-to-many* interaction, and (3) *many-to-one* interaction (see Figure 4.6).

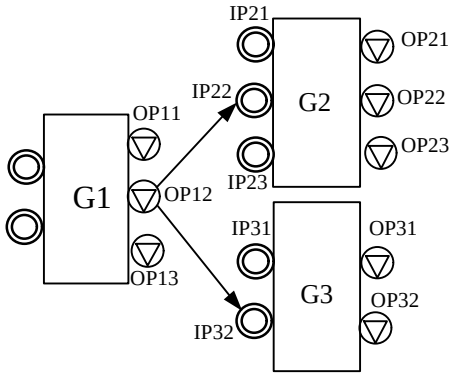


Figure 4.6 “1 : many” object interaction

The procedure of performing the structural analysis for the entire process model from *OPTN* modules may be summarized as follows:

Step 1. *Identify the involved activities in the process model;*

Step 2. *For each activity, identify the objects that can perform it (e.g., according to “Functional Operations ´ Functional Entities” matrix);*

Step 3. *Build the top-level Petri net models;*

Step 4. *For each function entity, construct its OPTN model;*

Step 5. *For each OPTN sub-model, calculate its P-invariants;*

Step 6. *According to the top-level Petri net model and object interaction mechanisms, refine the connections among OPTN modules by merging the corresponding output ports and input ports.*

Step 7. *Apply synthesis theorem to obtain the P-invariants of the whole process model;*

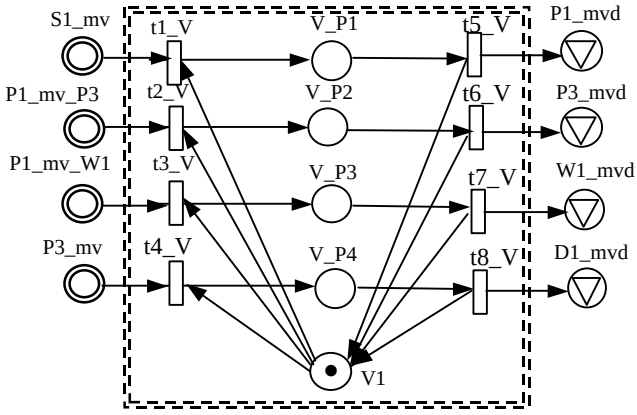
Step 8. *Make deadlock and boundedness analysis by P-invariants.*

In the following, the supply chain process for product line 1 is used to illustrate the verification analysis procedure of process models. Table 4.1 gives the process operating modes for product line 1 (mode 2 will be considered). Table 4.2 shows the “*Functional Operations ´ Functional Entities* ” matrix. Figures 4.7, 4.8 and 4.9 give the *OPTN* models for *V1*, *P1* and *P3*. The P-invariants of vehicle V1, P1 and P3 are calculated and given in Figure 4.10. In terms of process routing for product 1,

output port *P1_mvd* of V1 and input port *Pro1_S1* of P1 are fused together. Similarly, *Pro1_P3_IN* and *P1_mvd* are merged, *P3_mvd* and *Pro1_P1* are merged, and so on. By applying Theorem 1, the P-invariants of this process model (combined by S1, P1, V1, P3 and D1) can be obtained.

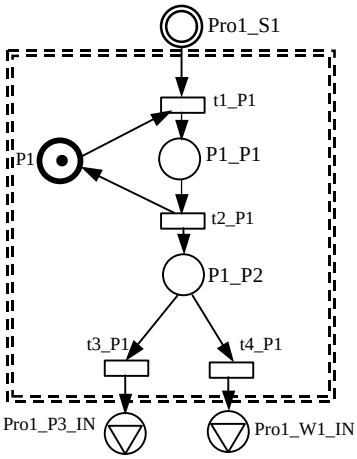
Table 4.1 Routing structure for product line 1

Stage Sequence	**Product Type 1**	
	Mode 1	**Mode 2**
1	Supplier 1	Supplier 2
2	Plant 1 (by V1)	Plant 2 (by V2)
3	Plant 3 (by V1) or (Inventory 1 by V1, Plant 4 by V2)	Plant 4 (by V2) or (Inventory 2 by V2, Plant 3 by V1)
4	Distribution 1 (by V1) or Distribution 2 (by V2)	Distribution 2 (by V2) or Distribution 1 (by V1)



t1_V, t4_V: V1 transports parts from S1 and P3, respectively;
t2_V, t3_V: V1 transports parts and moves them to P3 and W1, respectively;
t5_V, t6_V, t7_V and t8_V: V1 moves parts into P1, P3, W1 and D1, respectively;
V_P1, V_P2, V_P3 and V_P4: parts are being transported;
S1_mv, P3_mv: parts in S1 and P3 are ready for moving, respectively;
P1_mv_P3: parts in P1 are ready for being moved to P3;
P1_mv_W1: parts in P1 are ready for being moved to W1;
P1_mvd: parts have been moved into P1;
P3_mvd: parts have been moved into P3;
W1_mvd: parts have been moved into W1;
D1_mvd: parts have been moved into D1;
V1: V1 is available.

Figure 4.7 OPTN for vehicle V1



t1_P1: P1 starts processing product 1;
t2_P1: P1 finishes processing product 1;
t3_P1, t4_P1: parts start moving to P3 and W1, respectively;
P1_P1: product 1 is being processed;
P1_P2: product 1 is ready going out;
Pro1_S1: product 1 in S1 is ready for its first operation;
Pro1_P3_IN: product 1 is ready to go to P3;
Pro1_W1_IN: product 1 is ready to enter into W1;
P1: facility in P1 is available.

Figure 4.8 OPTN for P1

Table 4.2 (Functional Operations × Functional Entities) matrix

	Operations (days needed for a batch of products)									
Objects	O11	O21	O22	T(S1,P1)	T(P1,P3)	T(P1,W1)	T(P3,D1)	T(W1,P4)	T(P4,D2)	S(Pro1, Pro2)
Plant 1		[15, 17]								
Plant 2		[19, 20]								
Plant 3	[23, 26]		[30, 32]							
Plant 4	[20, 24]		[29, 31]							
Vehicle 1				[2,3]	[3,4]	[1.5,2.5]	[2,3]			
Vehicle 2								[2,4]	[2,3]	
Inventory 1										8
Inventory 2										8

By the theory of invariants, we have $x^T m = x^T m_0$ (x is an ($n \times 1$) vector and m_0 is the initial marking) for all marking m that is reachable from m_0. From the resulted p-invariants, the following equations can be obtained:

P_1: m(V1) + m(V1_p1) + m(V1_p2) + m(V1_p4)+m(V1_p3) = 1

P_2: m(P1_p1) + m(P1) = 1

P_3: m(S1_mv)+m(P1_mv_P3)+m(P3_mvd)+

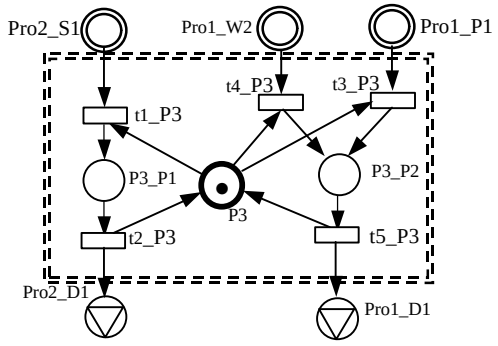
m(W1_mvd)+m(P3_mv)+m(D1_mvd)+m(P1_mv_Pro1_S1)+m(P1_p1)+m(S1_IN)+

m(Pro1_W2_Out)+m(P3_p2)+m(W1_p1)+m(W1_P4)+m(D1)+m(V1_p1)+m(V1_p2)

+m(V1_p4)+m(P1_p2)+m(P1_mv_W1) + m(V1_p3) = 0

P_4: m(P1_mv_P3)+m(P3_mvd)+m(P3_p2)+m(P3)+ m(P3_p1)+m(V1_p2) = 1

P_5: m(W1_mvd)+m(W1_p1)+m(W1)+m(P1_mv_W1) +m(V1_p3) = 8



t1_P3: P3 starts processing product 2;
t2_P3: P3 finishes processing product 2;
t3_P3, t4_P3: P3 starts processing product 1;
t5_P3: P3 finishes processing product 1;
P3_P1: product 2 is being processed;
P3_P2: product 1 is being processed;
Pro2_S1: product 2 from S1 is ready to be processed;
Pro2_D1: product 2 is ready to go to D1;
Pro1_W2: product 1 from W2 is ready to be processed;
Pro1_P1: product 1 from P1 is ready to be processed;
Pro1_D1: product 1 is ready to go to D1;
P3: facility in P3 is available.

Figure 4.9 OPTN for P3

P-invariants of V1:

S1_mv	1 0 0 0 0
V1	0 1 0 0 0
P1_mvd	1 0 0 0 0
P1_mv_P3	0 0 1 0 0
P3_mvd	0 0 1 0 0
W1_mvd	0 0 0 0 1
P3_mvd	0 0 0 1 0
D1_mvd	0 0 0 1 0
V_p1	1 1 0 0 0
V_p2	0 1 1 0 0
V_p4	0 1 0 1 0
P1_mv_W1	0 0 0 0 1
V_p3	0 1 0 0 1

P-invariants of P1:

Pro1_S1	1 0
P1_p1	1 1
P1_p2	1 0
P1	0 1
Pro1_P3_IN	1 0
Pro1_W1_IN	1 0

P-invariants of P3:

Pro1_P1	1 0 0
Pro1_W2	1 0 0
P3_p2	1 0 1
Pro1_D1	1 0 0
Pro2_S1	0 1 0
P3_p1	0 1 1
Pro2_D1	0 1 0
P3	0 0 1

Figure 4.10 P-invariants for V1, P1 and P3

The deadlock analysis is given as follows:

t1_V: If the pool of product 1 raw materials is not empty (S1_IN), then m(S1_mv) is not empty, and if m(V1) = 1 (P_1), then t1_V1 is enabled.

t2_V: If m(S1_IN) ≠ 0, and if m(P1) =1 (P_2), and if m(V1) =1 (P_1), and if m(P3) = 1 (P_4), then t2_V1 is enabled.

t3_V: If m(S1_IN) ≠ 0, and if m(P1) =1 (P_3), and if m(V1) =1 (P_1), and if m(W1) >1 (P_5), then t3_V1 is enabled.

t4_V: If m(S1_IN) ≠ 0, and if m(P1) =1 (P_2), and if m(V1) =1 (P_1), and if m(P3) = 1 (P_4), then t4_V is enabled.

t3_P1 or *t4_P1*:

If the pool of product 1 raw materials is not empty (S1_IN), then m(S1_mv) is not empty, and if m(V1) = 1 (P_1), then t3_P1 or t4_P1 is enabled.

t1_P3: If the pool of product 2 raw materials is not empty, i.e., m(Pro2_S1) ≠ 0, and if m(P3)=1 (P_4), then t1_P3 is enabled.

t3_P3: If m(S1_IN) ≠ 0, and if m(P1) =1 (P_2), and if m(V1) =1 (P_1), and if m(P3) = 1 (P_4), then t3_P3 is enabled.

t4_P3: If the WIP inventory 2 is not empty, i.e., m(Pro1_W2_Out) ≠ 0, and if m(P3)=1 (P_4), then t4_P3 is enabled.

From the above analysis, all transitions in this process model are enabled, so no deadlock is found in the Petri net model. Boundedness is another important property for supply chain process models, it guarantees that no overflow occurs in the system. A Petri net is structurally bounded if and only if there exists an ($n \times 1$) vector x of positive integers such that $x^T A \leq 0$ (A is incidence matrix). Such a positive vector can be easily found from the minimal P-invariants. For example, by adding

minimal P-invariants together, the positive P-invariant [1 2 2 2 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 2 3 2 1 2 3] can be obtained for this process model. Therefore, the process model is bounded.

4.3.3 System P-Invariants with Changeable Object Structures

Some objects may change their internal behaviors and/or their external interfaces. Here, the changing of interfaces means that some input ports and/or output ports will be removed or added to the changed object. Let OPTN G_j be the changed object, and P_j={IP_j, OP_j, SP_j} (P_j - place set of G_j, IP_j - input port set, OP_j -output port set, SP_j - state place set). Similarly, let G_i be the object that sends messages to G_j, and P_i={IP_i, OP_i, SP_i}. Suppose the following message passing relations are held (without loss of generality):

$$OP_i = \{OP_{i1}, OP_{i2}, OP_{i3}\},$$

$$IP_j = \{IP_{j1}, IP_{j2}\},$$

$$OP_{i1} \rightarrow IP_{j1}, OP_{i2} \rightarrow IP_{j1},$$

$$\text{and } OP_{i3} \rightarrow IP_{j2}$$

When input port IP_{j1} is removed from G_j (the corresponding message passing paths are also removed), the minimal P-invariants of G_j are recalculated. Because IP_{j2} still has a message passing relation with G_i through $OP_{i3} \rightarrow IP_{j2}$, from recalculated minimal P-invariants, we choose the one in which the P-invariant of IP_{j2} is equal to the P-invariant of OP_{i3}. Then the chosen P-invariant is used to replace the corresponding part in the system's P-invariants. Similarly, the system's P-invariants can be obtained for other cases: (1) new input ports are added, (2) adding/removal of output ports, and (3) changes with internal state places. In this way, the system's P-invariants with changeable object structures can be readily obtained.

4.4 Sequencing Analysis of Supply Chain Operations by Net Unfolding

The enormous size of the reachability tree is an inherent barrier for applying it to large PNs. Using partial order semantics rather than interleaving semantics, Petri net unfolding can be used to avoid the "state explosion problem." For a review of Petri net unfolding the reader is referred to McMillan (1995) and Taubin et al (1998). The elements in Petri nets (places and

transitions) have three types of relations between them (see Definition A7 in Appendix A for details): preceding (represented as $x_1 \Rightarrow x_2$), conflict ($x_1 \# x_2$) and concurrent ($x_1 \| x_2$) (McMillan 1995). Acyclic PNs are easier to analyze because their properties can be completely specified by these three relations. This suggests that an effective way to analyze a PN: first the cyclic PN is unfolded into an equivalent acyclic representation, and then the finite prefix of the acyclic net is used to analyze the properties of nets. In the following, the analysis of cyclic schedule for the determination of the optimal cycle time is presented.

4.4.1 Process-oriented Modeling of Objects

Some facilities are requested by several activities, resource-sharing is a common situation in process models. Concurrency among objects is another common case in process models. Therefore, for a shared facility, how to sequence its activities so that the potential concurrency between it and other objects can be exploited is critical for scheduling problems.

Suppose that a shared-facility R_s is involved in two sub-processes: SP_1 is the sub-process for supplying product 1 and SP_2 is the sub-process for supplying product 2. To supply product 1, three operations are needed: t_{11}, t_{12} and t_{13}, in which t_{11} and t_{13} are performed by R_s. Similarly, to supply product 2, four operations t_{21}, t_{22}, t_{23} and t_{24} are required, in which t_{21} and t_{23} are performed by R_s. The relations among activities can be described as follows.

Preference relations (by operation sequence constraints): $t_{11} \Rightarrow t_{12} \Rightarrow t_{13}$ and $t_{21} \Rightarrow t_{22} \Rightarrow t_{23} \Rightarrow t_{24}$

Conflict relations (by resource-sharing constraints): $t_{11} \# t_{21}$, $t_{11} \# t_{23}$, $t_{13} \# t_{21}$ and $t_{13} \# t_{23}$.

To obtain the optimal cycle time, the concurrent relations among activities have to be found. Concurrency incurs the interleaving semantics in the reachability graph. To avoid this, the partial order semantics by unfolding is adopted. For the above simple example, the following concurrent relations exist:

$$t_{11} \| t_{22},\ t_{13} \| t_{22},\ t_{21} \| t_{12} \text{ and } t_{23} \| t_{12}.$$

Thus, for complex systems, the potential concurrent relations among activities can be found by the unfolding technique. To achieve this, we first extract the process model of each

object from the entire process model, and then unfold it into an equivalent acyclic net. For the unfolded nets, the cycle time is calculated and the one with the minimal cycle time will be selected. Then the optimal scheduling can be determined. In summary, the following steps will be adopted.

Step 1. *Identify the shared resources.*

Step 2. *Extract sub-process models of shared resources from the whole process model.*

Step 3. *Unfold the process model of each object and calculate the cycle time, choose the unfolding net with minimal make-span. If more than one unfolding nets have the same cycle time, arbitrarily select one.*

Step 4. *Determine the optimal sequence.*

4.4.2 A Case Study for Sequencing Analysis by Ordering Relations

Consider the manufacturing supply chain network described in Figure 4.1, there are product 1, product 2 and product 3 to be produced in this network. The corresponding processing time is given in Table 4.3.

Table 4.3 (Functional Operations × Functional Entities) matrix for different products

	Product 1				**Product 2**				**Product 3**	
Objects	O11 (t1)	O1t (tt)	O12 (t2)	O13(t3)	O21 (t4)	O22 (t5)	O23 (t6)	O24 (t7)	O31 (t8)	O32 (t9)
Plant 1			3							2.5
Plant 2						2.5			1.5	
Plant 3	4.5	1.5			7		2			
Plant 4				3.5				5		

From Table 4.3, four activities will request the facility in plant *P3*, two (*t1* and *tt*) of them is for product 1 and two (*t4* and *t6*) of them are for product 2. Therefore, *P3* is involved in two processes, one is for the supply of product 1 and the other is for the supply of product 2. By extracting these two sub-processes from the whole supply chain process model, the process models for *P3* can be obtained (shown in Figure 4.11). The model in Figure 4.11 is a Petri net with two parallel mutual exclusions and two sequential mutual exclusions. This process model can be unfolded into three acyclic Petri nets. As shown in Figure 4.12, each transition t_i of the

initial PN has a set of corresponding transitions in the unfolding net $t_i^{'}, t_i^{''}, t_i^{'''}, \cdots$, that are called instantiations of t_i. Similarly, for each place p_i the unfolding net contains the set of the corresponding instantiations $p_i^{'}, p_i^{''}, p_i^{'''}, \cdots$. It can be shown that under the partition r that associates each transition t of PN with its instantiations $t^{'}, t^{''}, t^{'''}, \cdots$, for the transition firing sequence, the original PN is equivalent to its unfolding nets.

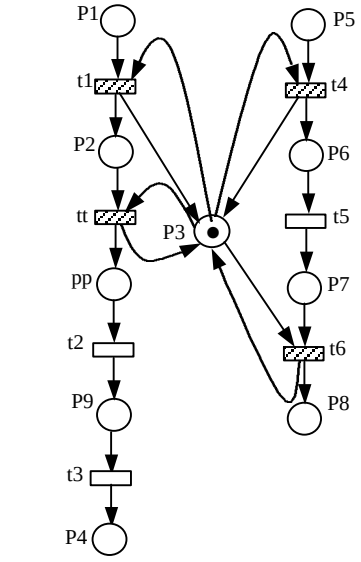


Figure 4.11 Process model of object P3

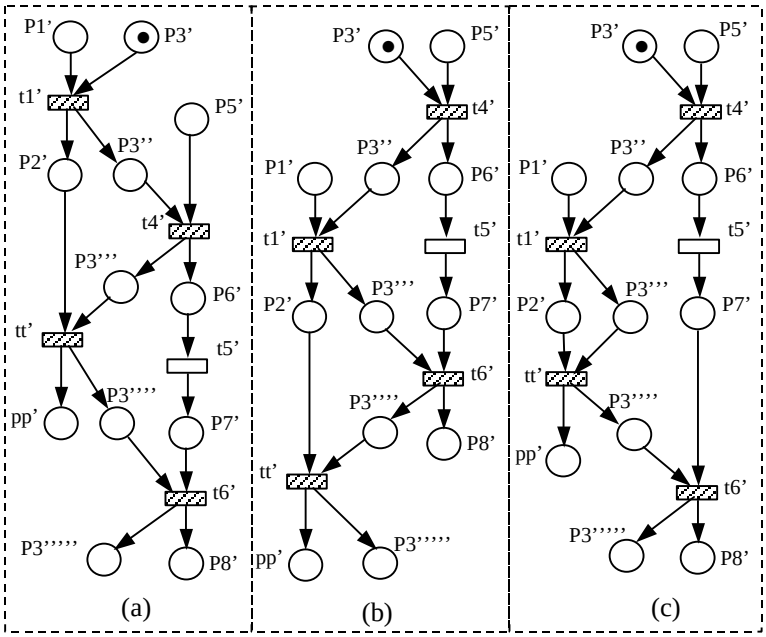


Figure 4.12 Unfolding nets of process model for object P3

Once places p_1 or p_5 has a token, the operations of *P3* can be started. The desired final markings of *P3* are: (P3, P8) or (P3, pp). For marking (P3, P8), there are two ways to achieve it (Figure 4.12a and 4.12c). For marking (P3, pp), there is one way to achieve it (Figure 4.12b). From the unfolding nets (a), the basic marking of local configuration $\{\Rightarrow t_6'\}$ is (P3, P8). From the local configuration $\{\Rightarrow t_6'\}$, the transitions precede t_6' can be easily found. Similarly for other two unfolding nets. From unfolding nets (a), (b) and (c), the following transition firing sequences can be obtained:

(a) $t_1 \rightarrow t_4 \rightarrow (t_t \| t_5) \rightarrow t_6$;

(b) $t_4 \rightarrow (t_1 \| t_5) \rightarrow t_6 \rightarrow t_t$;

(c) $t_4 \rightarrow ((t_1 \Rightarrow t_t) \| t_5) \rightarrow t_6$.

The cycle time for these two nets are calculated as follows:

(a) $CT_1 = 4.5\ (t_1) + 7\ (t_4) + max\{1.5\ (t_t), 2.5\ (t_5)\} + 2\ (t_6) = 4.5 + 7 + 2.5 + 2 = 16$;

(b) $CT_2 = 7\ (t_4) + max\{4.5\ (t_1), 2.5\ (t_5)\} + 2\ (t_6) + 1.5\ (t_t) = 7 + 4.5 + 2 + 1.5 = 15$;

(c) $CT_3 = 7\ (t_4) + max\{(4.5\ (t_1) + 1.5\ (t_t)), 2.5\ (t_5)\} + 2\ (t_6) + 1.5\ (t_t) = 7 + 4.5 + 1.5 + 2 = 15$.

Therefore, unfolding nets (b) and (c) provide shorter cycle time than (a). By comparing the waiting time of place p_7, we have

(b) $WT_1 = 4.5\ (t_1) - 2.5\ (t_5) = 2$;

(c) $WT_2 = 4.5\ (t_1) + 1.5\ (t_t) - 2.5\ (t_5) = 3.5$.

Thus, unfolding net (b) provides shorter waiting time than (c). In the same fashion, other objects P1, P2 and P3 can also be analyzed. Finally, the Gantt chart for this manufacturing supply chain process can be obtained (as shown in Figure 4.13). The minimal cycle time for this supply chain process is 26.5.

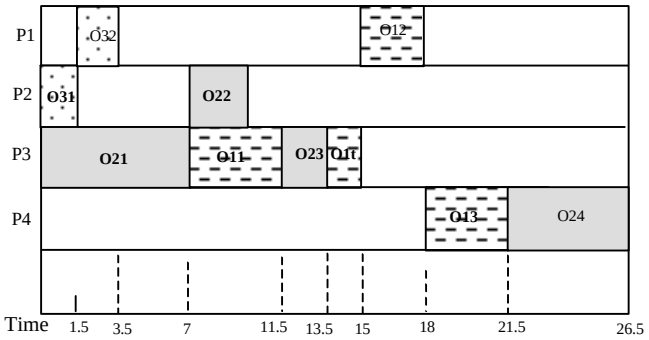


Figure 4.13 Gantt chart for the schedule of supply chain process

4.5 Summary

To facilitate the verification analysis of Petri net-based workflow models, a modular modeling and analysis approach based on object-oriented predicate/transition nets is proposed. First, the top-level object models for the studied supply chain workflows are built. Then, in terms of the structure of object predicate/transition nets, object-level modeling and an approach to find the relations between input and output ports of objects are given. A procedure to obtain

the system's P-invariants through objects' P-invariants is suggested. From the obtained P-invariants, system structural properties such as deadlock and overflow can be analyzed. Finally, by using Petri net unfolding techniques and extracting the sub-workflow model of each object from the entire supply chain workflow model, the sequencing analysis for operations in the supply chain workflow model can be made. The proposed approach has the advantages of modularity and maintainability. It is suitable for quick-change supply chain workflow models. The verification and sequencing analysis can be easily implemented. Several case studies are used to illustrate the proposed method.