CINVESTAV

Centro de Investigación y de Estudios Avanzados del IPN
Unidad Guadalajara

MODELADO MODULAR Y ADAPTABLE DE SISTEMAS DE GESTION DE FLUJO DE TRABAJO

Tesis que presenta:
RAÚL CAMPOS RODRÍGUEZ

Para obtener el grado de:
MAESTRO EN CIENCIAS

En la especialidad de:
INGENIERÍA ELÉCTRICA

Guadalajara, Jal. Octubre del 2002

CINVESTAV

Centro de Investigación y de Estudios Avanzados del IPN
Unidad Guadalajara

MODULAR AND ADAPTIVE MODELING OF WORKFLOW MANAGEMENT SYSTEMS

A thesis presented by
RAÚL CAMPOS RODRÍGUEZ

To obtain the degree of
MASTER OF SCIENCES

On the subject of
ELECTRICAL ENGINEERING

Guadalajara, Jal. October, 2002

MODELADO MODULAR Y ADAPTABLE DE SISTEMAS DE GESTION DE FLUJO DE TRABAJO

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:

Raúl Campos Rodríguez

Ingeniero en Computación
Universidad de Guadalajara

Becario de CONACYT, expediente No. 157967

Asesor de Tesis:
Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Octubre del 2002

MODULAR AND ADAPTIVE MODELING OF WORKFLOW MANAGEMENT SYSTEMS

Master of Sciences Thesis
In Electrical Engineering

By:

Raúl Campos Rodríguez

Computer Engineer
Universidad de Guadalajara

Grant CONACYT no. 157967

Thesis Advisor:
Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, October, 2002

Agradecimientos:

A mis padres y familiares

A mi asesor y compañeros

Al colegio de profesores del CINVESTAV-GDL.

Al CONACYT.

Becario de CONACYT No. 157967

Contents

Chapter 1

Introduction

Nowadays the workflow concept is used to describe the sequences of operations or activities performed into systems or organizations for obtaining a finished product or for providing a service; furthermore, a workflow specification includes the management of resources during the sequences execution. In general this concept is embedded into most of discrete event systems, namely manufacturing systems, computer supported collaborative systems, and logistic systems.

The term workflow itself has been widely used in this last kind of systems studied in management sciences. Thus the term workflow management systems is a generic term to refer to the systems that automate (or at least aids to) the activities into an hospital or the processes into a bank or an enterprise. The commonly term used to design the activities is business processes and a set of these processes is called workflow processes (WP).

Today, business enterprises must deal with global competition, reduce the cost of doing business, and rapidly develop new services and products. To address these requirements enterprises must constantly reconsider and optimize the way they do business and change their information systems and applications to support evolving business processes. Workflow technology facilitates these by providing methodologies and software to support:

- (i) business process modeling to capture business processes as workflow specifications,
- (ii) business process re-engineering to optimize specified processes, and
- (iii) workflow automation to generate workflow implementations from workflow specifications.

Workflow diagrams are an excellent tool for many types of projects: analyzing scenarios, designing new process flows, documenting procedures for new employee training, documenting standard operating procedures, or to support business process re-engineering. The results or solutions depicted by Workflow diagrams help to determine if automation can enhance or support an activity.

Diagrams, with supporting documentation, are more concise to review than textual documents. Describing complex procedures using text alone can yield very lengthy documents that allow for conflicting interpretations by reviewers.

There exist many formal and informal modeling methods for the diagram construction of workflow process. For example, UML, State Charts, Graphs, Flow Diagrams, Case Activity Diagrams, IDEF3, etc.

Petri Nets (PN) are extensively used in the modeling of workflow process (WP). There exist several advantages for the using of PN in the modeling of WP:

- GRAPHICAL NATURE: the PN formalism, has a well-defined graphical nature, allowing a more easy verification of structural conflicts, etc.
- FORMAL SEMANTICS: this reason for using a PN-based modeling of WP, is the fact that business logic can be represented by a formal but also graphical language. The semantics of the ordinary Petri net, and extensions (color, time, hierarchy) have a formal definition.
- FORMAL METHODS FOR THE PROPERTY ANALYSIS: this reason for using PN-based modeling of WP, is that there exist a solid formal theory related to property analysis in PN;
- SEVERAL AUTOMATED TOOLS FOR SIMULATION, SCHEDULING, AND VERIFICATION: there exist, in the PN world, several automated tools for the simulation, scheduling and verification for PN.

Features of a Petri-net-based WFMS are most prominent in the design and analysis phase. Once defined a WP as a PN then, there exist formal methods and techniques for the analysis of properties.

During the selection process some of the leading WFMS are involved a list of generic selection criteria: the standardization, the business processes representation in a natural manner, many WFMS have restrictions with respect to the nesting and/or mixing of parallelism and alternative routing. Moreover, most of the WFMS do not allow for the explicit modeling of states. There exist some Petri-net-based WFMS that meet all of the functional requirements needed. Moreover, formal semantics, state-based instead of event-based, and abundance of analysis techniques, are the mainly features of a WFMS based on Petri nets.

This thesis deals with the modeling of Workflow Management Systems (WFMS) using Petri Nets (PN). The problems regarding the modular construction and the dynamic modification of models in WFMS are addressed.

First, it is presented how to construct Workflow Nets (WFNets), a sub class of Petri Nets for each workflow pattern (WFPatterns) -a set of elementary models for the construction of complex workflow processes-. In this construction of WFNets, the resulting models are sound -that is, they have well properties, such as liveness and boundedness-.

For the construction of sound WFNets, it is proposed an incremental synthesis method that avoid the subsequent entire property analysis.

In order to support the dynamic change in the WFMS, it is proposed the Rewriting Nets (RWNets) -a class of high level Petri Nets- that allow the in-line modification of WFNets models, through the firing of its transitions. A formal definition for the RWNets is presented, and a characterization of them in terms of graph transformations.

Finally, a software application -that allows to perform structural changes over a given WFNet- is developed. This software application -developed over the Petri Net Kernel (PNK)-, allows to implement a set of RWNets that perform structural changes over a given .WFNet.

This thesis is organized as follows: Chapter 1 presents an overview of Workflow Management, and its relevance in business today. Also, it presents some basic concepts and the workflow patterns. Chapter 2 presents a relationship between Petri Nets and Workflow patterns. Then, this chapter defines a Workflow Net for each workflow pattern. A Workflow Net is a special class of Petri Nets. In the end of this chapter, a place-refinement method for Petri Nets is adapted from the manufacturing systems to the particular case of workflow management. Chapter 3 addresses the Dynamic change problem in WFMS. An overview of this problem and the previous solutions proposed are presented. RWNets -an extension for Dynamic Nets- are proposed in order to support such dynamic change. Also, some similar topics between graph grammars and RWNets are highlighted. Chapter 4 presents a practical implementation of RWNets. On it, some "rewriting" transitions are implemented as Java classes over the Petri Net Kernel. Finally, some conclusions and future work are presented.

Chapter 2

Workflow Management Systems

In this chapter, an overview of Workflow Management -an area related to the automation of business processes- and its importance in business today are prcsented. Also, it is presented basic concepts on Workflow Management, such as tasks, conditions, routing, parallel and sequential activities, choices and more, and how to make a business process model mixing these concepts. Finally, Workflow patterns, a class of primitive forms to build complex workflow process models, are presented.

2.1 Basic concepts on Workflow Management

Workflow is defined as the automation of a business process, as a whole or in part, during which, documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules [WFMC-TC-1011]. Workflow Management aims for the automation of business process reducing costs and time, and increasing productivity and quality. Workflow Management Coalition (WfMC) is an organization that has over 300 member organizations worldwide, representing all aspects of workflow, from vendors to users, and from academicians to consultants [http://www.wfmc.org/]. The WfMC has a press room [http://www.wfmc.org/pr/press.htm], where the official WfMC documents are published.

Workflow is an important and a valuable technology. In 1998, Viewstar installation in Revenue & Benefits Department of Lewisham Borough Council (London, England) has impacted their operations in the first year in the following way:

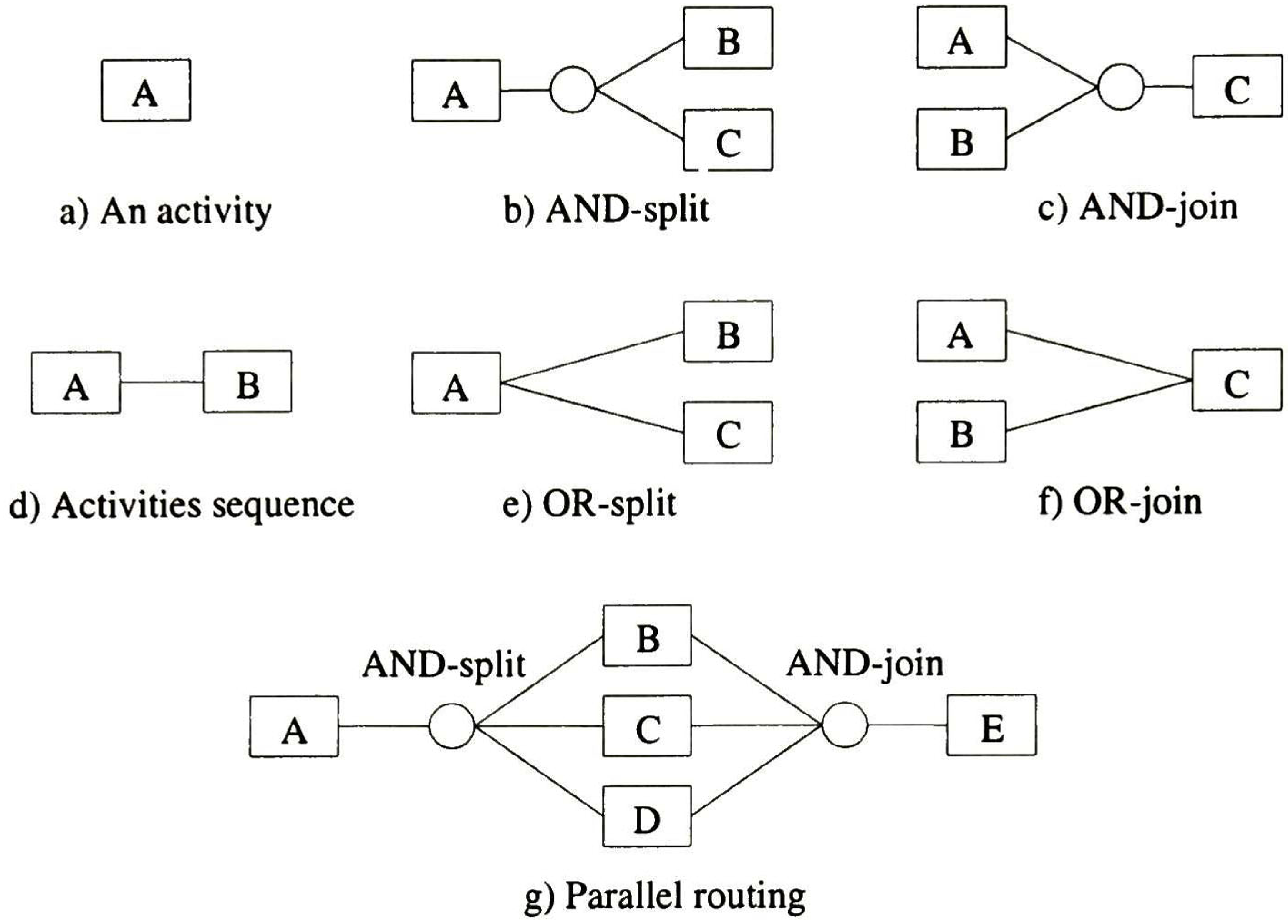
- $5 (euro) million additional revenues
- Frau Investigation saves an additional $1.7 (euro) million through speedier processes and cross checking capability
- $0.5 (euro) million savings on operational costs.

This is an example of how a correctly implemented workflow can impact the operations of an organization [Workflow Handbook, 2001].

In other words, Workflow Management aims for automation of business process with the Min-Max attribute: minimize costs and time, and maximize productivity and quality. Workflow Management uses many concepts presented on Computer Supported Cooperative Work, but it is not limited to them.

When it is talked about business processes, it surges concepts such as tasks (in the rest we will use task and activity indistinctly) , scheduling, parallel and sequential activities, shared resources, routing task, iterations, and so on. The next are some definitions taken from [WFMC-TC-1011]:

- Workflow. The automation of a business process, as a whole or in part, during which, documents, information or tasks are passed from one participant to another in order to perform some action over them, according to a set of procedural rules.
- Workflow Management System. A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, to interact with workflow participants and, when required, to invoke the use of IT tools and applications.
- Activity. A description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or an automated activity. A workflow activity requires human and/or machine resource(s) to support process execution; when a human resource is required, an activity is allocated to a workflow participant.



a) An activity

b) AND-split

c) AND-join

d) Activities sequence

e) OR-split

f) OR-join

g) Parallel routing

Figure 2.1: Workflow Concepts

- Process. A formalized view of a business process, represented as a coordinated (parallel and/or serial) set of process activities that are connected in order to achieve a common goal.

- Parallel Routing. A segment of a process instance under enactment by a workflow management system, where two or more activity instances are executing in parallel within the workflow, giving rise to multiple threads of control.

- AND-Split. A point within the workflow where a single thread of control splits into two or more threads, which are executed in parallel, allowing multiple activities to be executed simultaneously (see Parallel Routing).

- OR-Split. A point within the workflow where a single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches.

- Transition. A point during the execution of a process instance where one activity completes and the thread of control passes to another process instance, which it starts.

The concepts of Activity, parallel routing, AND-Split, and OR-Split are also well defined in Petri Nets. With these concepts and others from [WFMC-TC-1011], it is possible to make a mapping from Workflow concepts into Petri Nets. In Figure 2.1 it is shown some of these concepts. In the next section, it is presented the Workflow Patterns, schemes to construct complex workflow process models.

2.2 Workflow Patterns

In Workflow Management, some primitives called Workflow Patterns were developed for the construction of complex workflow models, for the analysis of tools, for workflow requirements, and so on. Workflow patterns can be found on the web page [http://tmitwww.tm.tue.nl/research/patterns/]. They are divided into 6 basic groups: Basic Control, Advanced Branching and Synchronization, Structural, Multiple Instances, State-based, and Cancellation workflow patterns. Also, in the literature is mentioned the Inter-Workflow Synchronization patterns. Here, it is analyzed these patterns with the aim to represent them using Petri Nets concepts [Petri, 1962].

2.2.1 Basic Control Flow Patterns

Basic Control Flow Patterns capture elementary aspects of process control. Concepts such as Sequence, Parallel split, Synchronization and Exclusive choice, are basic concepts that give form to the more elemental workflow models. These concepts accommodate for further construction of elaborated workflow patterns.

The first pattern is the simple sequence pattern. This pattern allow us to construct a causal sequence of activities, when for example, before the execution of an activity it is necessary that other activity must already be finished.

The Parallel Split pattern allows us to construct a pair of activities, or a sequence of them, executing in parallel. A real example of such pattern, is presented when we need to send a parallel request to a local server and to a remote one.

The Synchronization pattern can be seen as the complement of the Parallel Split pattern. At this point, we wait for multiple subprocesses running in parallel to be finished before the next activity starts.

The Exclusive Choice pattern allows to make a simple decision, i.e., at this point, it is possible to choose between two or more alternatives. On the other hand, the Simple Merge pattern allows us to perform the inverse situation, when it is waited one of many alternatives to be completed.

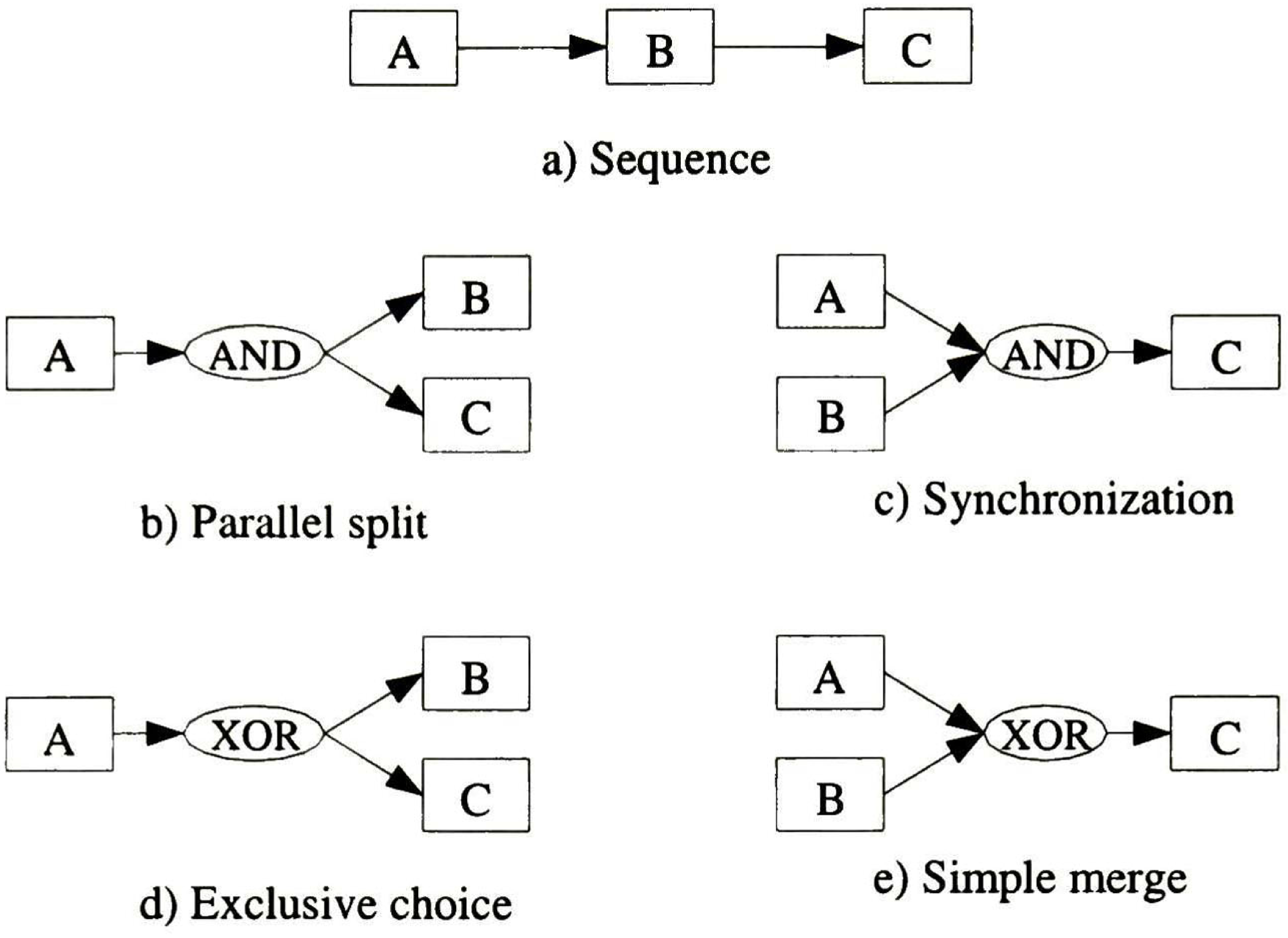
Definition 2.1 (Sequence) *An activity in a workflow process is enabled after the completion of another activity in the same process.*

Definition 2.2 (Parallel Split) *A point in the workflow process where a single thread of control splits into multiple threads of control which can be executed in parallel is called Parallel Split. This allows us to execute all the activities simultaneously or in any order.*

Definition 2.3 (Synchronization) *A point in the workflow process where multiple parallel subprocesses or activities converge into one single thread of control, thus synchronizing multiple threads, is called Synchronization.*

Definition 2.4 (Exclusive choice) *A point in the workflow process where, based on a decision or workflow control data, just one of several branches is chosen is called Exclusive choice.*

Definition 2.5 (Simple Merge) *A point in the workflow process where two or more alternative branches come together without synchronization is called Simple merge. In other words the merge will be triggered after any of the incoming transitions are triggered.*



a) Sequence

b) Parallel split

c) Synchronization

d) Exclusive choice

e) Simple merge

Figure 2.2: Basic Control Flow Patterns

These basic concepts are depicted on Figure 2.2. The simple sequence pattern, Figure 2.2a is isomorphic to the sequence of transitions in Petri Nets and have the same meaning. The parallel split pattern, Figure 2.2b has its isomorphic net into Petri Nets too, and it is a transition with an input place and two output places. The synchronization pattern is the inverse of the parallel split pattern, and it has its equivalent net into Petri Nets, too, Figure 2.2c. Also, exclusive choice and simple merge are complementary and they have also, their equivalent net in Petri Nets, Figure 2.2d and Figure 2.2e.

The next is only a selection of the five groups of workflow patterns. For more information see the Workflow Pattern home page given above.

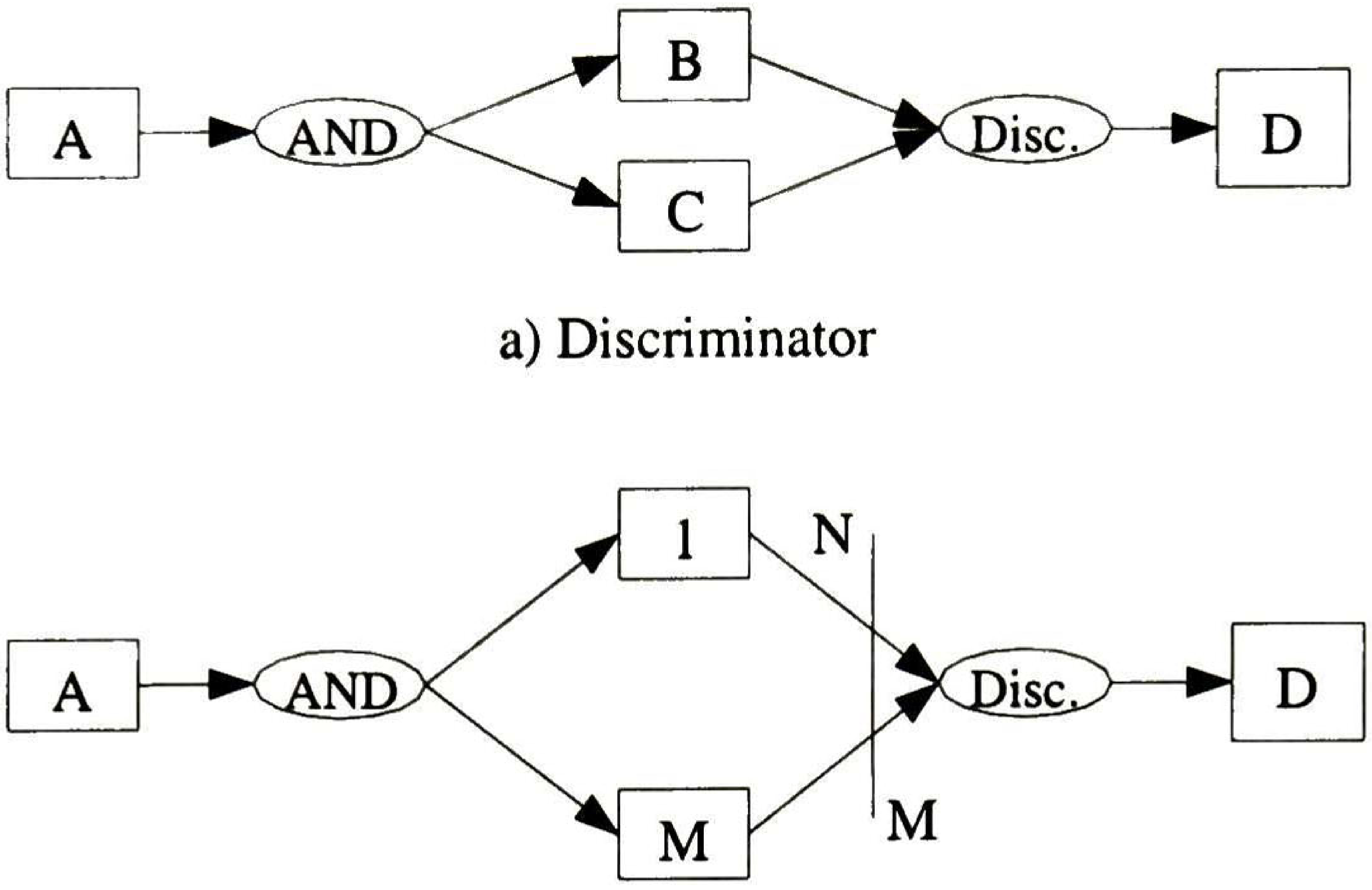
2.2.2 Advanced Branching and Synchronization

Opposite Basic Control Flow Patterns, Advanced Branching and Synchronization patterns have no-straightforward equivalent in Petri Nets. However it is possible to build an equivalent model in Petri Nets using Basic Control Flow Patterns.

These patterns are involved in more complex situations in Workflow Management. For example, a point in a Workflow process where it waits for the completion of one or several activities before activating the next task. An example of a real situation is presented in a distributed search, where many requests are send to different databases, and it is expected only one reply, the remaining replays are ignored.

In this section, Discriminator and N-out-of-M Join patterns are presented. For more, visit the Workflow Patterns home page given below.

Definition 2.6 (Discriminator) *The discriminator is a point in a workflow process that waits for a number of incoming branches (parallel activities) to complete before activating the subsequent*



a) Discriminator

b) N-out-of-M join

Figure 2.3: Advanced Patterns and Synchronization

activity. When one of these activities is completed, all the remaining branches are "ignored" Once all incoming branches have been triggered, it resets itself, so that it can be triggered again.

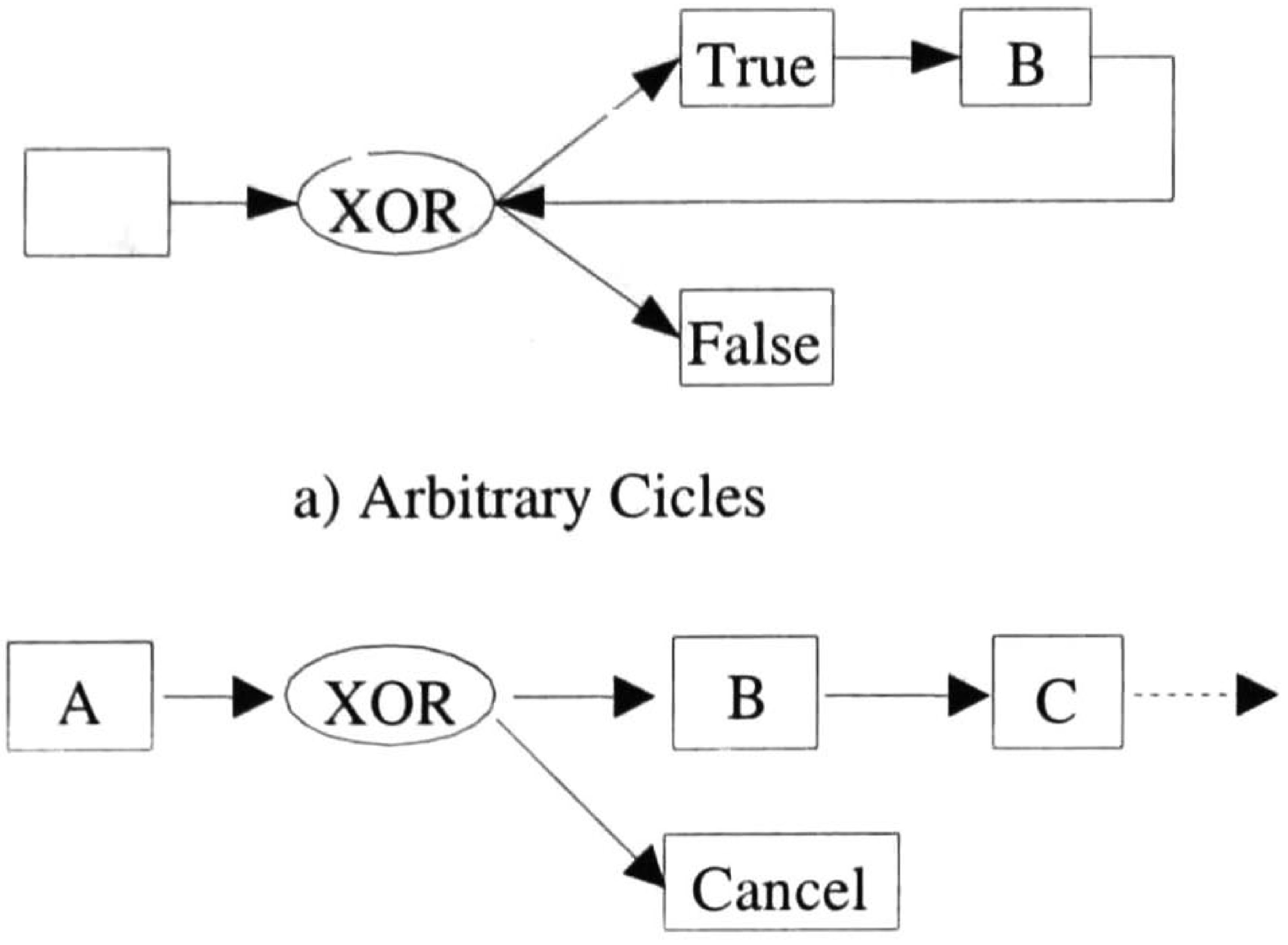
Definition 2.7 (N out of M Join) *N-out-of-M Join is a point in a workflow process where M parallel paths converge into one. The subsequent activity should be activated after N paths have been completed. Completion of all remaining paths should be ignored. Similarly to the discriminator, once all incoming branches have been "fired", the join resets itself, so that it could be fired again.*

Discriminator pattern "waits" for the completion of a certain number of incoming branches, then it enables the next activity while it ″waits" for the remaining branches. When all of these incoming branches are done, it goes to its final state. N out of M Join pattern could be understood as a generalization of the Discriminator pattern. In this pattern, n of m incoming branches are expected to be done before the enabling of the next activity, once again, the pattern waits for all the remaining branches and then goes to its final state. There is a graphic representation of this concepts on Figure 2.3.

2.2.3 Structural Patterns

Structural patterns are a special class of structures that can help us in some situations. For example, the well known sentences WHILE-DO and GOTO are two special instructions in many programming languages, that provide special control flow behavior. In Workflow Management, there exists some elaborated control flow patterns. For example, in a Workflow Management System where is not allowed a great-scale parallel execution of a certain task, it can be achieved using a WHILE-DO cycle executing into it the maximum number of allowed parallel activities by the system.

Definition 2.8 (Arbitrary Cycles) *A point in a workflow process where one or more activities can be done repeatedly.*



a) Arbitrary Cicles

b) Implicit termination

Figure 2.4: Arbitrary Cycle and Implicit Termination Patterns

Definition 2.9 (Implicit Termination) *A given subprocess should be terminated when there is nothing else to be done. In other words, there are no active activities in the workflow and no other activity can become active (and at the same time the workflow is not in deadlock).*

Arbitrary Cycle pattern has a straightforward representation in Petri Nets. Implicit Termination pattern often is achieved transforming the original model into a model with only one termination node.

The patterns on this section are depicted in Figure 2.4.

2.2.4 Multiple Instances Patterns

This class of pattern is related to multiple instances of a certain task running in parallel. This pattern by its own has no problem, but there are some variants of it: Multiple Instances with a priori-known design-time knowledge, Multiple Instances with a priori Runtime Knowledge, Multiple Instances with no a priori runtime knowledge, and Multiple Instances requiring synchronization. The first case is when it is known, at design time, the number of copies of the task to be performed. The second case is when the number of copies of the task is known at some stage in runtime. The third case is when the number of copies of the task is known only after the execution of such task. Finally, the last pattern is related to the complement of the three others, i.e., its synchronization -a point in the workflow process, where these parallel activities are expected-.

In this work, it is explored only the third case, the Multiple Instances with no a priori runtime knowledge pattern, and its synchronization. All of the other variants are a particular case of this last one.

Definition 2.10 (MI with no a priori runtime knowledge and Synchronization) *For one case an activity is enabled multiple times. The number of instances of a given activity for a certain case is neither known during design time, nor at any stage during runtime, but it has to be*

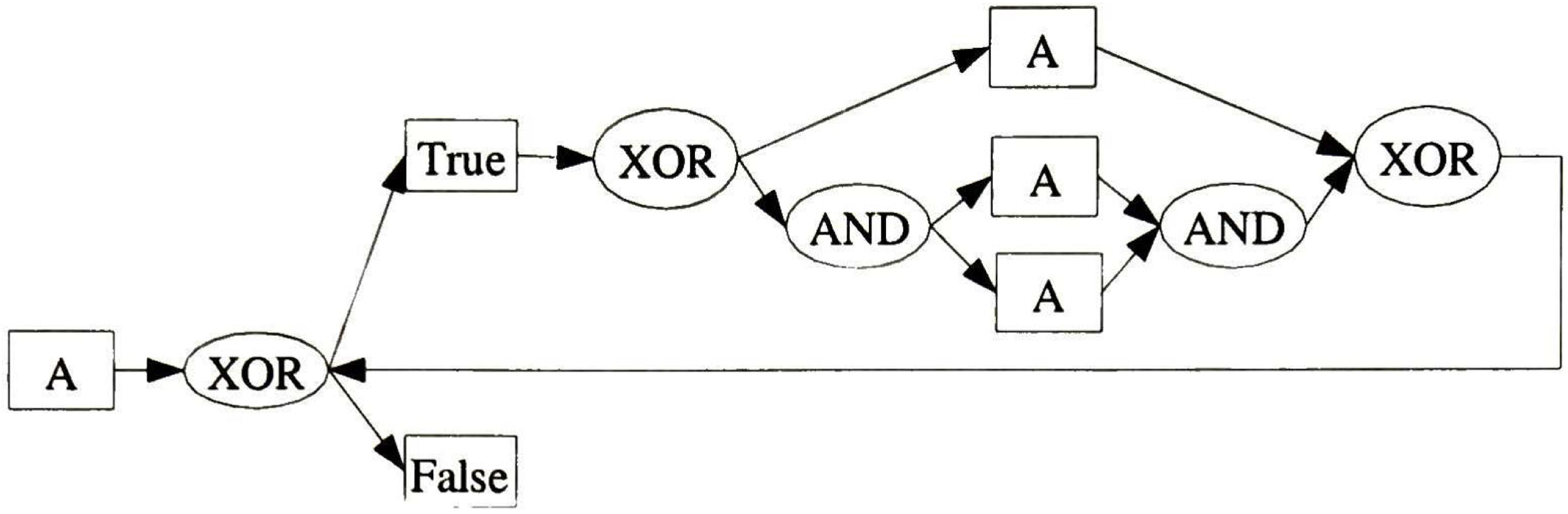


Figure 2.5: Multi Instances Patterns

know before the creation of the activity. The Synchronization waits for the termination of the first executed activity and all its copies running in parallel before the execution of the next task.

This pattern fires one given task A, then it runs in parallel a certain number of copies of the task, then it waits for its termination, and then it gives the control to the next task. These concepts are depicted in Figure 2.5.

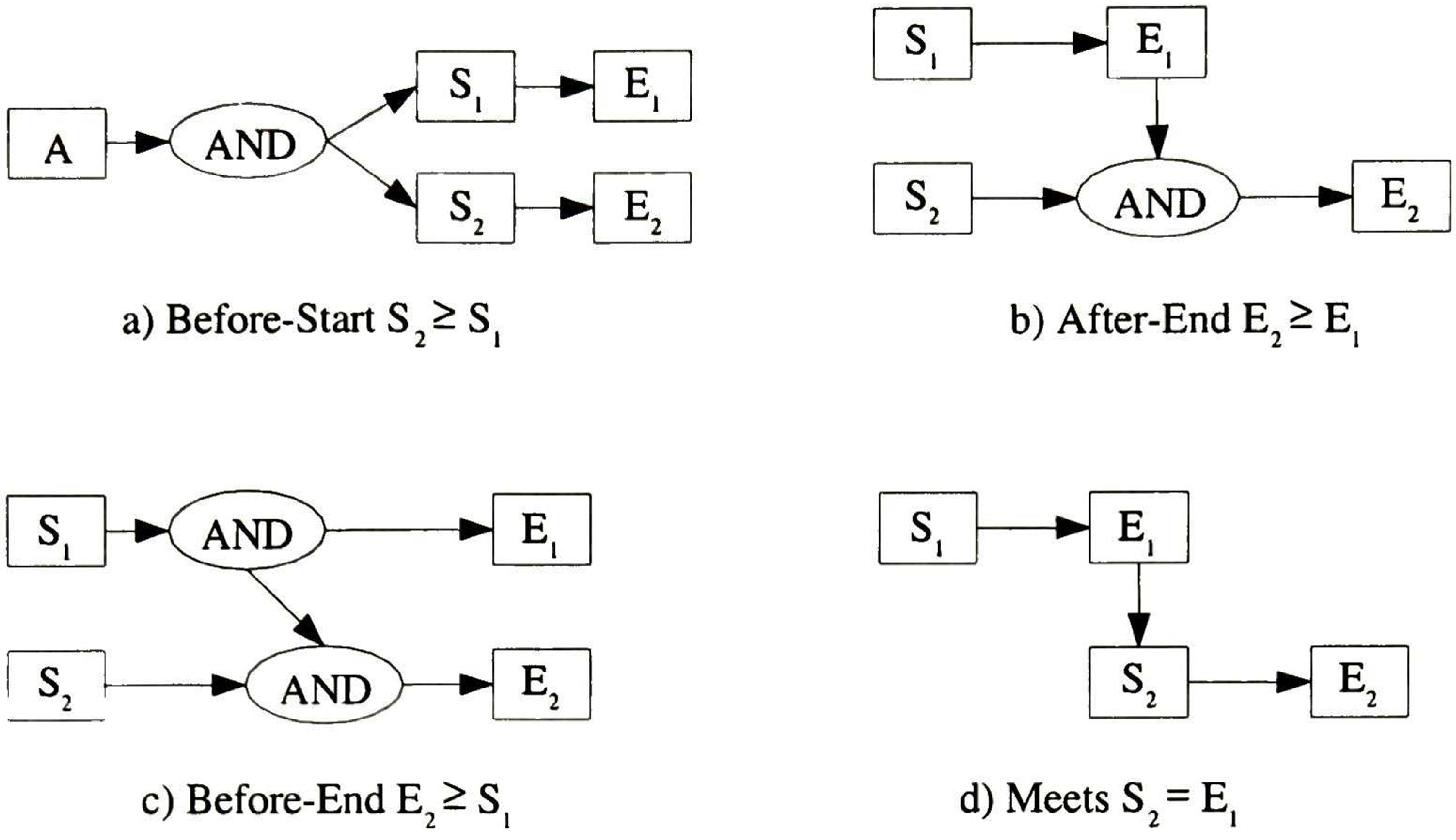
At first, the task A is executed, then in the Arbitrary Cycle pattern, the maximum number of parallel copies allowed by the system are executed repeatedly until achieve the desired number of copies of the task. Its synchronization is the complement case, it waits for the termination of the first execution of the task, also it waits for all remaining tasks, and then it gives the control to the next task.

2.2.5 Temporal Relation Patterns

A Temporal Relation Pattern is formed by a set of tasks temporally related. If s_1, e_1, s_2, and e_2 are the start and end points of *task*_1, and start and end points of *task*_2, respectively, then the temporal relation between *task*_1 and *task*_2 can be condensed as follows: (1) before-start: $(s_1 \geq s_2)$, (2) after-end $(e_2 \geq e_1)$, (3) before-end $(e_2 \geq s_1)$, and meets $(s_2 = e_1)$, where $\geq$ is the temporal relation "more-or-equal to" The next definition is for Temporal Relation patterns.

Definition 2.11 (Interleaved sequence) *Two activities that have another interdependent time relationship than simple sequencing. It might be that one activity has to start before the second activity finishes, or that one activity has to end at exactly the same time as the second activity ends.*

These concepts are represented on Figure 2.6. This pattern can be achieved, in most cases, transforming the model into an equivalent one that meets a particular temporal relationship. For example, if it is possible to divide each task into a task with four stages: start, after-start, before-end, and end, then it is possible to perform some temporal relations as as it is shown in Figures 2.6a, b, c, and d. There, one task can give the control to other task at its before-end stage in order to star the other task, performing in this way, the condition that the second activity starts before the first one ends.



a) Before-Start $S_2 \geq S_1$

b) After-End $E_2 \geq E_1$

c) Before-End $E_2 \geq S_1$

d) Meets $S_2 = E_1$

Figure 2.6: Temporal Relation Patterns

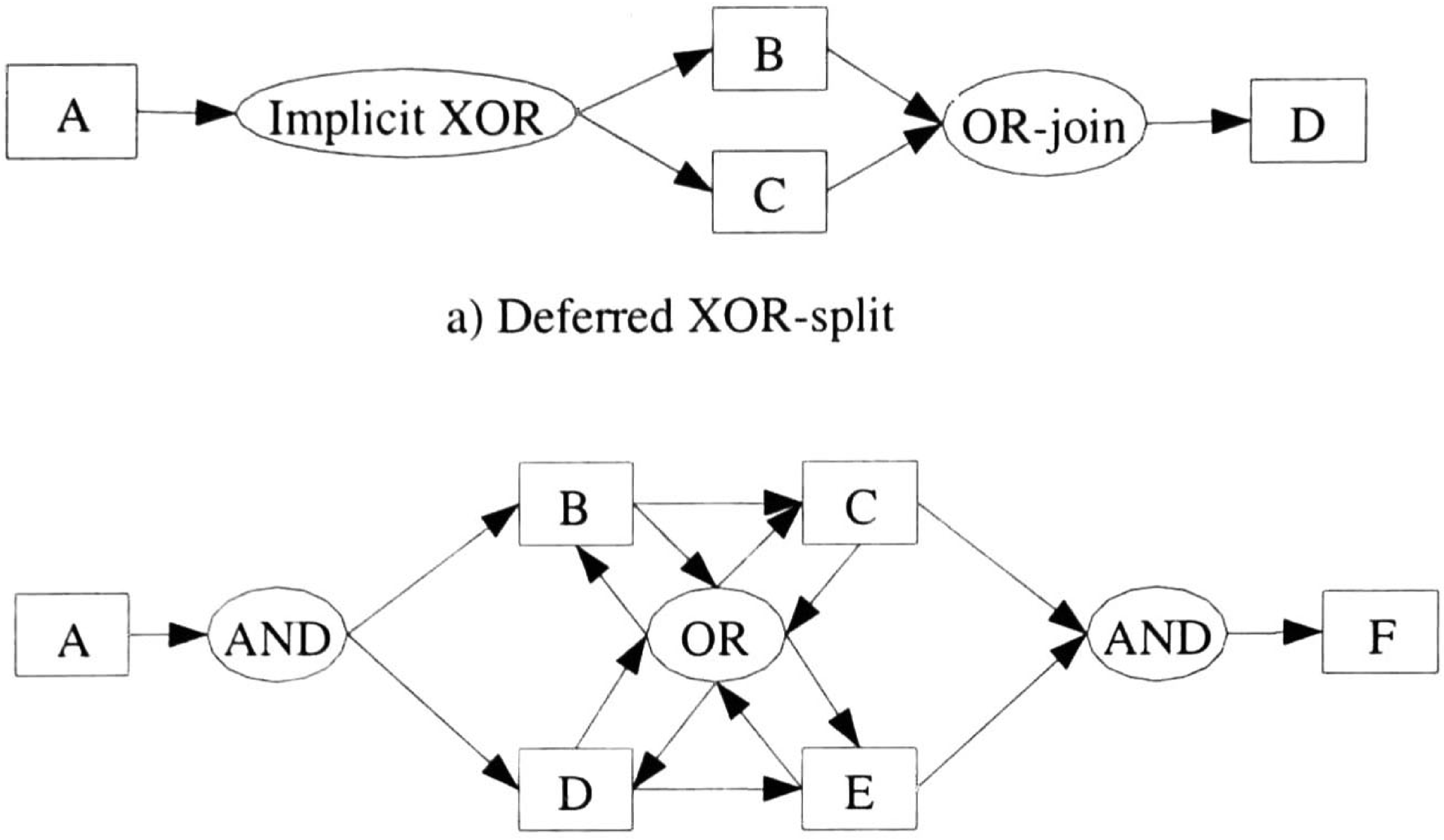
2.2.6 State-based Patterns

Deferred XOR-split pattern is the first pattern explored in this section. In its more simple case, two alternatives are offered to the system, but only one is executed, after that, the other one is withdrawn.

Definition 2.12 (Deferred Choice) *A point in the workflow process where one of several branches is chosen. In contrast to the XOR-split, the choice is not made explicitly (e.g. based on data or a decision) but several alternatives are offered to the environment. However, in contrast to the AND-split, only one of the alternatives is executed. This means that once the environment activates one of the branches, the other ones are withdrawn. It is important to note that the choice is delayed until the processing in one of the alternative branches is actually started, i.e. the choice time is as late as possible.*

Definition 2.13 (Interleaved Parallel Routing) *A set of activities is executed in an arbitrary order: Each activity in the set is executed; the order is decided at run-time and no two activities are executed at the same time (i.e. no two activities are active in the same workflow instance at the same time).*

In State-based Patterns, the selection of an alternative in most of the cases is implicit rather than explicit. In many Workflow Management Systems, it is necessary to offer to the System many alternatives, then one is chosen and the others are withdrawn. This is the case of Deferred Choice pattern. In the Interleaved Parallel Routing pattern, a set of activities is enabled, but only one is executed at each time. Figure 2.7 shows these concepts.



a) Deferred XOR-split

b)Interleaved parallel routing

Figure 2.7: State Based Patterns

2.2.7 Cancellation Patterns

Cancellation Patterns are related to the cancellation of activities and cases currently in execution. A case could be compounded by some activities in sequence or in parallel. So, if it is explored the cancellation of activities, then it is possible in the major of cases, to perform a case that can be cancelled completely, cancelling all of its individual activities. Below, the Cancel Activity pattern and Cancel Case pattern definitions are presented.

Definition 2.14 (Cancel Activity) *An enabled activity is disabled, i.e. a thread waiting for the execution of an activity is removed.*

Definition 2.15 (Cancel Case) *A case, i.e., workflow instance, is removed completely.*

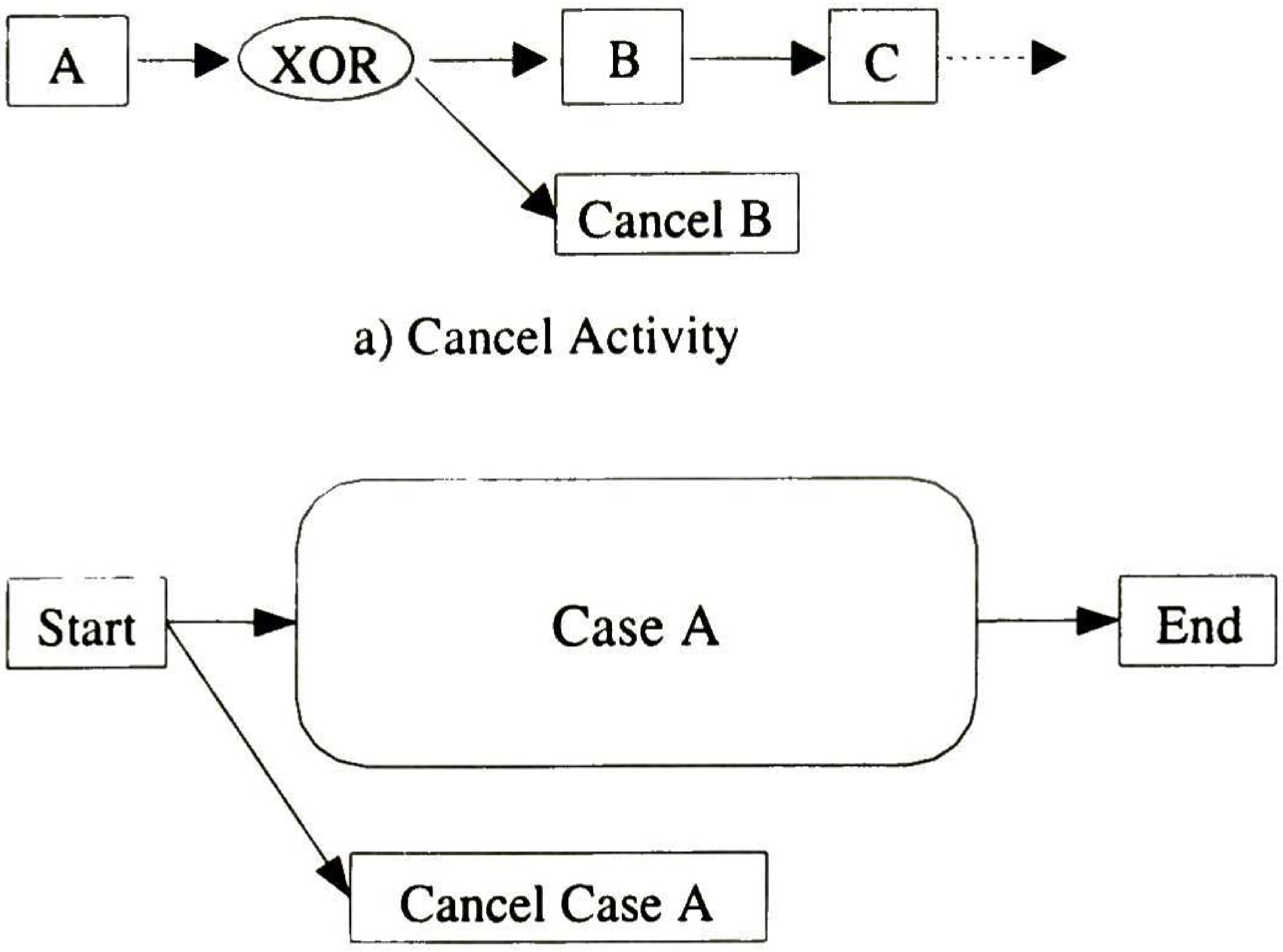
Workflow Management is case based, i.e., there is a Workflow instances for each case in the Workflow Management System. Accordingly to it, when it is used the Cancel Case pattern over a certain workflow instances, it has the same effect as if such instance is removed from the system.

The fact that Workflow Management is case based is explained afterwards, when it is explored the translation of Workflow Management concepts into Petri Nets concepts, in the next chapter.

Figure 2.8a shows the Cancel Activity pattern, and Figure 2.8b shows the Cancel Case pattern.

2.2.8 Inter-Workflow Synchronization patterns

The Inter-Workflow Synchronization patterns deal with the coordination between two independent workflow instances, i.e., the enabling of activities is not more dependable on the workflow in which this activity is present, but it depends on the synchronization between workflows. In this section we present the Message Coordination pattern and the Bulk Message Sending pattern.



a) Cancel Activity

b) Cancel Case

Figure 2.8: Cancellation Patterns

Definition 2.16 (Messaging coordination) *A sender issues a request and at the "sending" end a response is anticipated for that request by a subsequent receiver associated with the sender.*

Definition 2.17 (Bulk message sending) *Multiple instances of message senders of the same type execute concurrently. This allows to capture business situations where notifications of the same type are sent to several external stakeholders. The number of multiple message instances may be known a priori at design time or runtime, or may only be determined during runtime.*

The first pattern in this section, the Messaging coordination pattern, is a simple message protocol "send-wait", i.e., sending a request and waiting a response. In Figure 2.9, a Workflow instance sends a request, on its activity A, to other workflow instance, on its activity J, and then it waits for a response before it activates the task D. On the other hand, it is performed the request-response protocol, i.e., receiving a request and sending a response.

The Bulk message sending pattern deals with the case of one sender and many receivers, i.e., a kind of "message group" or a broadcast to a selected group of elements. In Figure 2.10 is depicted an Internal Workflow instance sending a "bulk message" to many stakeholders.

2.3 Conclusions

In this chapter it was reviewed most concepts on Workflow Management. Also, it was presented the Workflow patterns, a class of primitive schemes that meet many basic concepts of Workflow Management and that they allow us to build more complex schemes for more elaborated situations in Workflow Management. Workflow patterns are classified into different groups that have similar structure and semantic. The next chapter is related to the Workflow modelling using Petri Nets.

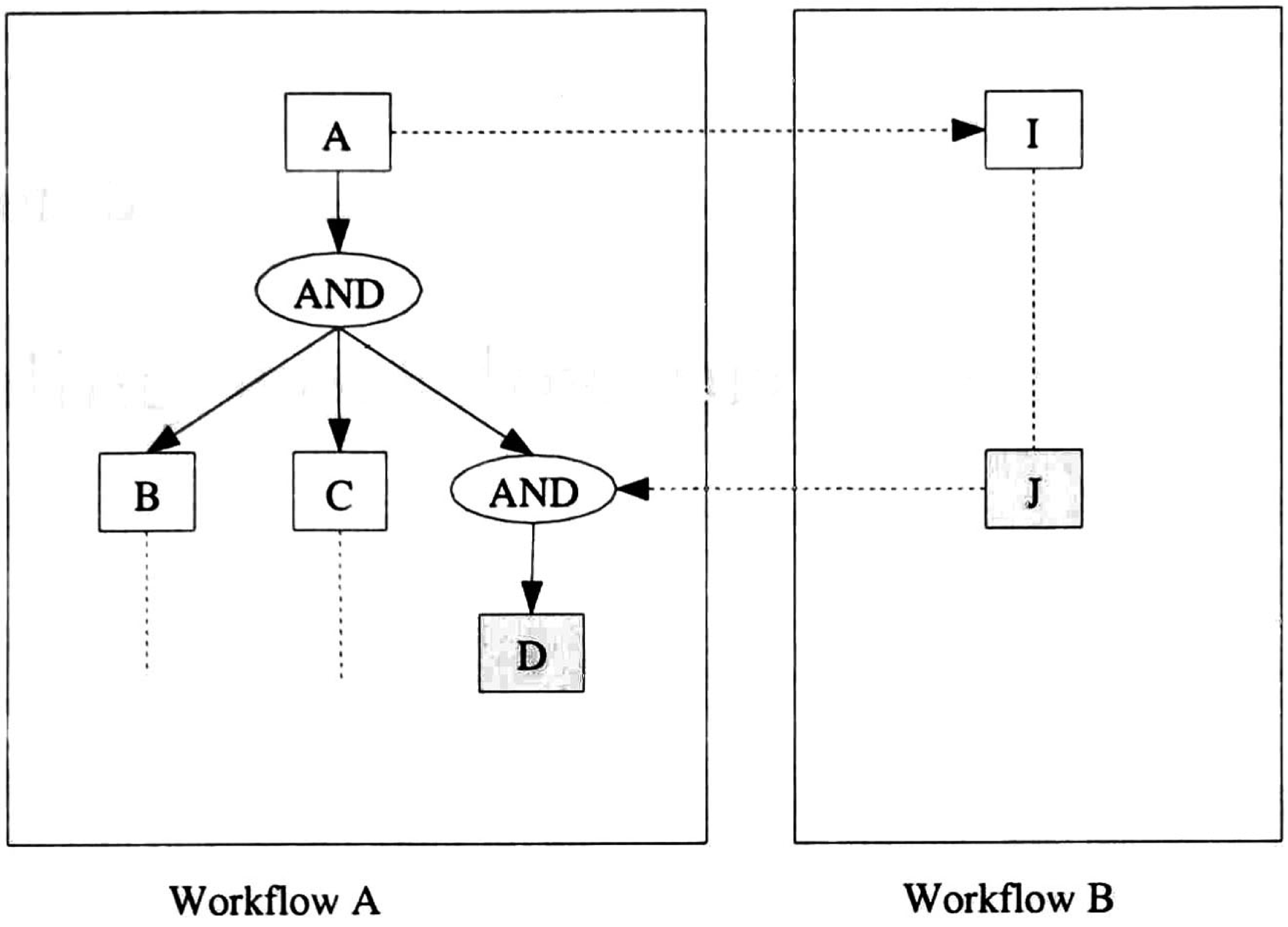


Figure 2.9: Inter Workflow Synchronization Pattern

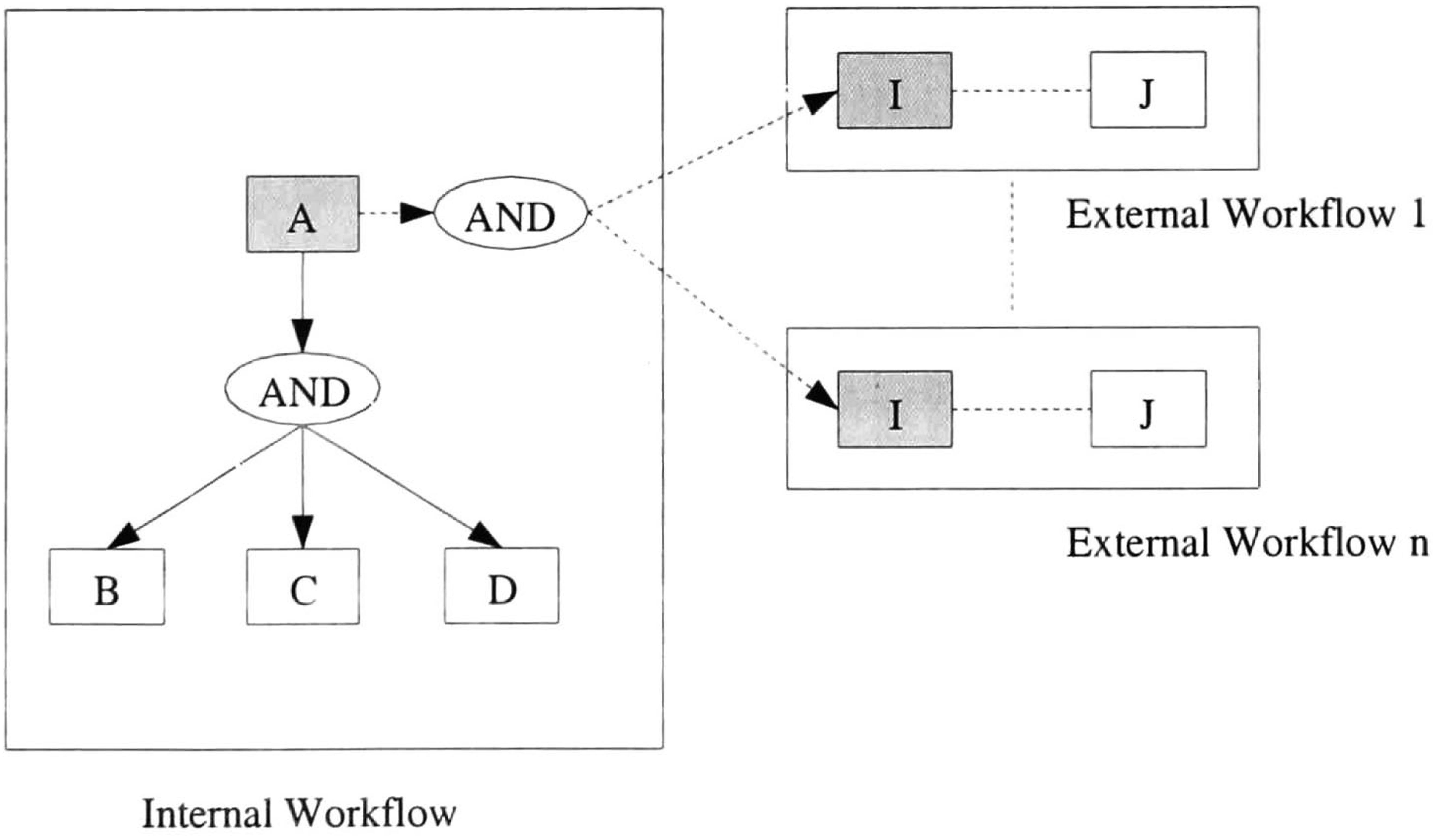


Figure 2.10: Inter Workflow Message Bulk Pattern

Chapter 3

Modeling workflow processes

This chapter presents the relationship between Workflow Management concepts and Petri Net concepts. First, basic concepts on Petri Nets are presented: Petri Net structure, Marking for Petri Nets, Dynamics for Petri Nets and Petri Net properties. Then Workflow patterns are translated into Workflow Nets, a subclass of Petri Nets that meets these patterns. Finally, a place-refinement method for Petri Nets is adapted from the workflow manufacturing systems to the particular case of workflow management.

3.1 Basic Notions on Petri Nets

Petri Nets were first presented by [Petri, 1962]. It is one of the most popular formal models for concurrent systems, used for both theoreticians and practitioners [Desel, 1995]. Petri nets are useful in Workflow modeling, [Aalst, 1997], [Badouel, 1998]. In this section it is presented the basic concepts on Petri Nets and some of their properties. For more see [Lopez, 1997].

3.1.1 Preliminaries

Before presenting Petri Nets, it is given some preliminary notions related to multisets. A multiset over a set P is a function $\mu : P \rightarrow \mathbb{N}$. A Multiset μ is finite iff $\mu(x) \neq \emptyset$ for a finite number of elements $x \in P$. A multiset μ is empty iff $\mu(x) = \emptyset$ for all $x \in P$ The set of finite multiset over P is denoted $\mu(P)$. We write $x \in \mu$ iff $\mu(x) \neq 0$. The sum of two multisets $\mu_0, \mu_1 \in \mu(P)$ is the multiset $\mu = \mu_0 \bigoplus \mu_1$, such that $\mu(x) = \mu_0(x) + \mu_1(x)$. The difference of two multisets $\mu_0, \mu_1 \in \mu(P)$ is the multiset $\mu = \mu_0 \backslash \mu_1$, such that $\mu(x) = \mu_0(x) - \mu_1(x)$, if $\mu_0(x) \geq \mu_1(x)$, 0 in other case. For two multisets $\mu, \mu' \in \mu(P)$, we write $\mu \subseteq \mu'$ iff $\mu(x) \leq \mu'(x)$ for all $x \in P$.

3.1.2 Petri Net structure

A Petri Net is a bipartite directed graph having two classes of nodes: places and transitions. These nodes are bridged by directed edges that always join different kind of nodes, i.e., an edge joins either a place to a transition or a transition to a place. Graphically, the places are represented by circles, transitions by rectangles (or bars), directed edges by arrows. A Petri Net typifies a dynamic behavior aided with "tokens" depicted as small dots into the places. When a place contains a "token" it is said that this place is a "marked" place. For Petri Nets, it will be distinguished two class of places: private and public places.

In the next, Nm will be used as a set of public places, ω as an infinite set of private places, and $Nm_\omega = Nm + \omega$. Also, L will be used as a infinite set of labels, such that $\lambda \in L$. The λ label is called the "invisible" label. The formal definition for Petri Net structure is as follows, and it was taken from [Asperti, 1996].

Definition 3.1 (Generalized Marked Petri Net) *A generalized marked Petri Net (MPTNet), or* ***net system****, is a pair (N, μ_0), where N, called the* ***net structure****, is a bipartite directed graph defined by a 3-tuple $N = (T, \partial_-, \partial_+)$, where T is a finite set of transitions, $\partial_i : T \rightarrow \mu(Nm_\omega), i = -.+$ are the pre and post functions, respectively, and $\mu_0 \in \mu(Nm_\omega)$, is called the* ***initial marking****.*

For a $t \in T$, it is often written $t = c \rhd p$, where $c = \partial_-(t)$, and $p = \partial_+(t)$, or simply, $t = (c, p)$. Commonly, a net is expressed as a list of transitions $t_i = c_i \rhd p_i$ (see the example below). Also $\bullet t$ denotes $\partial_-(t)$, and $t\bullet$ denotes $\partial_+(t)$. Moreover $\bullet t$ is often called the ***input places*** *of t (or the* ***pre set*** *of t), and $t\bullet$ is often called the output places of t (or the* ***post set*** *of t).*

Any $\mu \in \mu(Nm_\omega)$, is called ***a marking****. Also, $Nm_\omega(N)$ denotes the places of Nm_ω used in N, i.e., $\{x \in Nm_\omega | x^n \in (\partial_- \cup \partial_+), n \in \mathbb{N}\}$. For a $x \in Nm_\omega(N)$, $x\bullet$ denotes $\{t \in T | s \in \partial_-(t)\}$, and $\bullet x$ denotes $\{t \in T | s \in \partial_+(t)\}$.*

A $MPTNet$ represents the flow relations between tasks (transitions), and the set of its preconditions (∂_-) and its post conditions (∂_+). A transition in a $MPTNet$ can fire consuming "tokens",

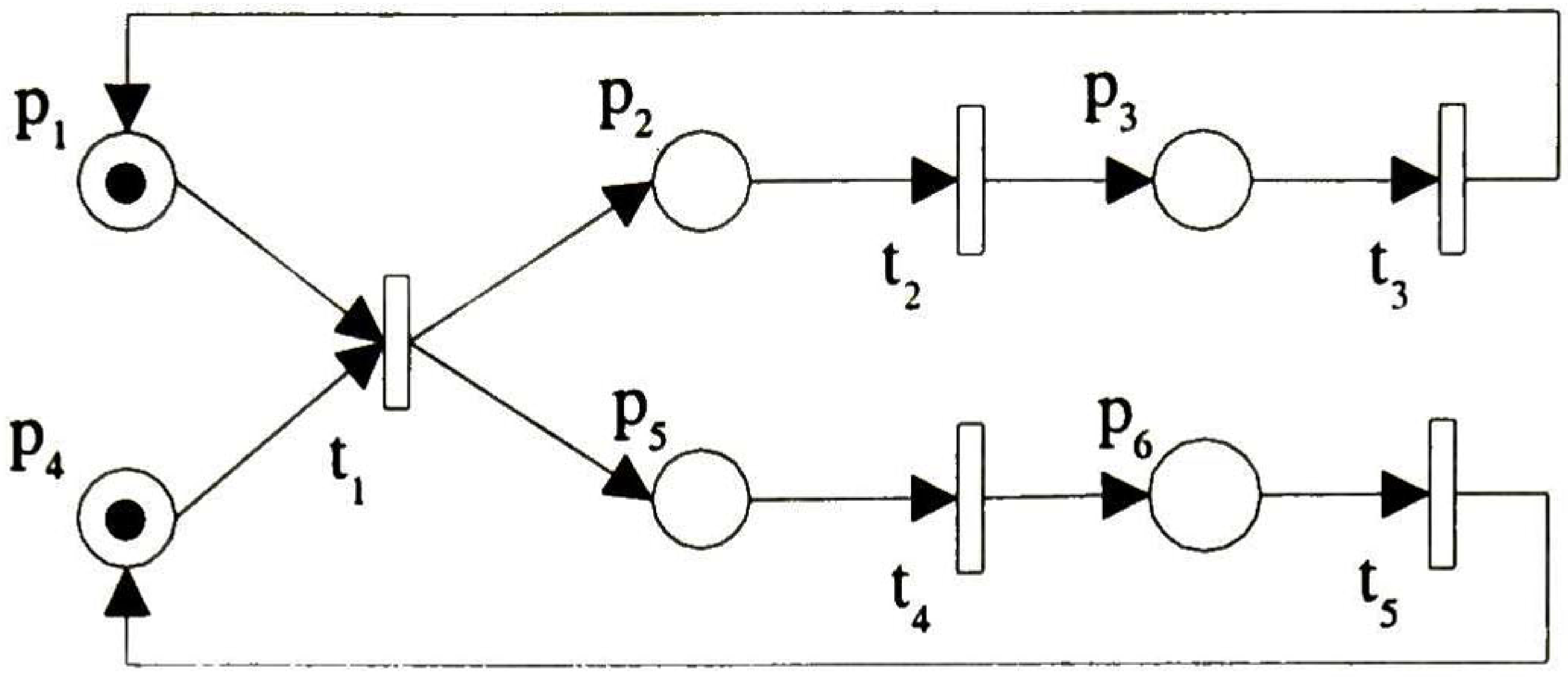


Figure 3.1: A Marked Petri Net

as it is indicated by the pre set function ∂_-, and produces "tokens" as it is indicated by the port set function ∂_+. When a transition fires it is said that the net is in execution. The set of marked places at the beginning of the net execution represent its initial state.

Example 3.1 (A MPTNet) *Consider the next net. Let* $N = (t_1 = \{p_1,p_4\} \rhd \{p_2,p_5\}, t_2 = \{p_2\} \rhd \{p_3\}, t_4 = \{p_5\} \rhd \{p_6\}, t_3 = \{p_3\} \rhd \{p_1\}, t_5 = \{p_6\} \rhd \{p_4\})$, *and* $\mu = \{p_1,p_4\}$. *Then* $(N,\mu) \in MPTNets$ *is the net depicted on Figure 3.1. The net have 6 places:* $p_1,...,p_6$ *(the ovals in the figure) and 5 transitions:* $t_1,...,t_5.$, *(the rectangles). The places have a label above them, that each one represents its name. Also, the transitions have their label name at their basis. When a transition is horizontal, it has its label name at its right side. The arrows from a place to a transition represent the set of pre conditions for such transition. The arrows from a transition to a place represent the post set for such transition. The places* p_1 *and* p_4 *are initially marked by* μ. *The transition* t_1 *have the set* $\{p_1,p_4\}$ *as its pre of conditions and the set* $\{p_2,p_5\}$ *as its post of conditions. Finally* $Nm_\omega(N) = \{p_1,...,p_6\}$.

The notion of a directed path between two nodes in *PTNets* is very similar as a directed path between two nodes in directed graphs. For example, on Figure 3.1, it could be seen that there is a path between nodes p_1 and t_3, since there are some nodes and some edges that connect them. If it has a path from x to y, and a path from y to z, then it is quite natural to think that there is a path from x and z. The formal definition for paths in Petri Nets is below.

Definition 3.2 (Paths, Connectedness) *Let* $N = (T, \partial_-, \partial_+) \in PTNets$. *A path in N is defined recursively as follows:*
a) $\forall t \in T, \forall u \in \partial_-(t), \forall v \in \partial_+(t)$, *there are paths,* $u \rightarrow t, t \rightarrow v$, *respectively.*
b) If $x \rightarrow y \wedge y \rightarrow z$, *then* $x \rightarrow z$.
N is ***weakly connected****, or* ***connected****, iff* $\forall x,y \in Nm_\omega(N) \cup T$, *there is a path* $x \rightarrow y$ *or* $y \rightarrow x$.
N is ***strongly connected****, iff* $\forall x,y \in Nm_\omega(N) \cup T$, *there are paths* $x \rightarrow y$ *and* $y \rightarrow x$.

Now, it is defined the isomorphism for Petri Nets. Isomorphism determines when two Petri Nets have "similar" structure and initial marking, and accordingly "similar" behavior. Two Petri Net diagrams that represent the same Petri Net may look quite different. Often, it is important to determine when two diagrams of Petri Nets are in fact the same Petri Net. Intuitively, two Petri Nets are isomorphic if each one can be redrawn to obtain the other. Isomorphism maps places to

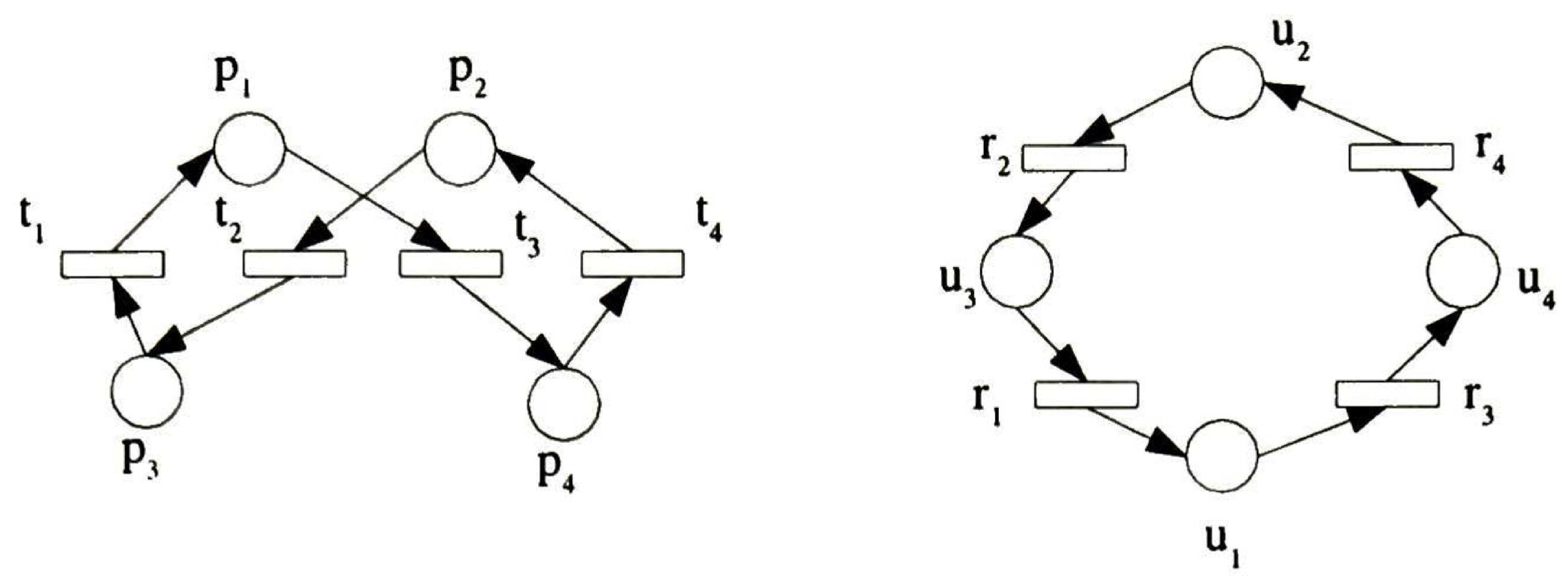


Figure 3.2: Two isomorphic Petri Nets

places and transitions to transitions, preserving the flow relation between them. The next is the formal definition for isomorphism in Petri Nets.

Definition 3.3 (Isomorphism for MPTNets) *Let $(N,\mu),(N',\mu') \in MPTNets$, such that $N = (T,\partial_-,\partial_+)$ and $N' = (T',\partial'_-,\partial'_+)$. Then the two nets (N,μ) and (N',μ') are isomorphic, denoted $(N,\mu) \cong (N',\mu')$, iff there exist a pair isomorphism $\langle f_t, f_p \rangle$, $f_t : T \rightarrow T'$ and $f_p : \omega \rightarrow \omega$ such that $\partial'_i \circ f_t = (id_{Nm} + f_p) \circ \partial_i$, $i = -, +$, and $(id_{Nm} + f_p)(\mu) = \mu'$.*

Private places are the only ones relevant for isomorphism, i.e., in order to two nets could be isomorphic it is necessary that they have the same number of private places. Also, it is necessary that the nets have the same number of transitions. Finally, it is necessary that the nets have similar flow relation between their transitions and their private places. For example, the two nets depicted in the Figure 3.2 are isomorphic, since there exist the pair isomorphism $\langle f : T \rightarrow T', g : \omega \rightarrow \omega \rangle$ where: $f(p_i) = u_i$, $i = 1, ..., 4$, and $g(t_j) = r_j$, $j = 1, ..., 4$.

Once defined the Petri Nets, it is possible to define a special composition between two Petri Net models. The "juxtaposed composition" between two nets is depicted in the Figure 3.3, that is the juxtaposed composition of two nets like the net on Figure 3.1. The composition is performed merging the public places p_1 and p_4 in both nets. All other places, are new places in the compound net. Note that, in this class of composition, the transitions are always new in the compound net. The next definition of net composition is also in [Buscemi, 2001].

Definition 3.4 (The juxtaposed composition for Petri Nets) *Let $(N,\mu_0),(N',\mu'_0) \in MPTNets$ such that $N = (T,\partial_-,\partial_+)$, and $N' = (T',\partial'_-,\partial'_+)$. The parallel juxtaposed composition, denoted by $\otimes$, of (N,μ_0) and (N',μ'_0) merges public places that have the same name preserving the private ones. Formally it is defined as: $(N,\mu_0) \otimes (N',\mu'_0) = ((T + T', \partial'_- + \partial'_-, \partial_+ + \partial'_+), \mu_0 + \mu'_0)$, if $\omega(N) \cap \omega(N') = \emptyset$, where $\omega(N)$ are the local places of N. This product is also called the "coproduct without synchronization" Note that if $\omega(N) \cap \omega(N') \neq \emptyset$, it is always possible to perform an α renaming to achieve it.*

The next example explain the Juxtaposed composition of two nets.

Example 3.2 (The composition of two nets) *Let $(N,\mu),(N'\mu') \in MPTNets$, such that $N = (t_1 = \{p_1,p_4\} \triangleright \{p_2,p_5\}, t_2 = \{p_2\} \triangleright \{p_3\}, t_4 = \{p_5\} \triangleright \{p_6\}, t_3 = \{p_3\} \triangleright \{p_1\}, t_5 = \{p_6\} \triangleright \{p_4\})$, and*

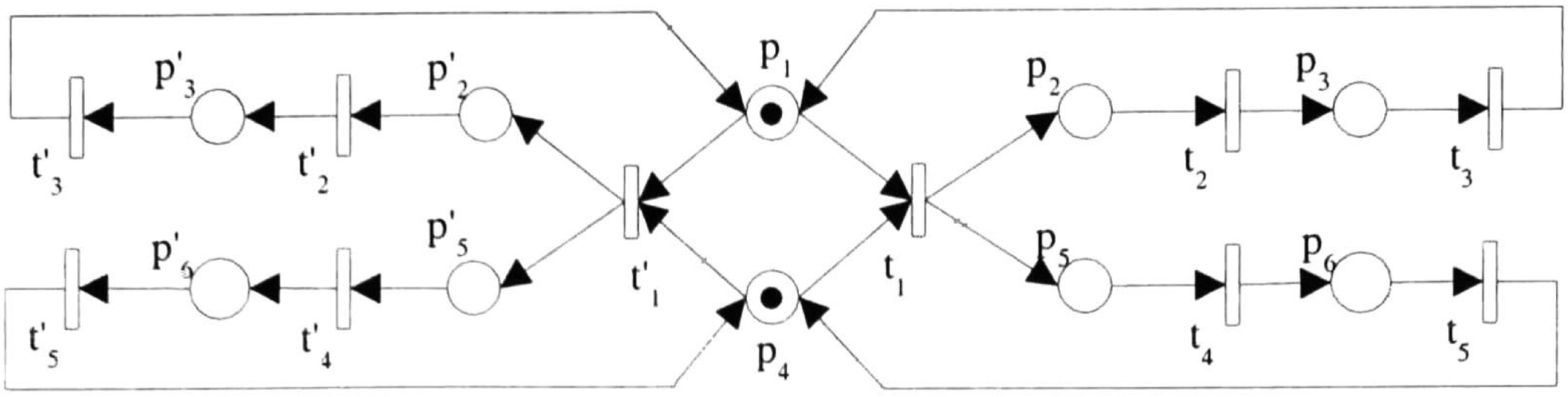


Figure 3.3: The Juxtaposed Composition of two Nets

$\mu = \{p_1, p_4\}$; *$N' = (t_1'\{p_1, p_4\} \rhd \{p_2, p_5\}, t_2' = \{p_2\} \rhd \{p_3\}, t_4' = \{p_5\} \rhd \{p_6\}, t_3' = \{p_3\} \rhd \{p_1\}, t_5' = \{p_6\} \rhd \{p_4\})$, and $\mu' = \emptyset$. The private places of N and N' are, $\omega(N) = \{p_2, p_3, p_5, p_6\}$, and $\omega(N') = \{p_2, p_3, p_5, p_6\}$. Then $N \otimes N' =$*
$((t_1 = \{p_1, p_4\} \rhd \{p_2, p_5\}, t_2 = \{p_2\} \rhd \{p_3\}, t_4 = \{p_5\} \rhd \{p_6\}, t_3 = \{p_3\} \rhd \{p_1\}, t_5 = \{p_6\} \rhd \{p_4\},$
$t_1' = \{p_1, p_4\} \rhd \{p_2', p_5'\}, t_2' = \{p_2'\} \rhd \{p_3'\}, t_4' = \{p_5'\} \rhd \{p_6'\}, t_3' = \{p_3'\} \rhd \{p_1\}, t_5' = \{p_6'\} \rhd \{p_4\}), \{p_1, p_4\}),$
is the net of the Figure 3.3. As it is shown, the nets are merged by the places p_1 and p_4; all other places are new ones in $N \otimes N'$ Also, in this composition, the transitions are always new in $N \otimes N'$ The marking in $N \otimes N'$ is $\mu'' = \mu + \mu' = \{p1, p4\}$.

3.1.3 Dynamics for Petri Nets

The evolution of a *MPTNet* is produced by the "firing" of its transitions. The tokens in the net are moved from a place to another one by the firing of net transitions. When the conditions in the pre set of a transitions are satisfied by the current net marking, then it is said that such transition is enabled. When a transition is enabled, then it is possible to fire such transition. Now, when a transition is fired, then the actual marking is modified, producing a new marking.

Definition 3.5 (Transition enabling) *Let (N, μ) be a MPTNet, such that $N = (T, \partial_-, \partial_+)$. Let $t \in T$, t is enabled at μ, denoted $(N, \mu)[t\rangle$, iff $\partial_-(t) \subseteq \mu$.*

Given a net transition, if there are at least so many marks in its input places as it is indicated by the function ∂_- it is said that such transition is enabled. As an example of enabled transition see the Figure 3.4. The transition t is enabled, since $\partial - (t) = \{p1, p2\} \subseteq \{2p1, p2\} = \mu$. The figure also shows the new marking produced by the firing of t.

Remark 1 *For any $(N, \mu) \in MPTNet$ it is possible to associate a transition labeling function $l : T \rightarrow L$ such that it assigns a label for each transition on the net. The nets with a function l are often called Labeled nets (see the definition below). Given a $(N, \mu) \in MPTNet$, it is possible to assume, without lost of generality, that such net is a Labeled Net, using the $\lambda -$ label function -that is, the transition labeling function that assigns the invisible label λ to each transition- l over it. This is only for convenience on the next definition.*

The firing of a transition, considered instantaneous, produce a change of marking. The change in the marking of a net system by the firing of a transition is represented by the next relation.

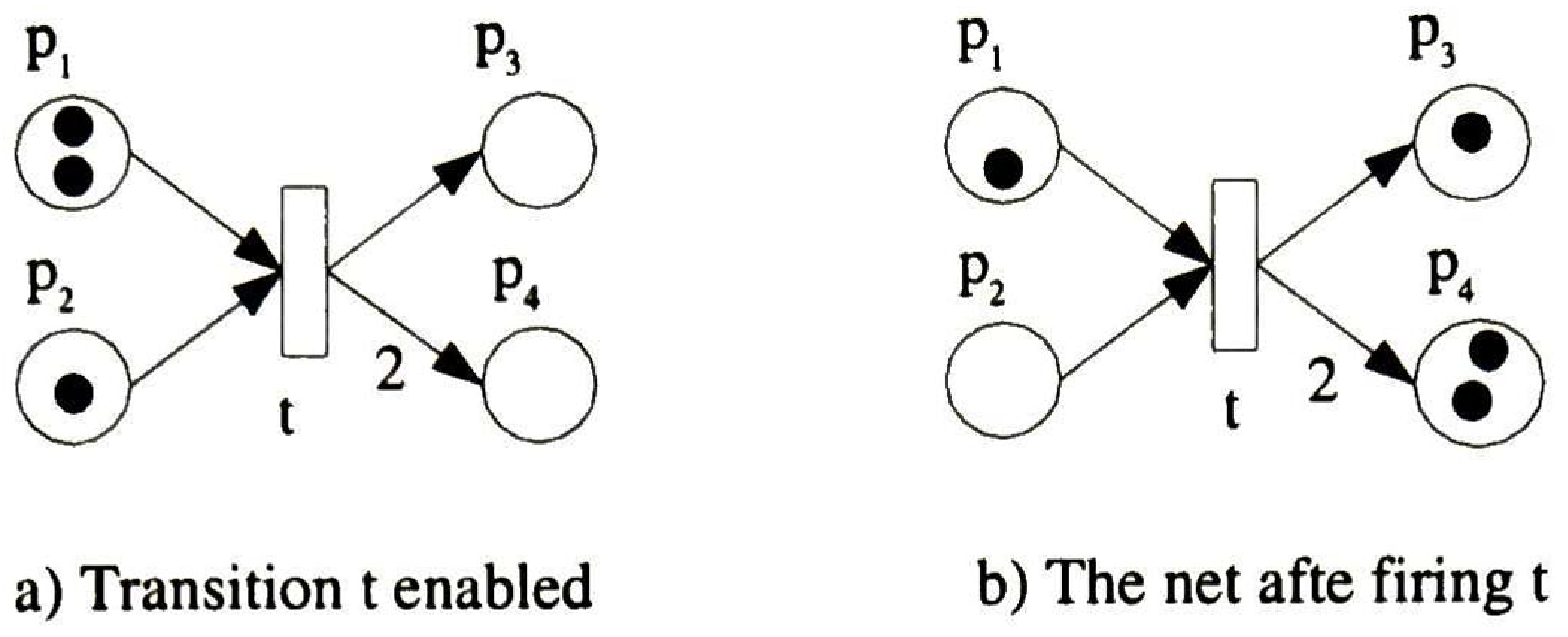


Figure 3.4: Transition Enabling

Definition 3.6 (Firing Rule) *The firing rule for a labeled $MPTNet$, denoted, $\_[\_\rangle\_ \subseteq LMPTNets \times L \times LMPTNets$, is the smallest substitutive relation satisfying $\forall(N,\mu)|(N,\mu)$ is a $MPTNet$ and $\forall t \in T$,*

$$(N,\mu)[t\rangle \Longrightarrow (N,\mu)[l(t)\rangle(N,\mu\backslash\partial_-(t)+\partial_+(t)).$$

where $l(t)$ is the label-function l over $(N.,\mu)$.

When a transition is fired, a new marking is produced. If this new marking enables other transitions, then it is possible to fire them and, consistently, to produce new markings. If this mechanism is done repeatedly, it is possible to get a "Firing Sequence" as the concatenation of firing of transitions that enable other transitions, and so on. The formal definition is given below.

Definition 3.7 (Firing sequence) *Let (N,μ) be a $MPTNet$. A sequence $\sigma \in T^*$ is called firing sequence of (N,μ) iff for some $n \in \mathbb{N}$, there exist markings $\mu_0,...,\mu_n \in \mu(Nm_\omega)$, and transitions $t_1,...,t_n \in T$ such that $\sigma = t_1,...,t_n$, and for all $0 \leq i \leq n, (N,\mu_i)[t_{i+1}\rangle$, and $\mu_{i+1} = (\mu_i\backslash\partial_-(t_{i+1}))+\partial_+(t_{i+1})$, and $\mu = \mu_0$. It is denoted $(N,\mu)[\sigma\rangle(N,\mu_n)$.*

In a similar way, the markings achieved by the firing of transitions can be grouped to form a set of "Reachable Markings", i.e., the markings reached by the transition firing. This represent the possible state of the system.

Definition 3.8 (Reachable markings) *Let (N,μ) be a $MPTNet$. The set of reachable markings, denoted by $[N,\mu\rangle$, is defined as:*

$$[N,\mu\rangle = \{\mu' \in \mu(Nm_\omega)|(\exists\sigma : \sigma \in T^* : (N,\mu_0)[\sigma\rangle(N,\mu'))\}$$

A subclass of Petri Nets very reviewed is Free-Choice Petri Nets [Desel, 1995]. In such nets, every pair transitions have the next property in its pre set: either, they have a disjoint set of places or they have the same set of places. The Figure 3.1 is an example of a Free-Choice Petri Net, since all transitions have no places in common in its pre sets. The importance of Free-Choice Petri Nets is that they have a polynomial proof of liveness [Desel, 1995], rather than no polynomial proof of liveness in the general case of Petri Nets [Petri, 1962]. The formal definition for Free-Choice Petri Net is as follows.

Definition 3.9 (Free-choice LPTN) *Let (N, μ) be a $MPTNet$, such that $N = (T, \partial_-, \partial_+)$, then N is a free-choice net ($FCNet$) iff $\forall t, u \in T$, either $\partial_-(t) \cap \partial_-(u) = \varnothing$ or $\partial_-(t) = \partial_-(u)$.*

The next theorem, called Commoner's Theorem, characterize the liveness for a $FCNet.$

Theorem 1 (Commoner's Theorem) *A $FCNet$ is live iff every proper siphon includes an initially marked trap*

Proof. See [Desel, 1995]. ■

Generalized Petri Nets have no determinism, i.e., given a net and two enabled transitions in it, if these transitions have common places in their pre sets, and the firing of one of these transitions disqualify the other one, then there is no mechanism to decide which one of these transitions must be fired. An extension to Generalized Petri Nets is performed by adding a special function l called transition labeling function. This function puts a label to each transition in the net, in such way that a labeled transition is enabled if and only if it is enabled as in Generalized Petri Nets and at the same time there is an input signal in the system with the same name as its label. Using this mechanism, it is possible to avoid the no determinism problem. Of course it is necessary an external agent making the decision. The formal definition is the next.

Definition 3.10 (Labeled Petri Net) *A $MPTNet$ (N, μ), together with a function $l : T \to L$, where $N = (T, \partial_-, \partial_+)$, is called a Label Marked Petri Net (LMPTNet).*

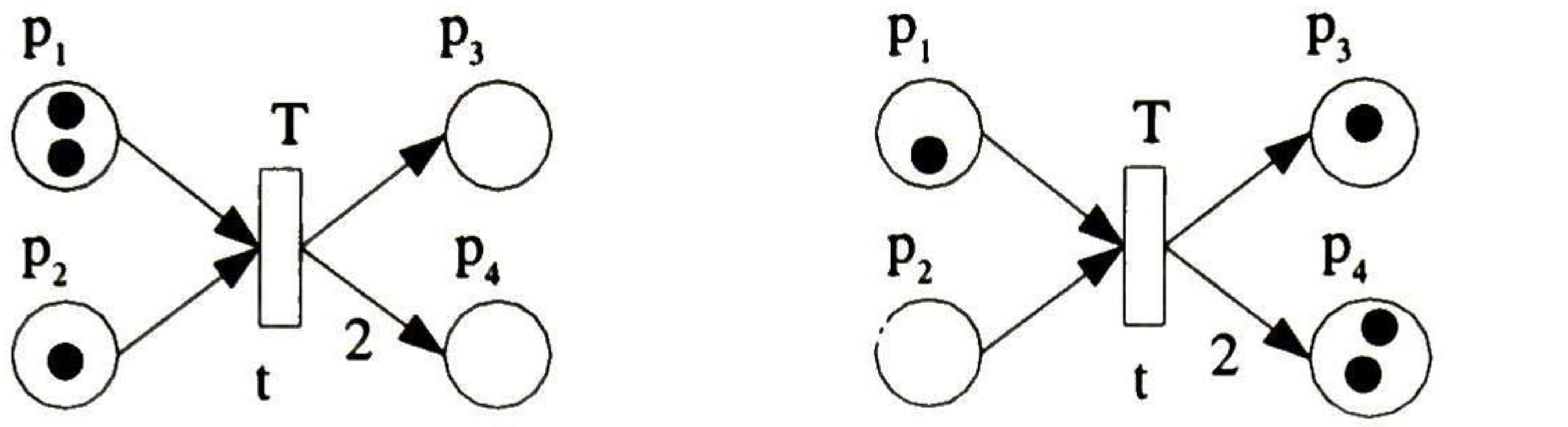
Definition 3.11 (Transition enabling for MLPTNets) *Let $((N, \mu), l)$ be a $LMPTNet$, where $N = (T, \partial_-, \partial_+)$, let $t \in T$, t is enabled at μ, denoted $(N, \mu)[t\rangle$ iff t is enabled in the sens of $MPTNets$ and at the same time there is an input signal in the system with the same name as its label.*

As an illustration of these concepts, see the Figure 3.5. The Figure 3.5a shows a enabled transition t in the sens of Generalized Petri Nets. Now, in order to t be enabled in Labeled Nets, it is only necessary that the input signal T is present in the system at this time. If it is assumed the input signal T, then it is possible to fire t and the new marking for the net is shown in Figure 3.5b. Now, however the input signal T is present in the system, it becomes irrelevant, since the transition t now is not enabled in the sens of $MPTNets$.

3.1.4 Petri Nets properties

There are some structural properties in discrete event systems that meet the Petri Nets. Two of these properties are liveness and boundedness. Liveness property is related to the absence of deadlocks and the total system lode. When, in the Petri Net evolution is reached a marking that enables no transitions, it is said that the net system is in deadlock. The liveness property is deals with the absence of deadlocks.

Definition 3.12 (Liveness) *Let (N, μ_0) be a $MPTNet$, such that $N = (T, \partial_-, \partial_+)$, N is live iff $\forall \mu \in [N, \mu_0\rangle, \forall t \in T, \exists \mu' \in [N, \mu\rangle$ such that $(N, \mu')[t\rangle$.*



a) Transition t enabled and the input signal T is present

b) The net afte firing t, the input signal T have no relevance

Figure 3.5: Labeled Petri Net

A Petri Net often represents a discrete event system. The markings represent the possible states of such system. Often, the discrete event systems have a finite number of states. Then, Petri Nets that represent a finite discrete event system must have a finite set of reachable markings. So, given a Petri Net, it is interesting that the net has a finite number of tokens in each place, giving as consequence a finite set of possible states in the net. When it occurs, the net is called bounded, in other case, the net is called unbounded.

Definition 3.13 (Boundedness) *Let (N, μ_0) be a $MPTNet$, then (N, μ_0) is bounded iff $[N, \mu_0\rangle$ is finite.*

The next theorem, called the "S-Invariant" theorem, characterize the boundedness on a Petri Net.

Theorem 2 (S-Invariant theorem) *Let (N, μ_0) be a $MPTNet$, then (N, μ_0) is bounded iff (N, μ_0) has a positive s-invariant.*

Proof. See [Lopez, 1997]. ■

Definition 3.14 (Soundness) *Let (N, μ_0) be a $MPTNet$. Then (N, μ_0) is sound iff it is live and bounded .*

In Petri Nets there is a variety of subclasses of nets. A Safe Net, also called a Binary Net, has a special property: every place in the net has always either, no tokens or one token. It is easy to see that a Safe Net is also a bounded net.

Definition 3.15 (Safeness) *A $(N, \mu_0) \in MPTNets$ is safe iff $\forall \mu \in [N, \mu_0\rangle$ and $\forall s \in Nm_\omega$, $\mu(s) \leq 1$.*

Example 3.3 (Petri Net properties) *Consider the net depicted in Figure 3.6 as an illustration of these concepts: the transition t_1 is enabled at $\mu = \{p_1\}$, since $\partial_-(t_1) \subseteq \mu$. The sequence $t_1t_2t_3$ is a firing sequence of (N, μ) since there exist markings $\mu = \{p_1\}, \mu_1 = \{p_2\}, \mu_2 = \{p_3\}, \mu_3 = \{p_5\}$ and transitions t_1, t_2, t_3 such that: $(N, \mu)[t_1\rangle(N, \mu_1)[t_2\rangle(N, \mu_2)[t_3\rangle(N, \mu_3)$. The markings μ, μ_1, μ_2, μ_3 form part of the set of reachable markings of (N, μ). It is easy to see that the net is also safe, live and bounded.*

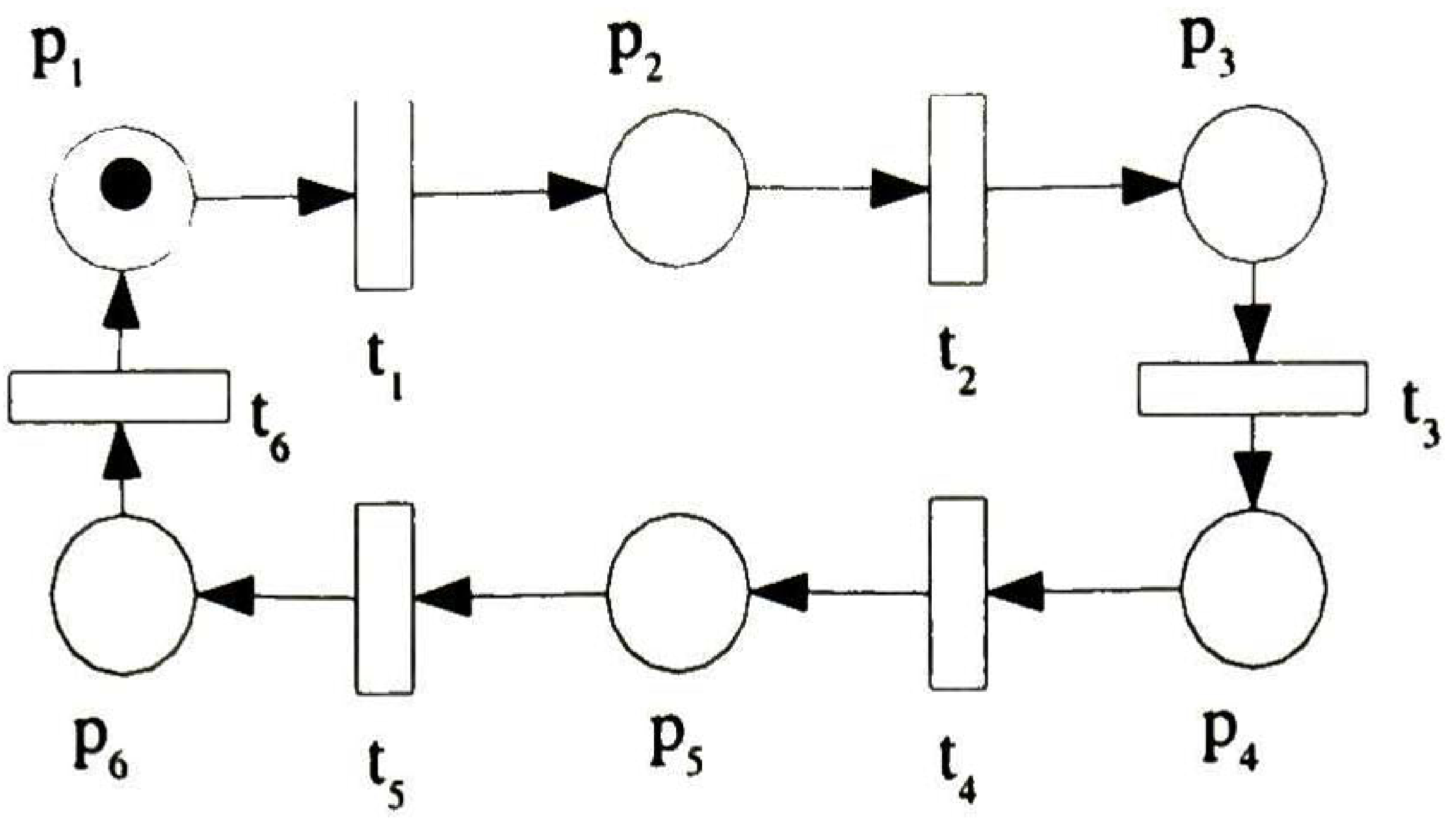


Figure 3.6: Simple Petri Net

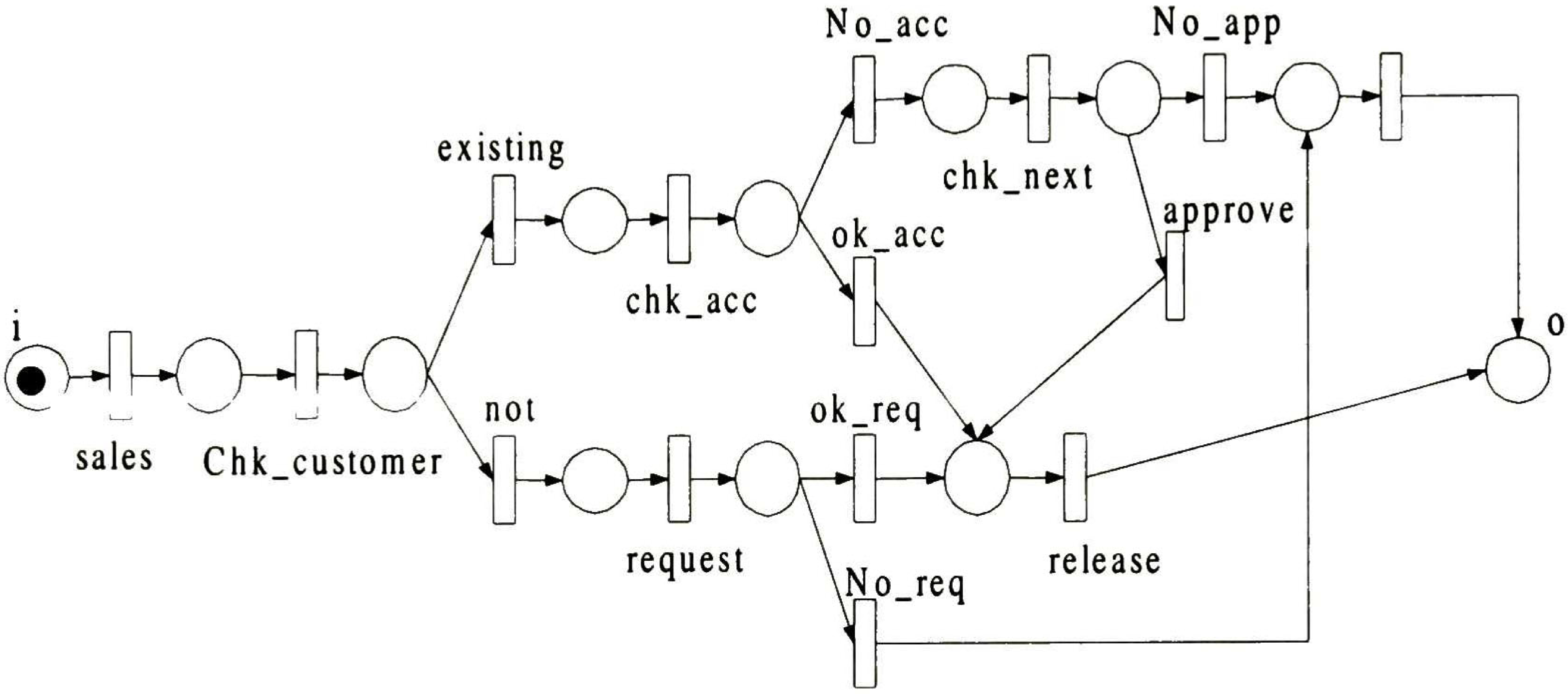


Figure 3.7: A Workflow Net

3.2 Workflow Patterns as Workflow Nets

Petri Nets are successfully used in Workflow Management for the modelling of business processes [Aalst, 1997]. In this section, it is presented a mapping from Workflow patterns into Petri Nets. In many cases, the translation is straightforward; in other cases the translation results more complex. First, it is presented Workflow Nets, a subclass of Petri Nets that meets the workflow patterns. Then, it is presented the Workflow patterns as Workflow Nets.

Workflow Nets was presented by [Aalst, 1997], and it was used in many works related to Workflow: [Aalst, 1998], [Aalst, 1999], [Aalst, 2000 WP 50], [Badouel, 1998]. Figure 3.7 shows an example of Workflow Nets. Workflow Management is case based, i.e., there is an individual instance of a Workflow process for each real case in the system. In other words, for example, if in a certain moment there are two clients that make the same request, then there are two copies of the same Workflow process to attend such cases.

A Workflow Net is a kind of "causal" net, i.e., it has a "flow" from one initial place, that has no inputs, to one final place, that has no outputs. A Workflow Net has two special states, initial state and final state. In the initial state, it is marked only the initial place, which denotes the case creation. On the other hand, in the final state it is marked only the final place, which denotes de case termination.

Also, in a Workflow Net, the flow of markings is from the initial place to the final place. In the Figure 3.7 it is shown a place labeled with "i" and a place labeled with "o", those places represent the initial place and the final place, respectively. Also some other concepts of Workflow Management are shown: the transitions with input signals "existing", and "not" that result in a simple choice from its common input place. Also, the output place "o" is in fact, a simple merge point. It is easy to see that the net is safe. The formal definition of Workflow Nets is as follows:

Definition 3.16 (Workflow Nets) *Let (N, μ) be a $MPTNet$ and $\bar{t}$ be a "fresh" name not in $Nm_\omega(N) \cup T$ Then (N, μ) is a Workflow Net ($WFNet$), iff:*

Case creation: N contains an input place $i \in Nm_\omega(N)$ such that $\bullet i = \varnothing$,

Case completion: N contains an output place $o \in Nm_\omega(N)$ such that $o\bullet = \varnothing$, and

Connectedness: $N' = (T \cup \{\bar{t}\}, \partial_- \cup \{(\bar{t}, o)\}, \partial_+ \cup \{(\bar{t}, i)\})$ is strongly connected.

The input place for a Workflow Net often will be represented by i, and its output place by o. Also, the $START$ state denotes the marking $\mu = \{i\}$, i.e., the marking with one token in the input place i, and no tokens in all other places. The END state denotes the marking $\mu = \{o\}$, i.e., the marking with one token in the output place o, and no tokens in all other places.

In Workflow Management, there are some desired properties for Workflow process models. Some of these properties are: safeness, proper completion, absence of deadlock and absence of dead transitions, etc. The safeness property has been defined in Petri Net, and it is necessary since Workflow Management is case based. The proper completion property for a workflow net, is related to only one "token" in its output place, and no tokens in all remaining places. The absence of deadlock property has the meaning that for every possible state in the net it is possible to reach the END state. The absence of dead tasks it is explained by itself.

Definition 3.17 (Soundness of WFNets) *Let (N, μ) be a $WFNet$, then (N, μ) is sound iff:*

safeness: $(N, [i])$ is safe,

proper completion: $\forall s \in [N, \{i\}\rangle, o \in s \Longrightarrow s = \{o\}$,

absence of deadlock: $\forall s \in [N, \{i\}\rangle, o \in [N, \{s\}\rangle$, and

absence of dead tasks: $(N, \{i\})$ contains no dead transitions.

Definition 3.18 (Workflow process definition) *A workflow process definition is a sound $WFNet$. The set of all workflow process definitions is denoted $\mathcal{W}$.*

In order to prove some properties, it is necessary to work with a modified Workflow Net. Such modification is made adding a new transition to a workflow net that connects its output place to its input place in this direction, i.e., this new transition moves the "tokens" from the output place to the input place. Using this modified workflow net it is easy to proof the soundness for a given Workflow Net, as it is shown in [Aalst, 1998]. The formal definition for the modified workflow net is below.

Definition 3.19 (Underlying PTNet) *Let $(N, \mu) \in \mathcal{W}$. Let $(\bar{N}, \mu)$ be a $MPTNet$, such that $\bar{N} = (\bar{T}, \bar{\partial}_-, \bar{\partial}_+)$, where: $\bar{T} = T \cup \{t^*\}$ and $\bar{\partial}_- = \partial_- \cup \{(t^*, o)\}$, and $\bar{\partial}_+ = \partial_+ \cup \{(t^*, i)\}$. Then*

*$(\bar{N}, \mu)$ is called the **underlying** MPTNet of the Workflow process definition (N, μ). That is, the underlaying PTNet for a Workflow process definition is the Workflow Net with a transition that joins its output place to its input place in this order.*

The proof of soundness of a Workflow Net use the notion of underlying Workflow Net and the properties of liveness and safeness for Petri Nets. The soundness of a *WFNet* has an easy proof, as the next theorem shows.

Theorem 3 (Characterization of soundness) *Let $(N, \{i\})$ be a WFNet, then $(N, \{i\})$ is sound iff: $(\bar{N}, \{i\})$ is live and safe.*

Proof. See [Aalst, 1998] ∎

Now, it is presented the translation from Workflow patterns to Workflow Nets. In many cases, this translation is straightforward, in many others the translation is more complex. Moreover, there exist cases, such as Case Cancelation pattern, that are very tricky in Petri Nets.

3.2.1 Workflow Patterns as Workflow Nets

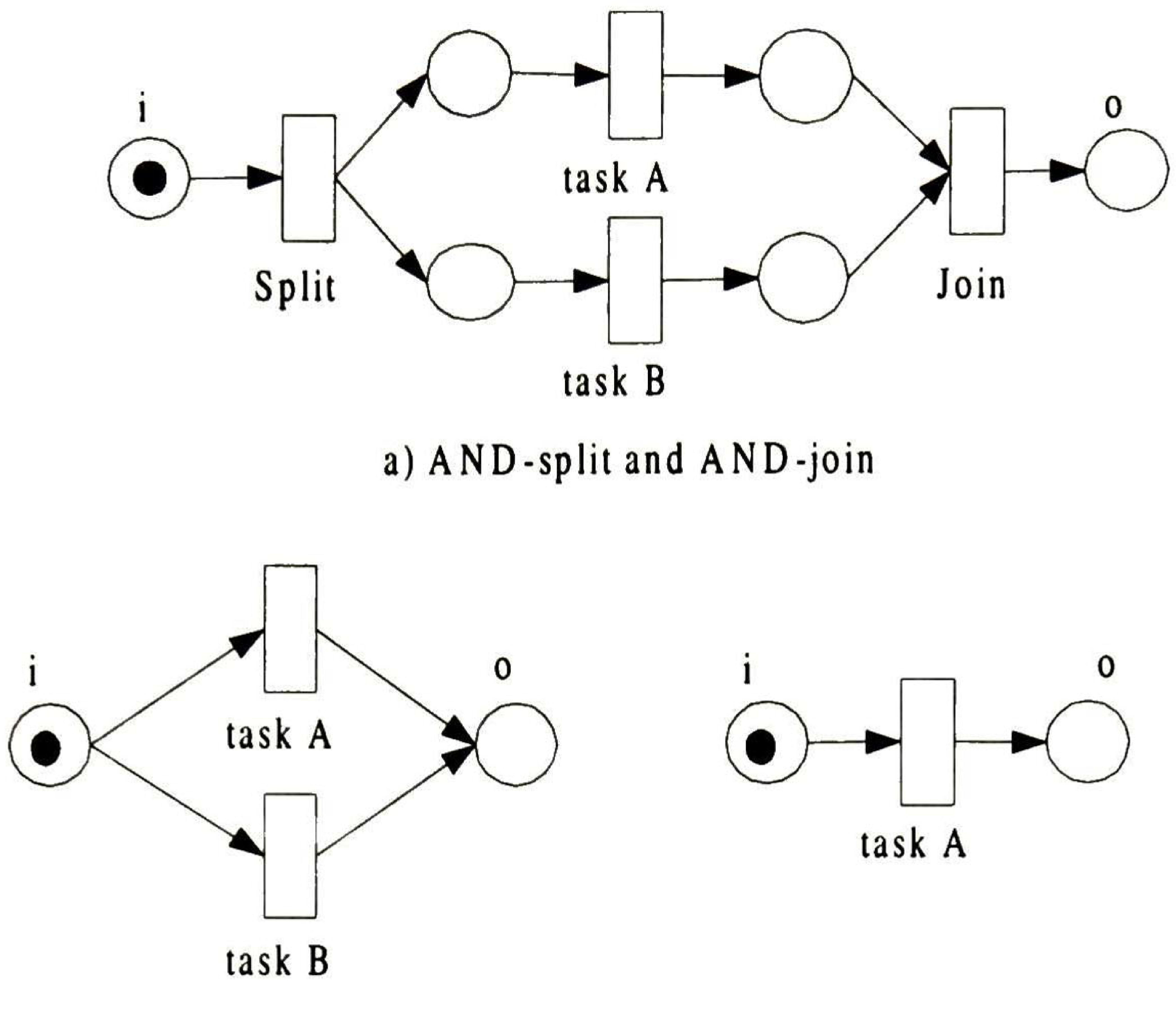
Workflow patterns have an equivalence in Workflow Nets. A Workflow model could be formed using many workflow patterns connected in sequence or in parallel, as it will show later.

In this section, it is presented some equivalent Workflow Nets for such patterns. Also it is shown that all Workflow Net proposed are live and bounded.

Remark 2 *Note that all of the proofs of soundness for all Workflow Nets are given respect to their underlying MPTNet. Also, they are used the Commoner's Theorem and the s-invariant Theorem, reviewed in the previous section, for proving of liveness and boundedness, respectively.*

Basic Control Flow Patterns

The Basic Control Flow patterns have a direct representation in Petri Nets. The Figure 3.8 shows the Basic Control Flow patterns as Workflow Nets. The Parallel split and the Synchronization patterns are presented in Figure 3.8a in its initial state, i.e., only place "i" is marked. The "Split" transition with two output places starts the transitions Task A and Task B, that run in parallel. The "Join" transition waits Task A and Task B end, then the synchronization take place, and the final state is reached. The Exclusive Choice and Simple Merge patterns are presented in Figure 3.8b, once again, in its initial state. Either, Task A or Task B could be fired consuming the token into the place "i". i.e., an exclusive choice is made between Task A and Task B. Then the final state could be reached by the fire of them. Finally, the Simple Sequence pattern is depicted in Figure 3.8c. The 1-length simple sequence is represented by the transition Task A with an input place "i" and output place "o" that are the initial and final places too. The soundness proof for these Workflow Nets is omitted due to its simplicity.



a) AND-split and AND-join

b) OR-split and OR-join

c) An activity

Figure 3.8: Basic Control Flow Patterns Nets

Advanced Branching and Synchronization patterns

Figure 3.9 shows the N-out-of-M join pattern as a Workflow Net. This net meets the particular case of 2-of-3 join. On it, "3" parallel transitions (tasks) are activated, and then, the place "p" waits for "2" of these "3" parallel transitions to be completed before the firing of the next activity; in the next step, all remaining tasks are "removed" by the action of the, "flush" transitions in its input place, enabled at this stage..

Note that there are no certainness of what transitions are completed and what are not, but it is sure that "2" of these "3" transitions are completed before the next activity starts. Also, note that there is a "flush" transition for any task from 1 to 3, $(t1', t2', t3')$. Such transitions are used to cancel any remaining task, after that "2" of these "3" are completed. The formal definition for this Workflow Net is as follows.

Definition 3.20 (N-out-of-M Join) *WFNet on Figure 3.9 shows a point (the place p) waiting for 2 of 3 incoming branches; once 2 of such branches are completed, the next activity is started, i.e., a token is delivered into its input place; at this stage, other activity can take place. Finally, in the next step, the all remained branches are take off. This net meets the particular case of 2-of-3 join. Note that the transitions $t1', t2'$, and $t3'$ are "flush" transitions for the tasks $t1, t2$, and $t3$ respectively.*

Proposition 4 *The 2-out-of-2 Join Workflow Net depicted in Figure 3.9 is live and there are no forgotten tokens on it.*

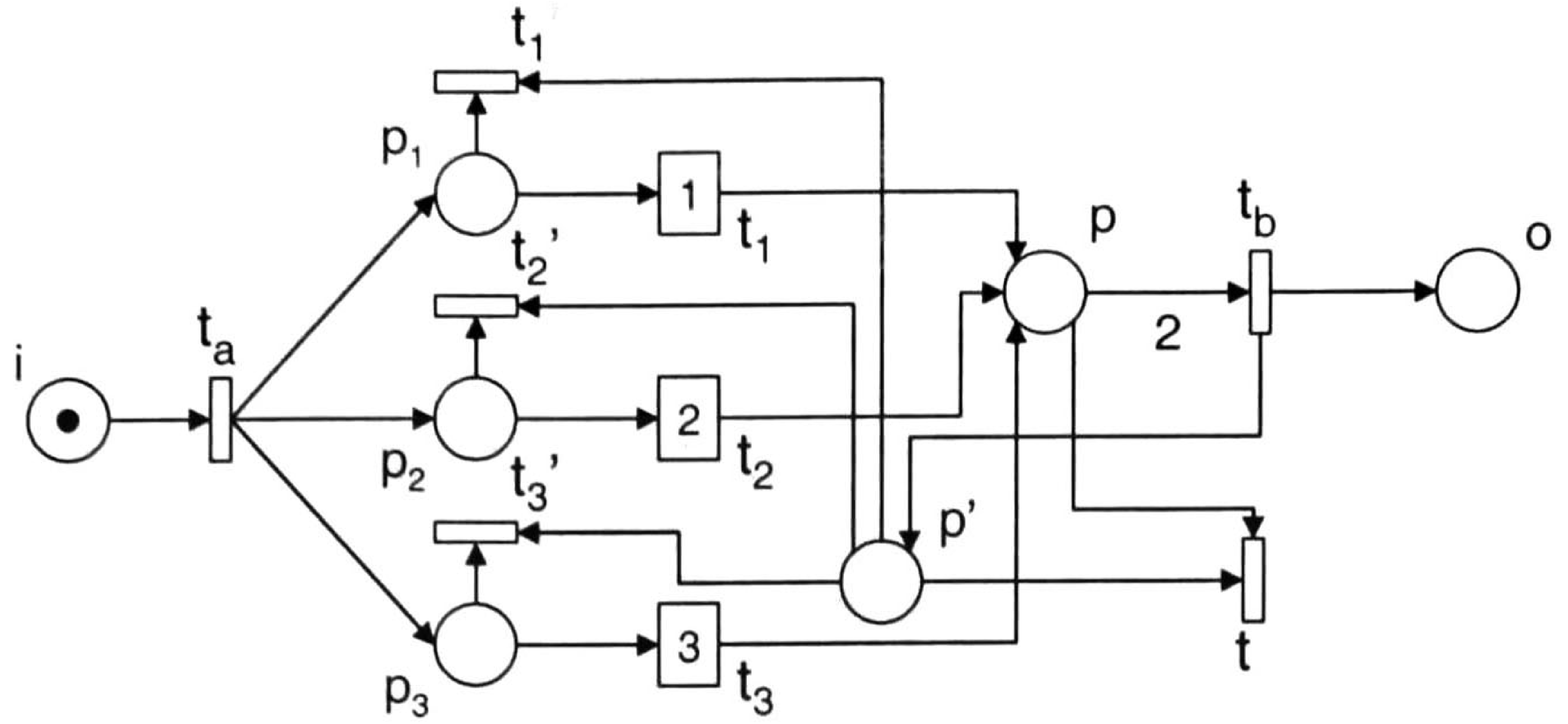


Figure 3.9: The 2-out-Of-3 Pattern Net

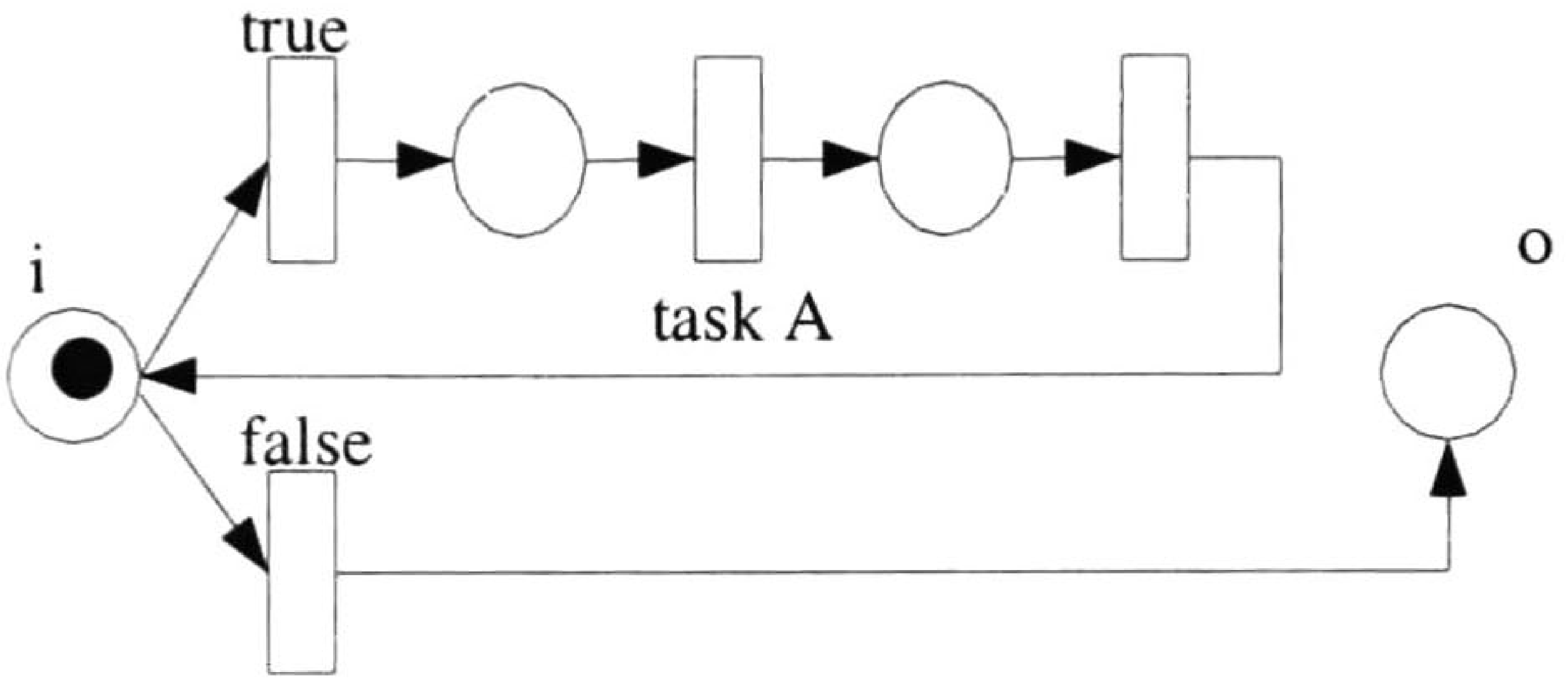


Figure 3.10: The Arbitrary Cycle Pattern Net

Proof. *a)Liveness: Let $I = (4313131414)$; I is a positive t-invariant of the net, then the net is live.*

b)No forgotten tokens. This property means that at the stage where the net has finished, then no other activity is performed on it. It is achieved allowing that all remaining task could be canceled. Figure 3.9 shows a place p' that has enough tokens to cancel all remaining task that have not finished at the stage where "N" of these "M" are completed. So, if all the transitions that cancel the remaining tasks are considered autonomous, then they fires immediately, and no tokens are forgot. ■

Structural patterns

The Figure 3.10 shows the Arbitrary Cycle pattern as a Workflow Net. The transition "Task A" can be done repeatedly until the transition with input signal "False" is fired. So the END state can be reached. This pattern meets the situation when an activity or a set of them must be repeatedly executed.

Definition 3.21 (Arbitrary Cycle) *The WFNet on the Figure3.10 shows a task that can be repeatedly done while the input signal "true" is on. It ends when the input signal "false" is on (Note that it is considered true and false input signals in mutual exclusion).*

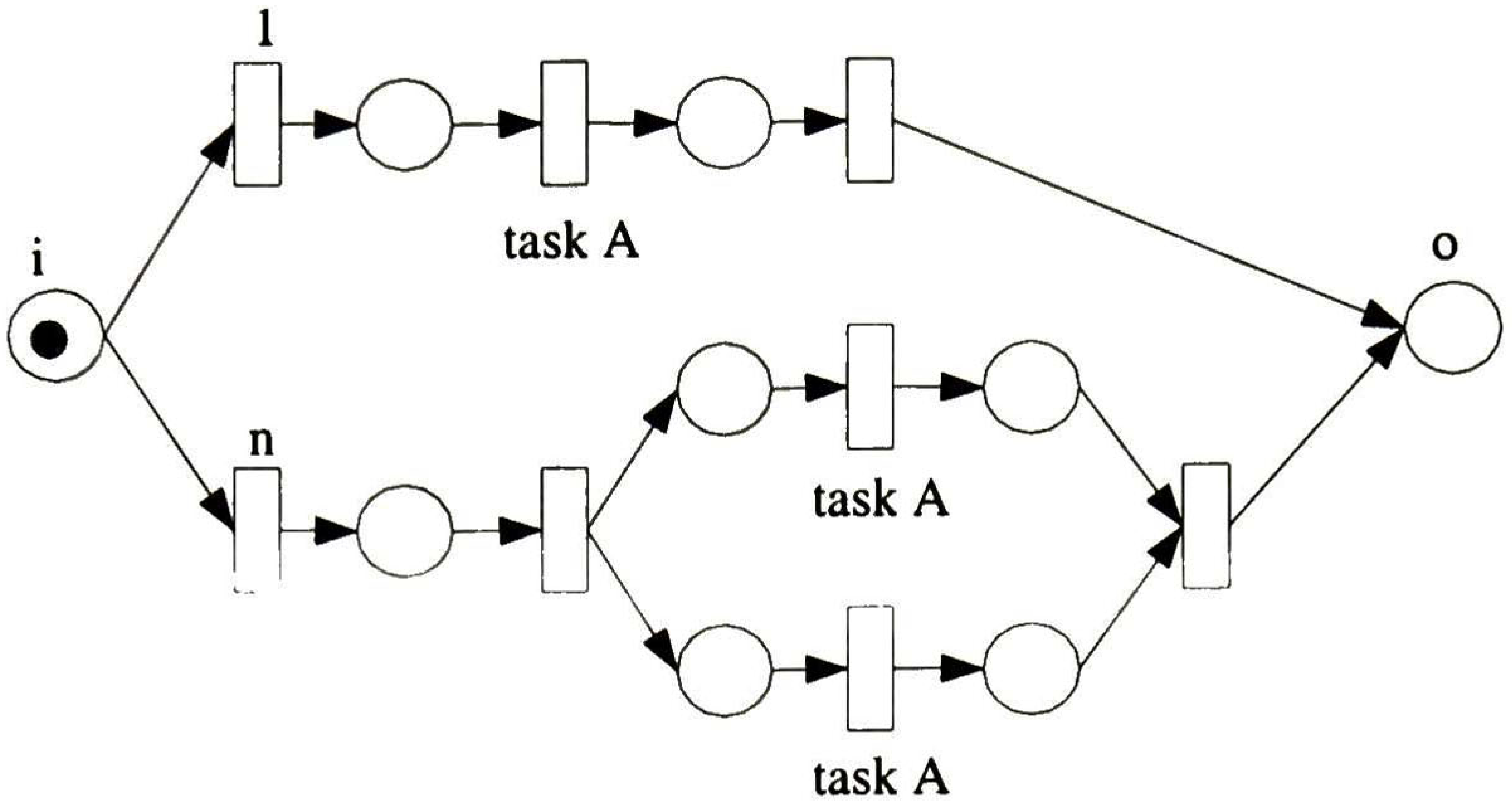


Figure 3.11: The Multi Instances Pattern Net

Proposition 5 *The Arbitrary Cycle Workflow Net is live and bounded.*

Proof. a)Liveness: the underlying $MPTNet$ is a $FCNet$ and every proper siphon includes an initially marked tramp, so the system is live.

b)Boundedness: let $I = (111)$; I is a positive s-invariant of the underlying $MPTNet$, so the system is bounded. ■

Definition 3.22 (Implicit Termination) *The Implicit Termination pattern is a property of markings that workflow patterns must have, rather than a structure that can be translated to Petri Nets concepts, i.e., given a Workflow pattern it is desirable that finishes where there are nothing else to be done. Then, this desirable property can be achieved transforming any pattern into a pattern with only one END point, in this way when it reaches the END state, it is possible to know that there are nothing else to be done in the workflow process. In fact, Workflow Nets already have this desirable property.*

Multiple Instances patterns

Figure 3.11 shows the Multiple Instances patterns as a Workflow Net, where the maximum number of copies of a Task-A is known at design time. Also, Figure 3.11 shows transitions with labels from 1 to n, where n is the maximum number of copies of a Task-A. Also it shows the synchronization for each branch from 1 to n. The Workflow Net operates as follows: in a certain moment, any of the i-labeled transitions, for i from 1 to n, can be fired activating its respective branch. When the branch ends, the synchronization takes place, and the END state is reached. In this way, when the maximum number for a Task-A is known at design time, it is always possible to choose the appropriate labeled transition. However there are some points to consider: When the maximum number of copies of a Task-A is too big, then it is possible to transform it into an iterative repetition of a small number of copies of Task-A using the Arbitrary Cycle pattern. In this way, when the maximum number of copies of a Task-A is greater than the maximum number of copies allowed by a certain Workflow Management system, it is possible to simulate this pattern performing a fewer

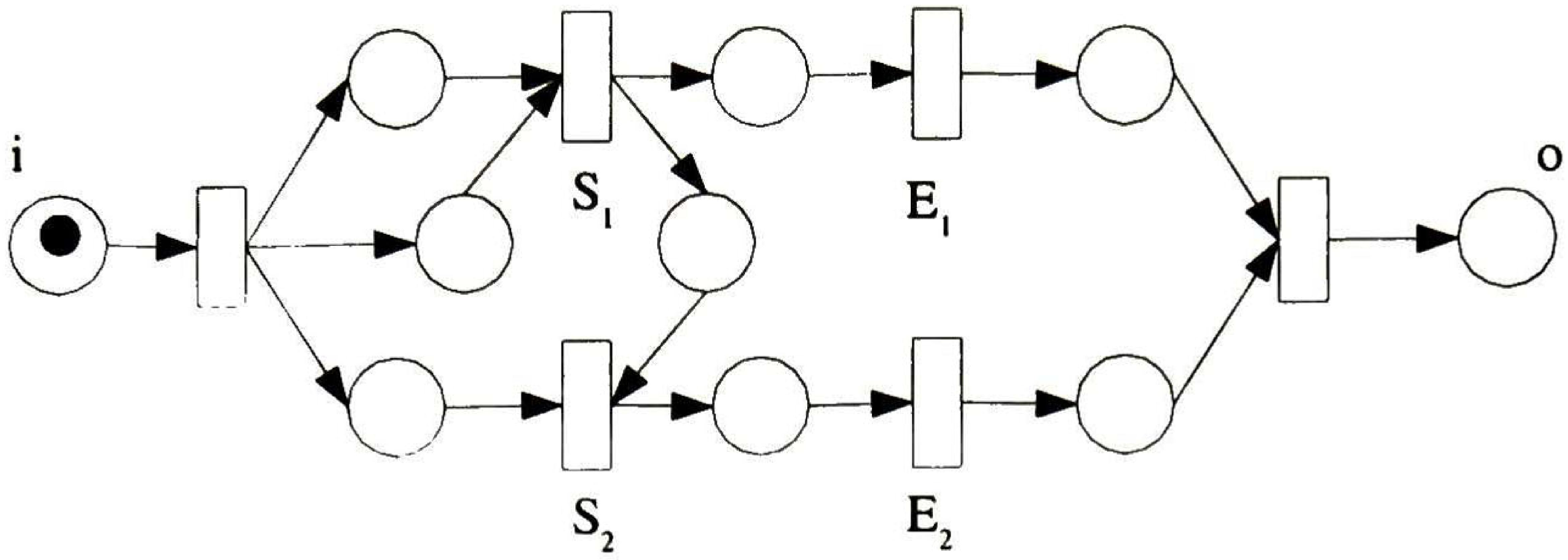


Figure 3.12: The Temporal Relation Pattern Net

number of parallel activities repeatedly. Of course, in this way, the maximum number of copies of a certain Task-A is not really in parallel.

Definition 3.23 (Multiple Instances and Synchronization) *In the WFNet depicted in Figure 3.11 one of several branches can be chosen. If the transition with input signal i is chosen, then i copies of the task A are executed, $1 \leq i \leq n$. Finally, the synchronization takes place for all of these branches $(1, ..., n)$. This pattern meets the case when the number of copies of a task A is n-bounded and it is known at design time.*

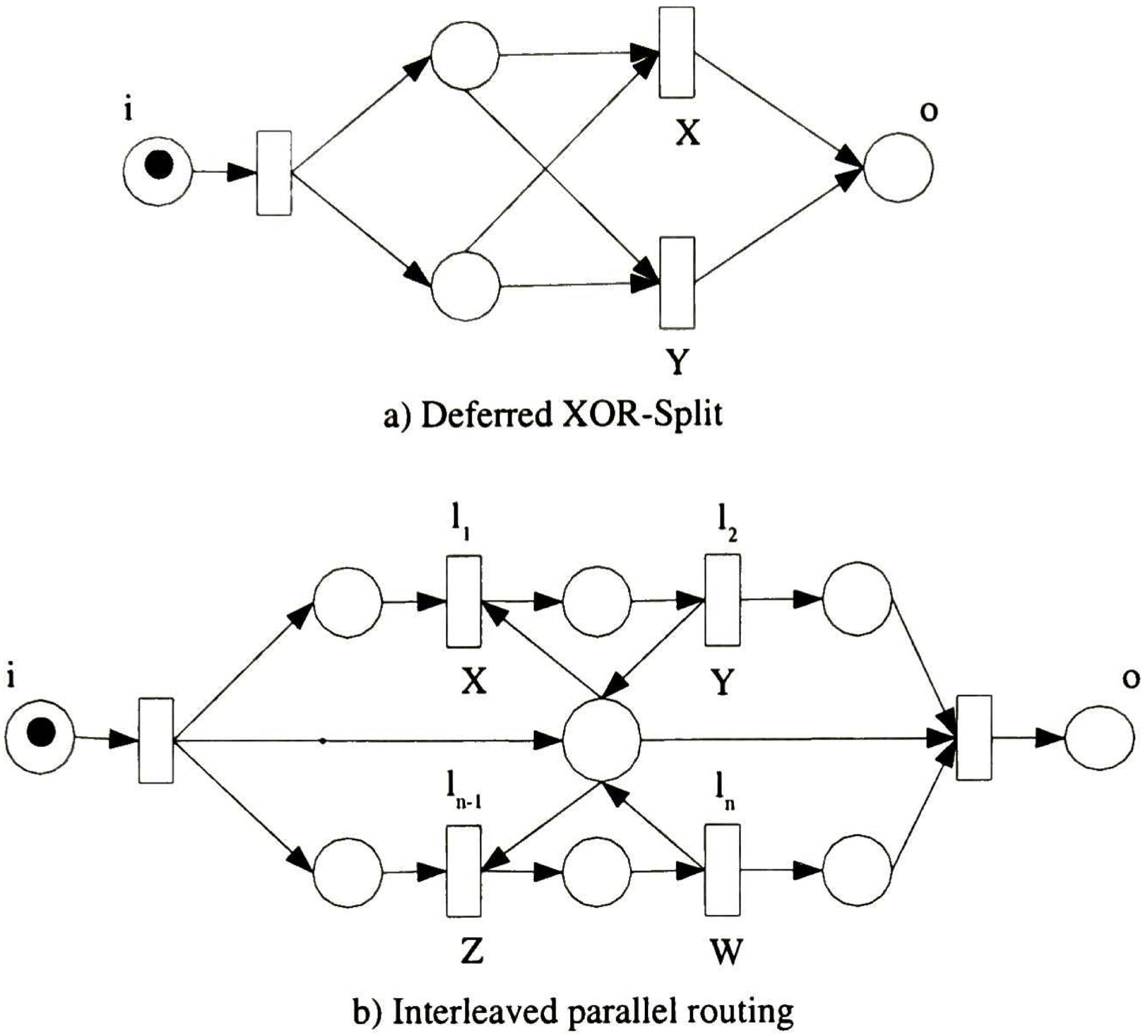
Proposition 6 *The Multiple Instance and Synchronization Workflow Net is live and bounded.*

Proof. a)Liveness: since the underlying $MPTNet$ is a $FCNet$ and since every proper siphon includes an initially marked tramp the system is live.

b)Boundedness: let $I = (111...111)$; I is a positive s-invariant of the underlying $MPTNet$. Note that $|I| = n(n+1) + 4$, where n is the maximum number of copies of the activity A. ■

Temporal Relation patterns

Temporal Relation patterns make a relationship between two tasks other than sequential or parallel ones. As an example, the temporal order relation $S_2 \geq S_1$ means that the activity with start point S_2 must starts after or at the same time than the activity with start point S_1. This have no information about exactly "when" the activity with start point S_2 must starts, but it must starts at the same time or later than the activity with start point S_1 does. Since the change of markings in generalized Petri Nets is considered discrete, and the time is continuous, the major representation of this pattern in Petri Nets is very stiff. However, if it is possible to split each activity into its main milestones, then it is possible to perform complex interleaved relation between them. As another example of temporal relationship between tasks, Figure 3.12 shows the temporal relation "before-start", $S_2 \geq S_1$, as a Workflow Net, where two tasks are divided into two major milestones: Start and End points, S and E respectively. In the Workflow Net, two branches are fired in parallel, one for the Activity-1 and other for the Activity-2. The branch for the Activity-1 could continue without difficult, but the branch for the Activity-2 must waits to the Activity-1 starts before it starts.



a) Deferred XOR-Split

b) Interleaved parallel routing

Figure 3.13: The Deferred XOR-Split and Interleaved Paralle Routing Pattern Nets

Definition 3.24 (Interleaved Sequence) *In the WFNet of Figure 3.12, two branches are shown: the upper one for the Activity-1 and lower one for the Activity-2. The temporal relation is $S_B \geq S_A$, that is, the Activity-2 must starts after or at the same time than Activity-1 starts. In a similar way, it is possible to construct any other "discrete" temporal relation.*

Proposition 7 *The Interleave Sequence Workflow Net is live and bounded.*

Proof. a)Liveness: since the underlying $MPTNet$ is a $FCNet$ and since every proper siphon includes an initially marked tramp the system is live.

b)Boundedness: Let $I = (1111111111)$; I is a positive s-invariant of the underlying $MPTNet$. ■

State-based patterns

Figure 3.13 shows the Deferred XOR-split and the Interleaved parallel routing patterns as Workflow Nets. In the Deferred XOR-split pattern, as in Exclusive Choice pattern, one of several branches is chosen, but in contrast with it, all possible branches are active before the selection, then when one is chosen, the others are cancelled. In this way, the selection is implicit rather than explicit, as in Exclusive Choice pattern. The END state is reached after it.

In the Interleaved parallel routing pattern, a set of activities is executed in an arbitrary order; such order is decided at runtime and no two activities are executed at the same time. This suggest us a mutual exclusion between the set of tasks. Figure 3.13b shows the equivalent Workflow Net for this pattern: a set of transitions have a shared input place that allows the progress of one activity at a time; once the active activity finishes, then the resource is released, and another activity can be executed. When all tasks have been executed, then it is possible to reach the END state. Note that the execution order could be decided at runtime, since each transition have an associated input label.

Definition 3.25 (Deferred XOR-split) *In the net of the Figure 3.13a there are two transitions (X and Y) waiting for an input signal. X and Y transitions are enabled, but only one will be chosen, the other one will be withdrawn.*

Proposition 8 *The Deferred XOR-split Workflow Net is live and bounded.*

Proof. a)Liveness: since the underlying $MPTNet$ is a $FCNet$ and since every proper siphon includes an initially marked tramp the system is live.

b)Boundedness: let $I = (1111)$; I is a positive s-invariant of the underlying $MPTNet$. ■

Definition 3.26 (Interleaved Parallel Routing) *In the net of the Figure 3.13b there are n tasks that can be executed in arbitrary order. When all of these tasks are done, the workflow net reaches its END state.*

Proposition 9 *The Interleaved Parallel Routing Workflow Net is live and bounded.*

Proof. a)Liveness: since the underlying $MPTNet$ is a $FCNet$ and since every proper siphon includes an initially marked tramp the system is live.

b)Boundedness: let $I = (111...111)$; I is a positive s-invariant of the underlying $MPTNet$. Note that $|I| = 3n + 2$. ■

Cancellation patterns

Cancellation patterns are related to the cases when some tasks currently in execution must be abruptly cancelled, i.e., for some external conditions it is necessary the cancellation of these tasks. As in Implicit Termination pattern, this change must be in the task state rather than in its structure. But in this case, any reachable state must be mapped into its final state despite there exist active tasks in the workflow net.

The modelling of these patterns will be addressed in the next chapter titled Dynamic Change within Workflow Management, where the dynamic structural change will be explored.

Inter-Workflow Synchronization patterns

Figure 3.14 shows the Message Coordination pattern as a Workflow Net. As it is shown, the "sender" sends a message to the "receiver" firing its "snd" transition, then it waits the response of the "receiver" The transition "rcv" in the "sender" waits the "receiver" response and then it can be fired to reach the END state. This Workflow Net meets the message coordination behavior. Of course, in both sides, at the "sender" side and at the "receiver" side it is possible to perform many other tasks. Note that it is no important the connection and disconnection dynamic between the client and the server, but only the message coordination behavior.

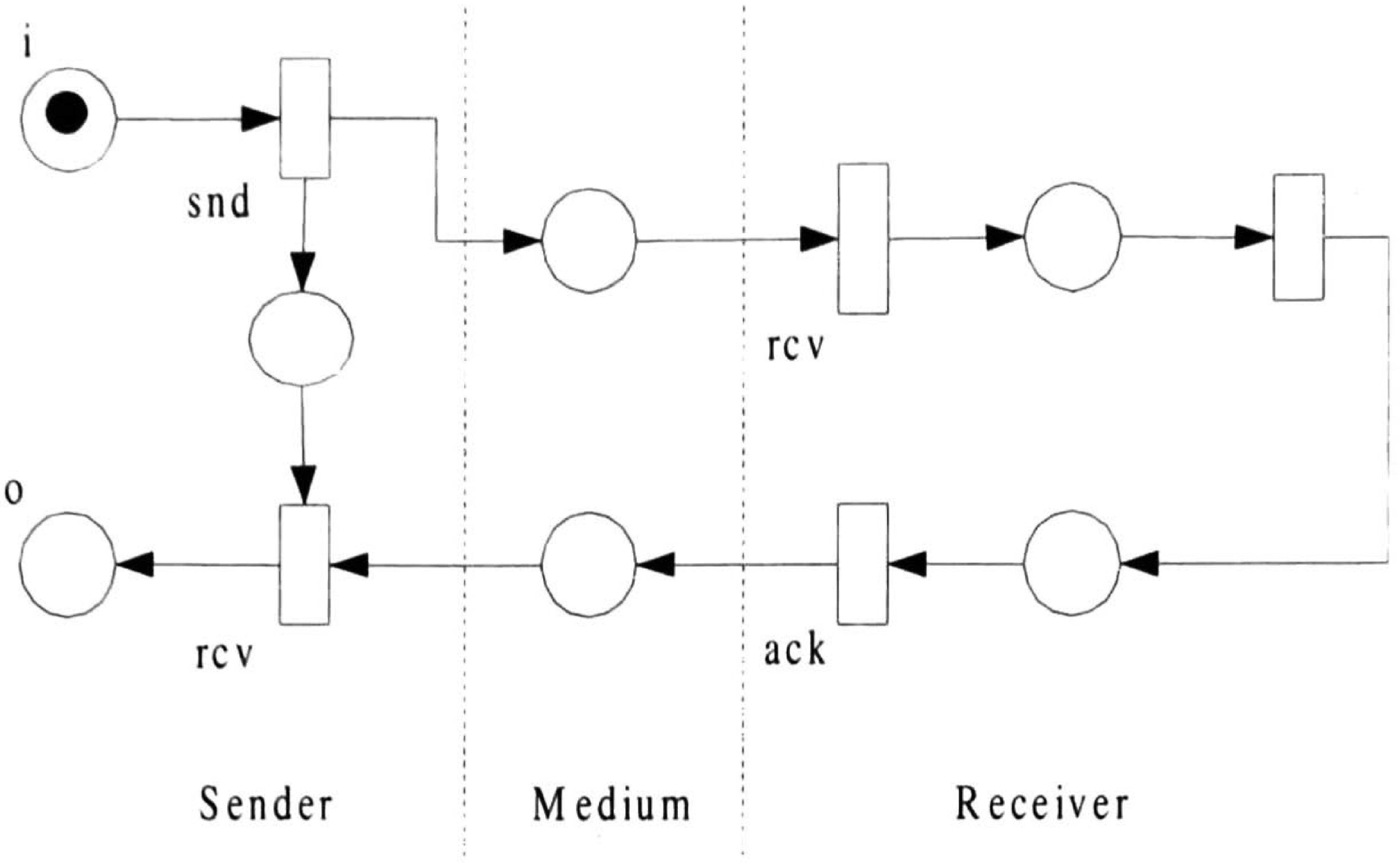


Figure 3.14: The Message Coordination Pattern Net

Definition 3.27 (Messaging Coordination) *Figure 3.14 shows three parts: a sender, a receiver and, a medium. Two transitions in the sender do the work: the transition "snd" put a message on the medium and the transition "rcv" waits for acknowledge. When the transition "rcv" fires, other task can take place.*

Proposition 10 *The Message Coordination Workflow Net is live and bounded.*

Proof. a)Liveness: since the underlying *MPTNet* is a *FCNet* and since every proper siphon includes an initially marked tramp the system is live.

b)Boundedness: let $I = (1111111)$; I is a positive s-invariant of the underlying *MPTNet*. ■

Up to here Workflow Nets were developed for some Workflow patterns. Now, when it is necessary to build a complex business process model, often it is constructed in an incremental way. In some cases, it is constructed small models for the different processes, and then such models are mixed. In other cases it is more convenient to construct a general model, and then refine it by the application of some rules or techniques, in order to reduce the complexity of the modelling job. The convenient way depends on the particular situation.

In the next section, a refinement method for top-down place-refinement construction of Petri Net models is adapted from [Zhou, 1992] to the case of workflow models, in order to avoid the property analysis in the final net.

3.3 Building Petri Net models for Workflow Management

Synthesis methods construct nets systematically such that the desired properties are guaranteed avoiding the analysis process in the final nets. Two kinds of approaches, top-down (refinement of nodes by sub-nets) and bottom-up (fusion of nodes or paths) have been studied in synthesis methods (Jeng and DiCesare, 1992) [Jeng, 1992], [Koh, 1991]. Intuitively, it is clear that the modelling of large concurrent systems demands some kind of modularization to break down the complexity. In

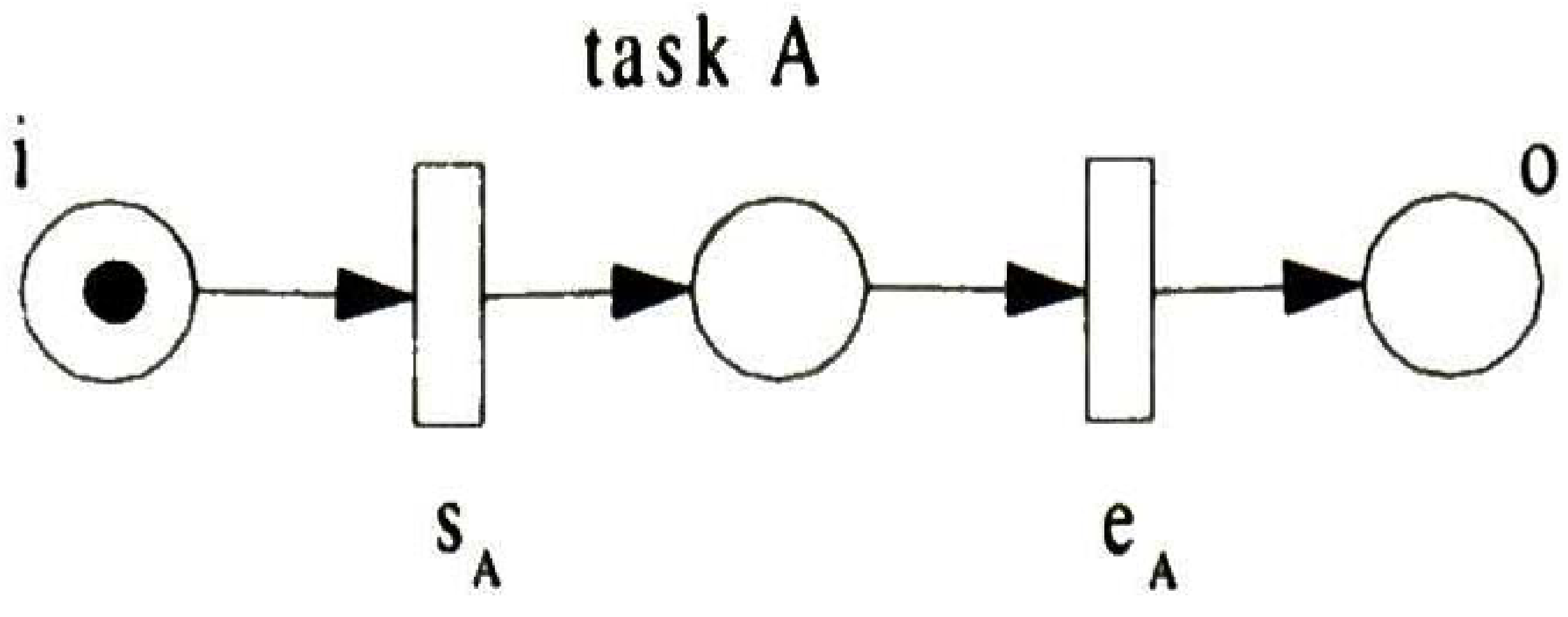


Figure 3.15: The Simple Activity Pattern Net

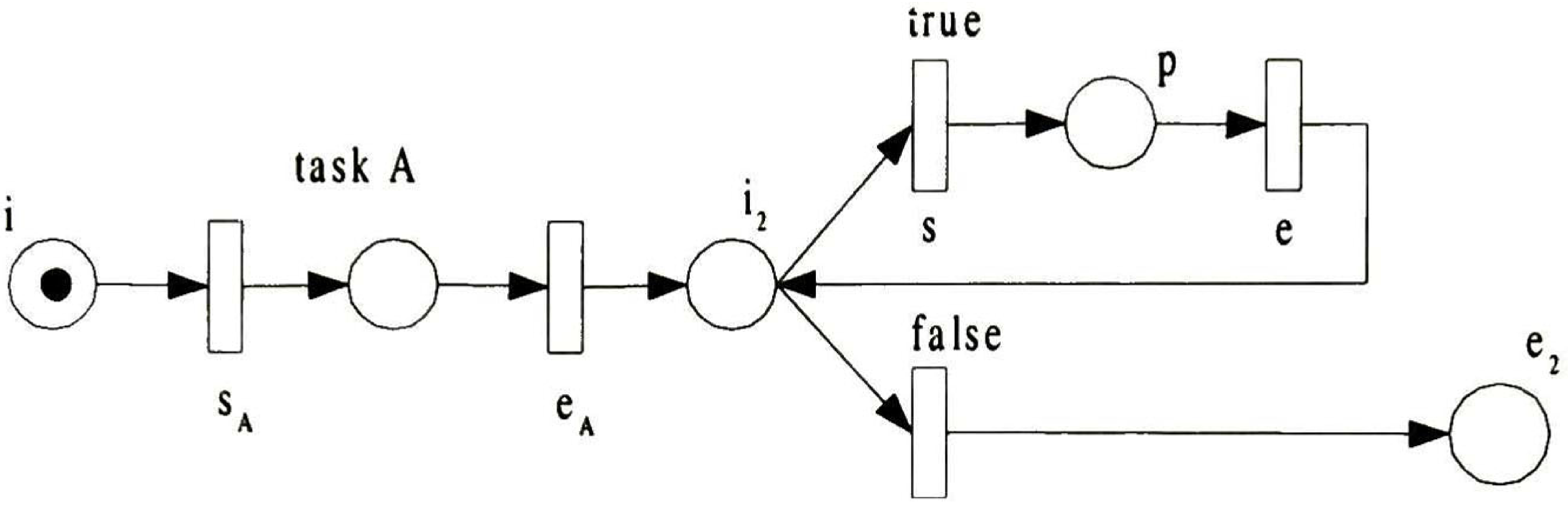


Figure 3.16: The First Step Refinement

this section, it is presented an adaptation of the place refinement method in [Zhou, 1992] to the case of Workflow Nets.

3.3.1 Building live and bounded models for Workflow Management

The place refinement in Petri Nets is a good method for the construction of complex models [Zhou, 1992]. Moreover, it is desired to define some place refinement rules that guarantee good properties in the final refined model. In this way, starting with a simple model, it is possible to construct complex models applying these refinement rules avoiding subsequent analysis.

For example, the Figure 3.15 shows the Workflow Net for the Simple Activity pattern. Now, it is possible to perform a replacement to the place *"o"* substituting it by a subnet, as the Figure 3.16 shows. In this particular case, the final net can perform an Activity A, then it can repeatedly do a certain activity. Finally it can reach its END state.

Intuitively, this refinement rule works as follows: as Figure 3.16 shows, this refinement method replaces a place p in a PTNet by a new PTNet that have an input and an output place. Then the input and output edges for the place p become the input and output edges for the new PTNet. The formal definition is as follows.

Definition 3.28 (p-replacement) *Let (N, μ) be a $MPTNet$, and $(N', \varnothing) \in \mathcal{W}$ such that $N = (T, \partial_-, \partial_+)$, and $N' = (T', \partial'_-, \partial'_+)$, and i' and o' are the input and output places of N' respectively. A* ***p-replacement*** *$p - N'$ where $p \in Nm_\omega(N)$, is the $MPTNet$ $(\bar{N}, \mu)$,where $\bar{N} = (\bar{T}, \bar{\partial}_-, \bar{\partial}_+)$ is defined as follow:*

- $\bar{T} = T \cup T'$

$$\bar{\partial}_-(t) = \begin{cases} \partial_-(t) \backslash np + no'.\ if\ t \in T\ and\ np \in \partial_-(t),\ n \in \mathbb{N} \\ \partial_-(t),\ if\ t \in T\ and\ np \notin \partial_-(t),\ for\ any\ n \in \mathbb{N} \\ \partial'_-(t),\ if\ t \in T' \end{cases}$$

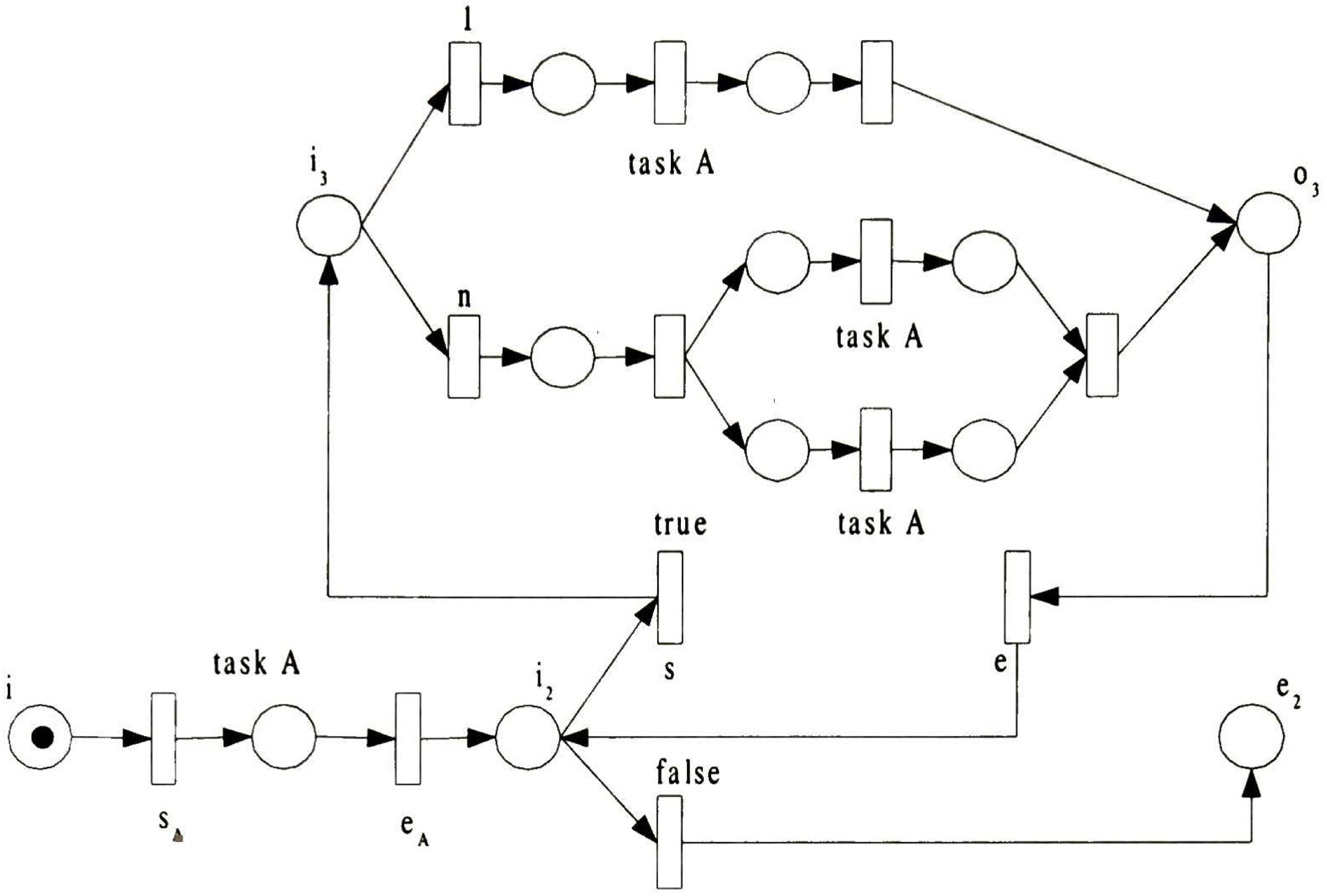


Figure 3.17: The Second Step Refinement

$$-\bar{\partial}_{+}(t) = \begin{cases} \partial_{+}(t)\backslash np + ni' \quad if\, t \in T \; and\; np \in \partial_{+}(t),\; n \in \mathbb{N} \\ \partial_{+}(t),\; if\, t \in T \;\; and\; np \notin \partial_{+}(t) \;\; , for\; any\; n \in \mathbb{N} \\ \partial'_{+}(t),\; if\, t \in T' \end{cases}$$

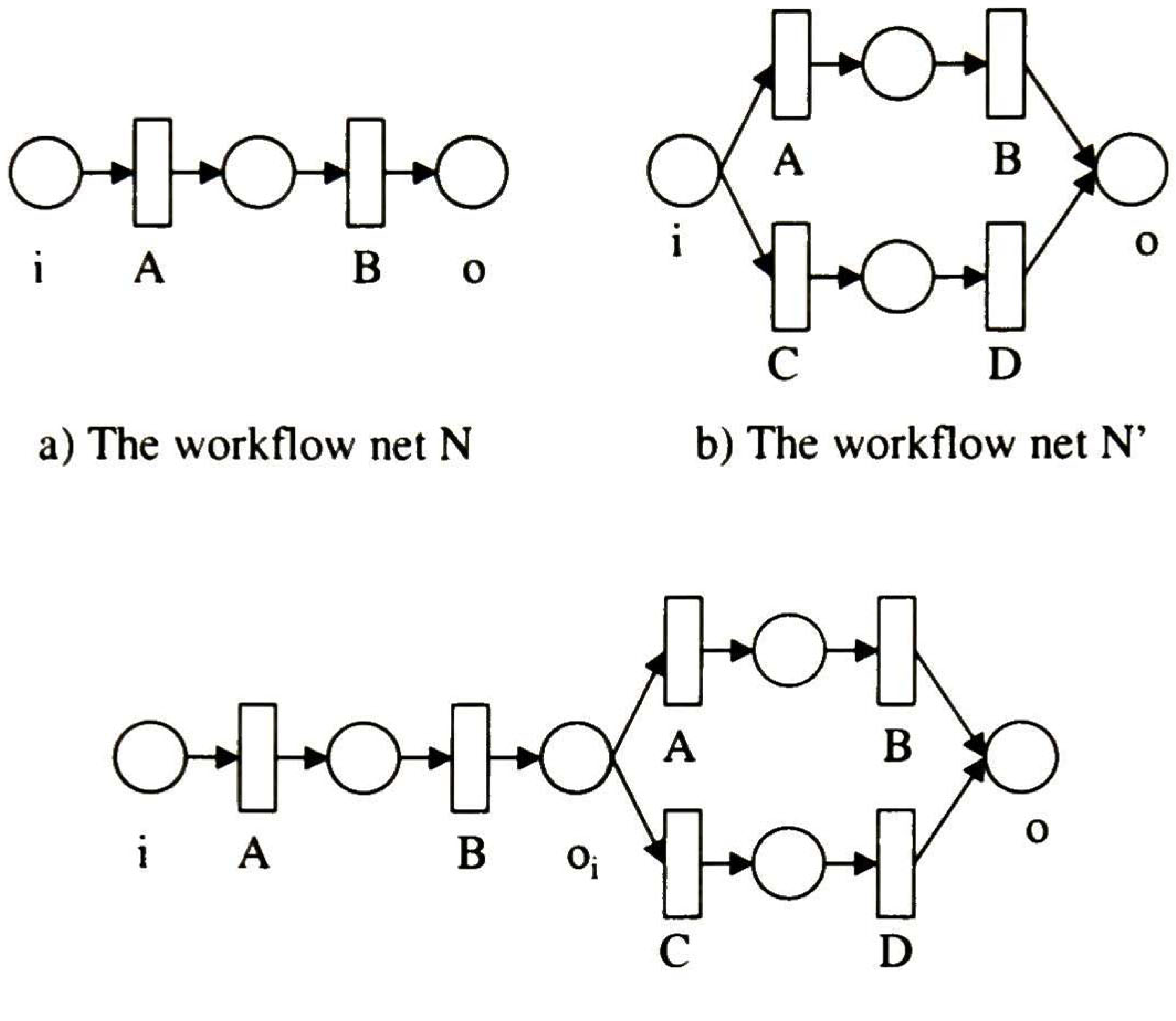
The next theorem characterizes the liveness and soundness on the application of the definition of p-replacement over a net. Intuitively, this theorem means that, given a sound net, if this p-replacement is applied over it using a sound net in the replacement, then the resulting net is always a sound net. The theorem says also, that the inverse case is hold.

Theorem 11 (Soundness characterization of p-replacement) *Let (N, μ) be a MPTNet, let $(N', \varnothing) \in \mathcal{W}$. Let $(\bar{N}, \bar{\mu})$ be the net that results of the p-replacement over (N, μ) using $(\bar{N}, \bar{\mu})$. Then $(\bar{N}, \bar{\mu})$ is live and bounded iff (N, μ) is live and bounded.*

Proof. Sketch: Since every $(N', \mu') \in \mathcal{W}$ is live and bounded, then the soundness condition imposed to the net used in the replacement is hold by this p-replacement. Using the result in [Zhou, 1992] the results follow. ■

Figure 3.17 shows the next step in the refinement of the net in Figure 3.16. The overall refinement is as follows: The Figure 3.15 shows the net which the refinement process was started. Then the place labeled with "*o*" was replaced by the Arbitrary Cycle pattern. Figure 3.16. Then the place labeled with "*p*" was replaced by the Multiple Instances with Synchronization pattern. Figure 3.17.

In this way, the final net, in fact a sound Workflow Net, has the next semantic: after the firing of the Activity A, it is possible to fire repeatedly i-parallel Activity A, where $1 \leq i \leq n$. Such



a) The workflow net N

b) The workflow net N'

c) The sequential composition of N and N'

Figure 3.18: The Sequential Composition of two workflow nets

final net meets the semantic for the Multiple Instances without prior runtime knowledge and its Synchronization pattern (see the Workflow Patterns section, in the previous chapter).

Another two common operations between nets, that result in a kind of refinement, are the sequential composition and the parallel composition of flow nets. They formal definitions are given below.

Definition 3.29 (Sequential Composition of nets) *Let $(N,\{i\}),(N',\{\varnothing\}) \in \mathcal{W}$, where o is the final place of N and i' is the initial place of N'*

Then the sequential composition of $(N,\{i\})$ and $(N',\{i'\})$, denoted $(N,\{i\}) \circledcirc (N',\{\varnothing\})$, is the juxtaposed composition of $(N,\{i\})$ and $(N',\{i'\})$ merging the place o in N with the place i' in N' formally, $(N \otimes N',\{i\})$, where o and i' are only ones public places in N and N' respectively, and $o = i'$.

Example 3.4 (The sequential composition of two nets) *As an example of the sequential composition of nets, see the net depicted in Figure 3.18. Figures 3.18a and b, shows two workflow nets. Finally, Figure 3.18c shows the compound net.*

Definition 3.30 (The Parallel Composition of nets) *Let $(N,\{i\}),(N',\{i'\}) \in \mathcal{W}$, where i and o is the initial and final places of N, respectively, and, i' and o' are the initial and final places of N', respectively.*

The Parallel Composition of $(N,\{i\})$ and $(N',\{i'\})$, denoted $(N,\{i\}) \oplus (N',\{i'\})$, is the juxtaposed composition of $(N,\{i\})$ and $(N',\{i'\})$ merging the places i and i' in N and N' respectively, and merging the places o and o' in N and N', respectively; formally $(N \otimes N',\{2i\})$, where i and o are public places in N, i' and o' are public places in N' and, $i = i'$ and $o = o'$

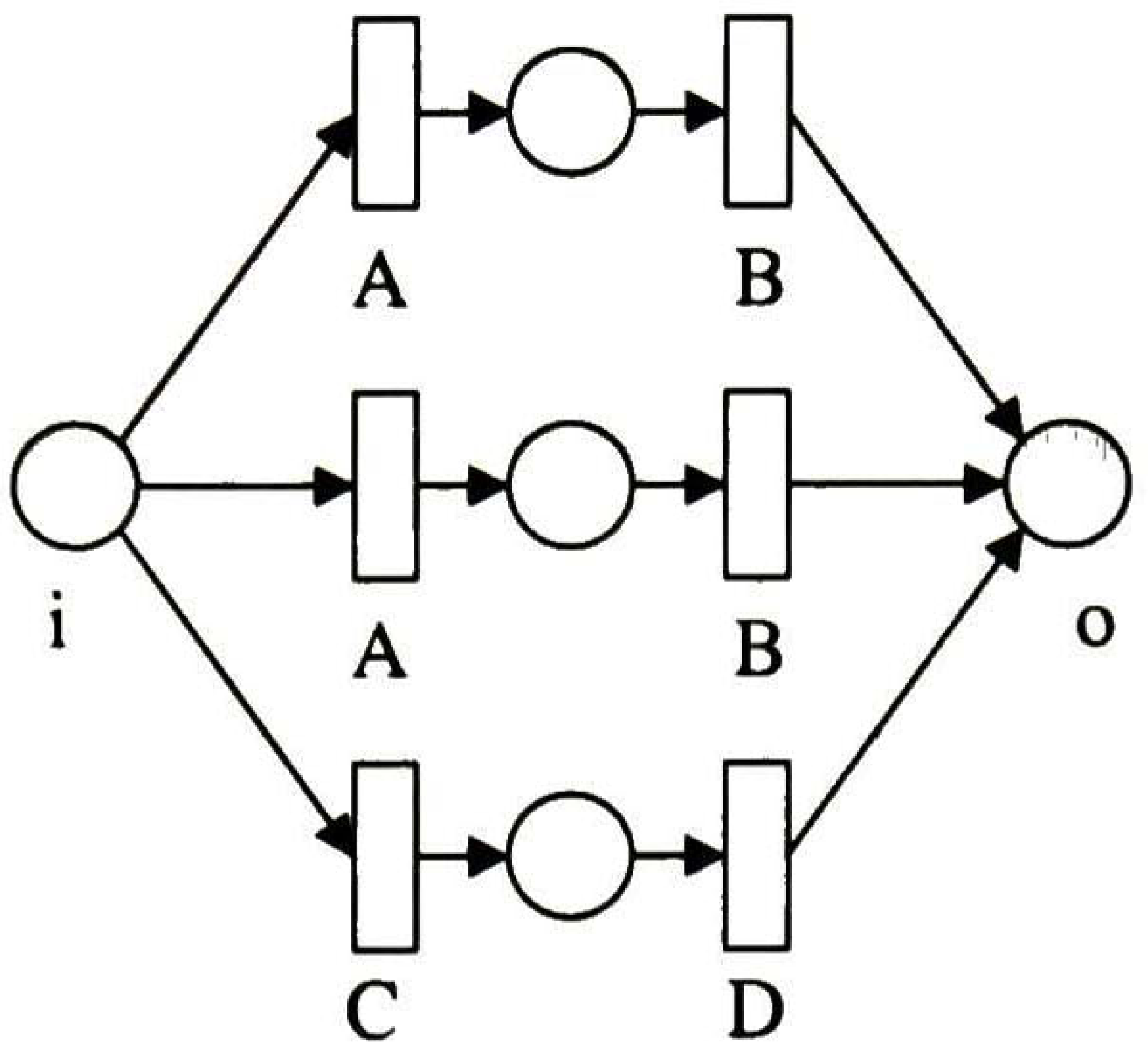


Figure 3.19: The Parallel Composition of workflow nets

Example 3.5 (The parallel composition of two nets) *As an example of the parallel composition of nets, see the net depicted in Figure 3.19. This net is the compound net resulting of the parallel composition of nets depicted in Figures 3.18a and b.*

The next section uses the place refinement method in a more complex application example.

3.3.2 An Application example

As an example of the refinement method in the previous section, this section presents the case of FBN Sports Equipment Company in Luxemburg, a sport and athletic equipment sales company.

MODELLING BUSINESS PROCESSES IN A SPORT EQUIPMENT COMPANY

Process specification. The FBN Sport Equipment Company in Luxemburg manufactures a complete range of sports and athletic equipment. All of its sales are made by Purchase Orders. Since the company receives 80 of its sales orders by mail and fax, it has been determined to have an image scanner in the mail room as in order to enhance the workflow system. The mail room is the point where the workflow starts, and it revises all mails and faxes to the company. The next processes are the way each department take for performing its activities.

Process 3.1 (The mail room activities) *The mail room performs the next activities:*

```
If the documents are a sales_order:
  all documents go to Sales, Finance and Manufacturing in parallel
If the documents are invoices for payment:
  the documents go to Finance
If the documents are addressed to the President:
  the documents go directly to the President Sccretary;
    the Secretary sorts mail for immediate response by President;
If the document is sales leads go to Vice President Sales;
Product inquiries go to Customer Support
If the documents are payments: the documents go to Finance;
```

Process 3.2 (At the Finance Department) *The Finance Department performs the next activities:*

```
If the documents are for a sales_order:
  then check if sales_order is from an existing customer:
    if an existing customer
      then check account limits:
        if Account limit OK
          then release to Manufacturing
            else refer to next manager level:
              if sales_order approved
                then release to Manufacturing;
                else decline sales_order
                  route to Sales Representative and to Manufacturing;
      else // if not an existing customer
        then request credit_check:
          if credit_check OK
            then release to manufacturing
            else decline sales_order
              Route to sales representative and to Manufacturing
```

Process 3.3 (At the Manufacturing Deparment.) *The Manufacturing department performs the next activities:*

```
If sales_order is released and can be shipped
  1. prepare product order
  2. issue shipping instructions and prepare forms
  3. notify sales, finance and customer service on ship date
  4. adjust inventory levels.
If sales_order is not released
  place sales_order into suspend mode.
If sales_order is released but lack of product:
  Then notify sales representative of partial ship plan
  Freeze parts and stock for 24 hours
  Suspend order for 24 hours
When the sales_order is released and the partial shipment program has been approved by
the customer then Manufacturing will:
  Build and ship to the partial program
  Notify Finance on shipping
  Expedite missing parts from suppliers
  Suspend the sales_order until the parts arrive
  Complete the sales_order
  Issue notification to Finance, Sales and Customer Service
```

Each different department performs a different set of activities. In this company processes start at the Mail Room. The Mail Room checks all incoming messages and deliver them to its respectively department for its processing.

The refinement process for the Mail Room Department starts with the Simple Activity pattern, Figure 3.15. The Mail Room's process shows four different classes of incoming messages: sales order messages, invoices for payment messages, messages addressed to the President, and payments

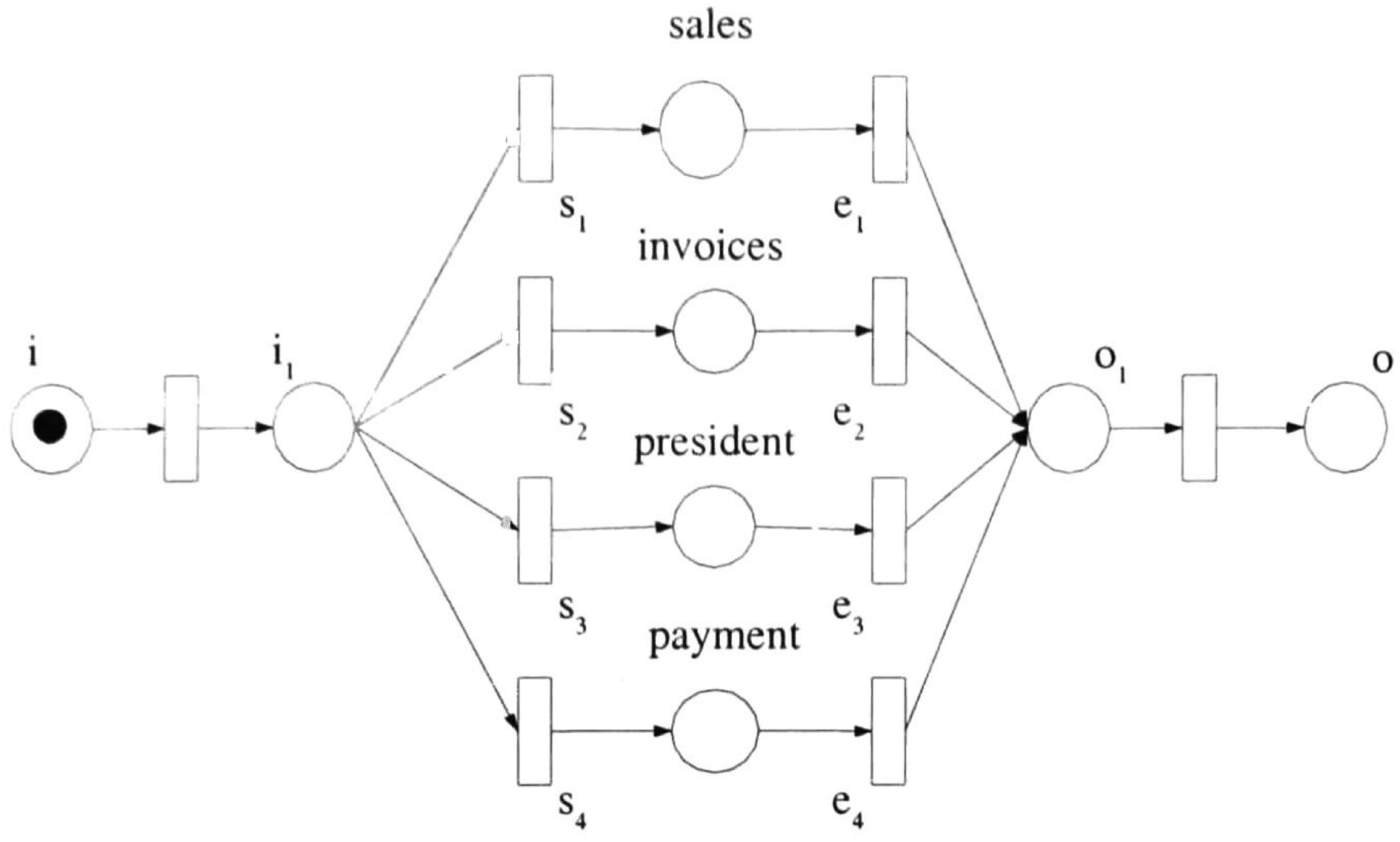


Figure 3.20: The First Step Refinement at the Mails Room

messages. So, the central place replaced by the Simple Choice pattern with four branches, one for each different class of incoming messages. This is shown in the Figure 3.20.

Now, if the incoming message is a sales order, the process indicates that the document must goes to Sales, Finance and Manufacturing departments in parallel. So, the next step is to replace the place "sales" on the Figure 3.20 with the 3-AND-Split pattern and its Synchronization, as shows the Figure 3.21. The rest of the refinement process for this department could be build in a similar way.

At the Finance department, using this refinement method, it is possible to construct a Workflow net like as the net depicted in in Figure 3.7. This workflow net is the one net used in the introduction to the section: Workflow Patterns as Workflow Nets.

The previous example was fetched from [WfMC-TC-1016].

3.4 Conclusions

In this chapter, basic notions on Petri Nets were presented. Also Workflow patterns are translated to Workflow Nets -a special class of Petri Nets-. Some Workflow patterns have a straightforward representation into Petri Nets concepts. Others have a more elaborated construction.

Also a synthesis method for the gradual construction of elaborated workflow schemes was presented. The synthesis methods allow to construct complex workflow schemes without the subsequent analysis.

Unfortunately, there are some important aspects in Workflow Management that have no direct equivalence into Petri Nets concepts. One of these concepts is the dynamic change problem. In this problem, some Workflow schemes must change their structure in a controlled way. These concepts are not allowed on the traditional Petri Nets formalism. However, there are some extensions for Petri Nets that allow to support successfully this dynamic change problem. The next chapter is

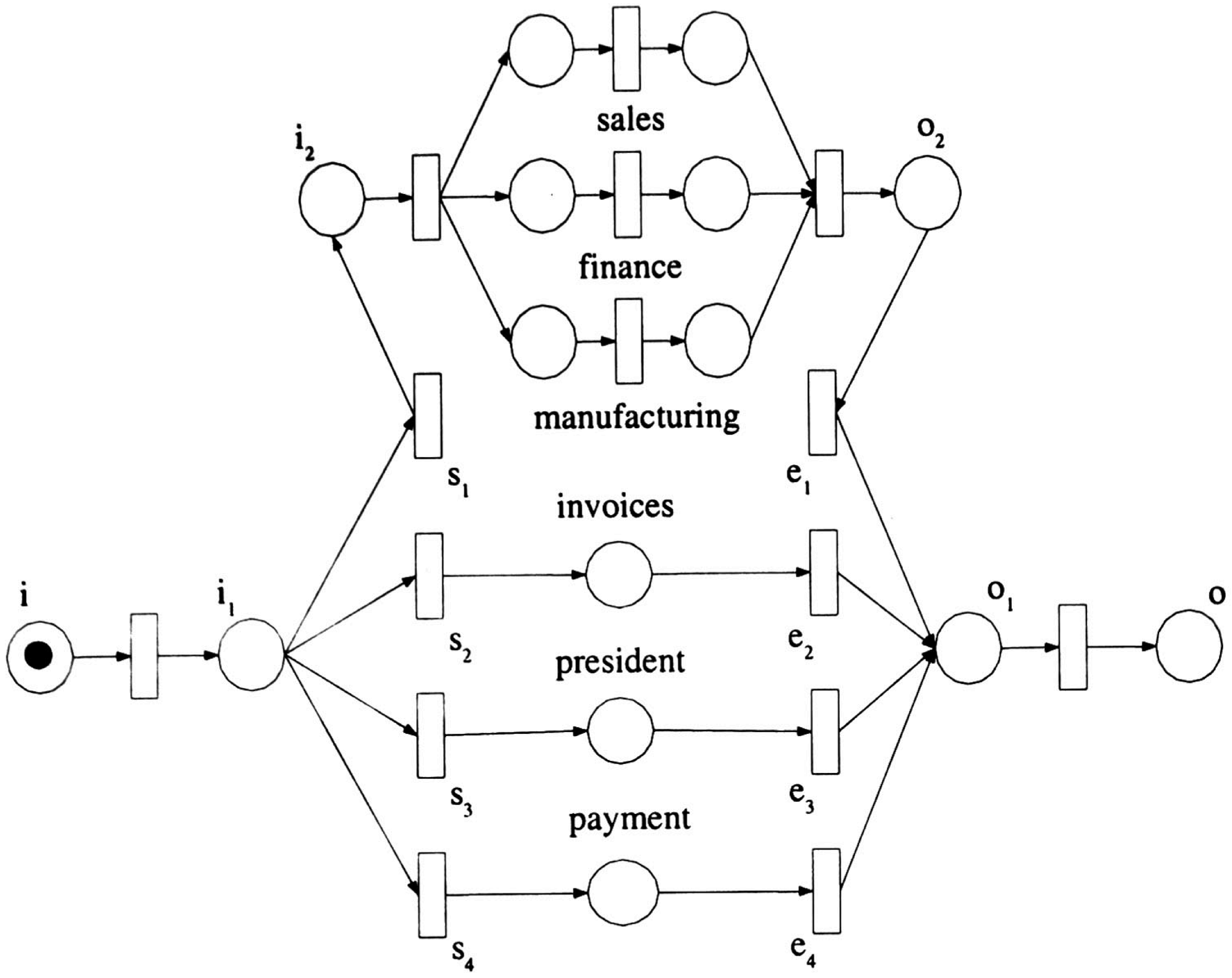


Figure 3.21: The Second Setp Refinement at the Mails Room

related on such class of problem and a proposed solution.

Chapter 4

RW-Nets for Dynamic changes in Workflow Management Systems

This chapter addresses the Dynamic change problem in Workflow Management Systems. An overview of approaches for workflow changes is presented. Dynamic nets from [Buscemi, 2001], and [Asperti, 1996] are introduced, and how they support the dynamic change is shown Finally, Rewriting Nets -an extension of Dynamic Nets- are proposed as modeling formalism to support the modifying mechanisms to cope with the dynamic change problem.

4.1 The Dynamic change problem

Business processes are subject to unexpected changes: market changes, customer changes, new business strategies, etc., usually result into a dynamic change.

The dynamic change is an important issue within Workflow Management systems. Several works on the matter point out that this problem is difficult to solve [Agostini, 1998], [Aalst, 2000 IS], [Ellis, 1995], [Casati, 1996], [Han, 1998].

In these systems, can mainly occur two kind of changes [Ellis, 1995], [Aalst, 2000 WP 50] :

- 1) Ad-hoc changes, and
- 2) Evolutionary changes.

Ad-hoc changes occur on individual cases or on a selected group of them, while evolutionary changes have structural impact, i.e., an evolution change into the workflow process affects all new cases from certain moment forth. This change is commonly the result of a new business strategy, reengineering efforts or a permanent alteration of external conditions. A workflow process definition resulting from an evolutionary change is often called a new "version" of a workflow process.

A wide known classification of dynamic change within workflow management is depicted on Figure 4.1. As it is shown, the dynamic change can be as follows:

- "Extend": a refinement of an actual task in the workflow process; Figure 4.2a shows the extension (or refinement) of an activity A into two activities A1 and A2.
- "Replace": a substitution of an existing task by a new one in the workflow process; Figure 4.2b shows the replacement of an activity A by an activity B; and
- "Re-order": a new causal order in the actual set of tasks over the workflow process; Figure 4.2c shows two sequential activities A and B, that must change its sequential relationship.

On the other hand, the change can be as follows:

- "Individual": the change affects only one case or a selected group of them, or
- "Structural": the change affects overall workflow process structure, and all new cases that arrive to the system.

On the "Individual" branch, there exist the next two cases:

- Entry-time: the time of the change is determined;
- On-the-fly: the time of the change is not determined; it may occur at any stage in the process.

On the "Structural" branch, there exist the next cases:

- Restart: a workflow process actually in execution must be abruptly restarted;
- Version: a new version for an existing workflow process must be present in the system, and

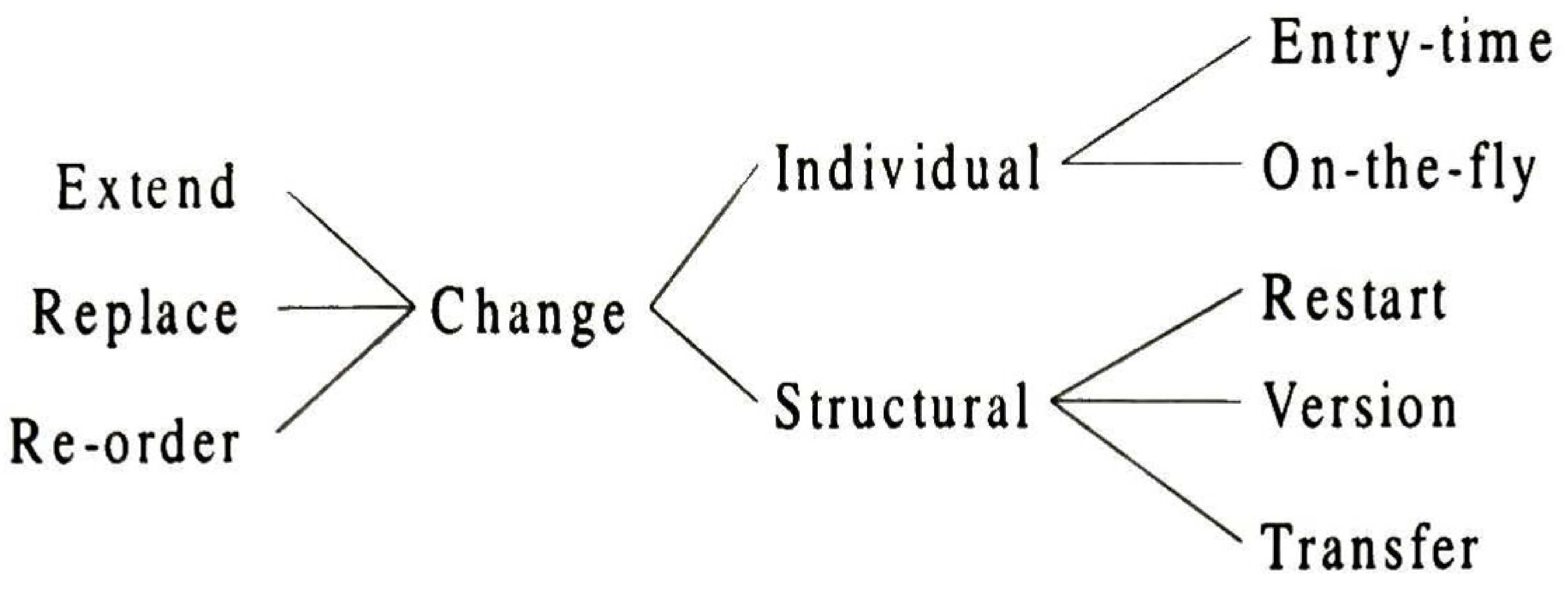


Figure 4.1: Change Classification

- Transfer: all possible states (or activities) in the actual workflow process must be present in the next version of such workflow process; Figure 4.2d shows an activity A that must be present in the next version of the workflow process.

Since in this work Petri Nets are used as modelling formalism of workflow systems, the dynamic change problem can be reduced to the problem depicted on the Figure 4.3; it shows two Petri nets called Workflow process definition; the change from the workflow process on the left to the workflow process on the right, and reciprocally, is the goal to achieve in this chapter.

It outcomes obvious that it is necessary to explore the flexibility on Petri Nets. Such flexibility on Petri Nets has been explored in previous works [Buscemi, 2001], [Asperti, 1996]. In these works, it was proposed a taxonomy of High Level Petri Nets that increase its flexibility lessening its restrictions.

4.2 Related work on Adaptive Workflow

In [Badouel, 1998], a class of high level Petri nets called Reconfigurable nets was presented. These nets can change its structure dynamically by the rewriting of some of their components. Basically, a Reconfigurable net is a Petri Net with a set of modification rules over its places. A modification rule is a function that maps places to places and it works as follows: the places to be mapped must be places in use on the current net and its image must be "invisible places" for the system at the time of the change. This High level nets can be used to change the "flow" in a workflow process definition.

In [Casati, 1996], an approach to the dynamic workflow change based on primitives of evolution was presented. First, a formal definition of a Workflow process is provided. Then some transformation primitives, like a programming-style, of a given workflow are given.

In [Aalst, 2001 WP 51], the concept of dynamic change within workflow, its variations and flavors were presented. Both, ad-hoc and evolutionary changes are undercover in the workflow. The presented approach uses the notion of "versions" of workflow -a property that allows to a new workflow process definition to have all states present on the old workflow version-. The goal in this work is to determine when is possible to migrate cases from an old workflow process version to a new one.

In [Aalst, 1999 UGA], "generic process models" -a kind of process model, based on Petri nets, that can be "instantiated" with a "specific" process model in a certain moment- was presented.

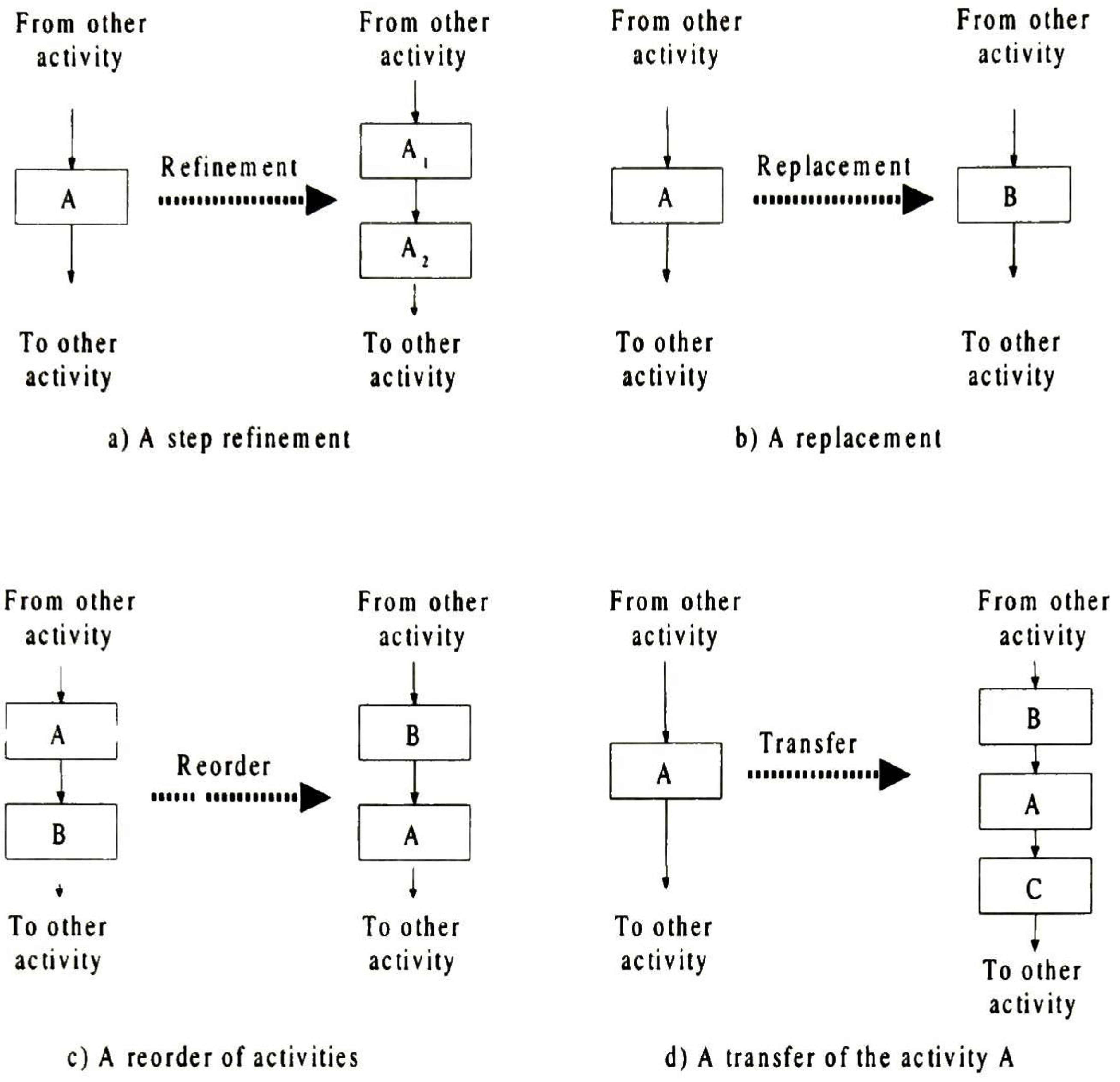


Figure 4.2: Concepts on Dynamic Change

The "dynamic changes" were supported "instantiating" the same "generic process model" with many -finite- "specific" process model.

In [Aalst, 2000 WP 50], an approach based on inheritance, a concept of Object Oriented Programing, applied to Petri nets was presented. Two mechanisms of inheritance: protocol inheritance and projection inheritance were explored. Using this notions was possible to decide when one workflow process is a "subclass" to another workflow process, based on these two inheritance mechanisms. A set of "well-defined" transformation rules was also presented, to migrate from one workflow process to a new one using a rule transformation and , preserving inheritance was possible. Using branching bisimilarity, it is possible to divide workflow process into equivalence classes of inheritance.

In [Jiang, 2001], a Colored Petri Net with changeable structure was presented. Two mechanisms for the structural change: change-by-modification and change-by-composition, were suggested The first one was related to the net structure changing by the modification of some net structure elements, while the second one was related to the net structure changing by the composition of the current net to another net.

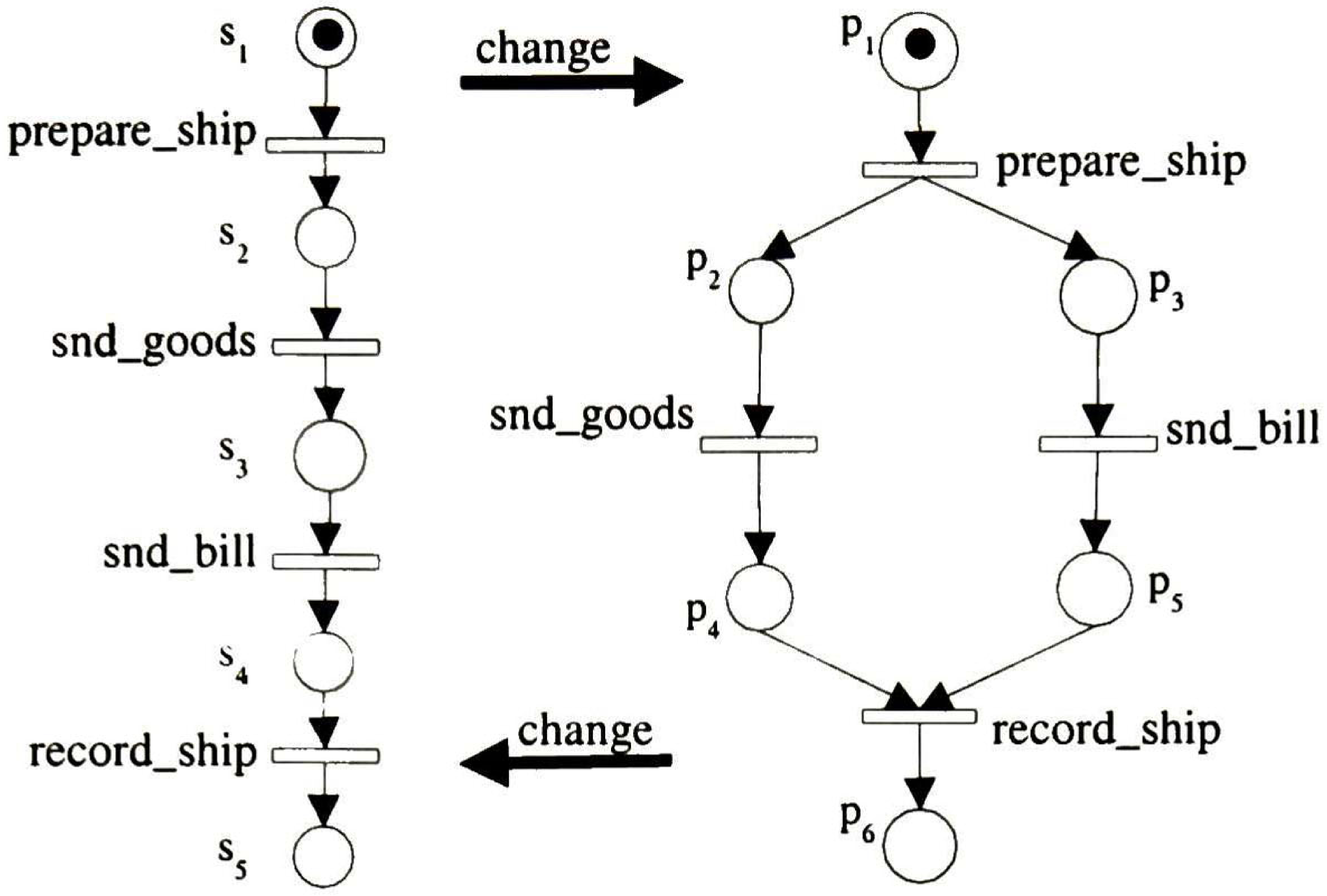


Figure 4.3: The Dynamic Change Problem

4.3 RWNets to support dynamic change in Workflow

In order to support the dynamic change within Workflow Management, it was necessary to increase the flexibility in Petri Nets. There are some works related on. For example in [Asperti, 1996] and [Buscemi, 2001] a hierarchy of nets that increase the flexibility in Petri Nets were suggested. Mobile Nets (also called Reconfigurable Nets), a class of nets that have an additional feature: the post of a transition can be delivered to different sets of places was suggested. Dynamic Nets -a class of high level nets that can increase its structure during the firing of a transition- were also presented.

In this thesis some of these concepts are hold and extend, in order to support the dynamic change in Workflow Management.

4.3.1 Colored Nets

Colored Petri Nets (CPN) were proposed by K. Jensen in 1981 for the modeling of very large systems [Jensen,1981]. CPN models have a high level of abstraction and they are very condensed.

In CPN, the tokens into the places could be individually identified, i.e., each token has a symbolic identity. This identity is called the "color" of the token . Also, in CPN, a transition could be able to fire respect to a specific color or set of them. The colored tokens travel across the net by the action of the transitions, transforming them into other colored tokens.

The main concepts on CPN are illustrated in Figure 4.4. It represents the Dinners 3-Philosophers problem.

In short, a number n of philosophers are sat at a round table. At the right and left hands of each philosopher there is a fork. When a philosopher wants to eat, then he takes the forks at his right and left hands and then he goes to the eating room. When the philosopher finishes to eat, then he gives back the forks. Then, other philosopher can eat.

The colored net have three places: one place for philosophers, one place for forks, and one place for the eating room. Also, there are two transitions in the net: one transition for when a philosopher

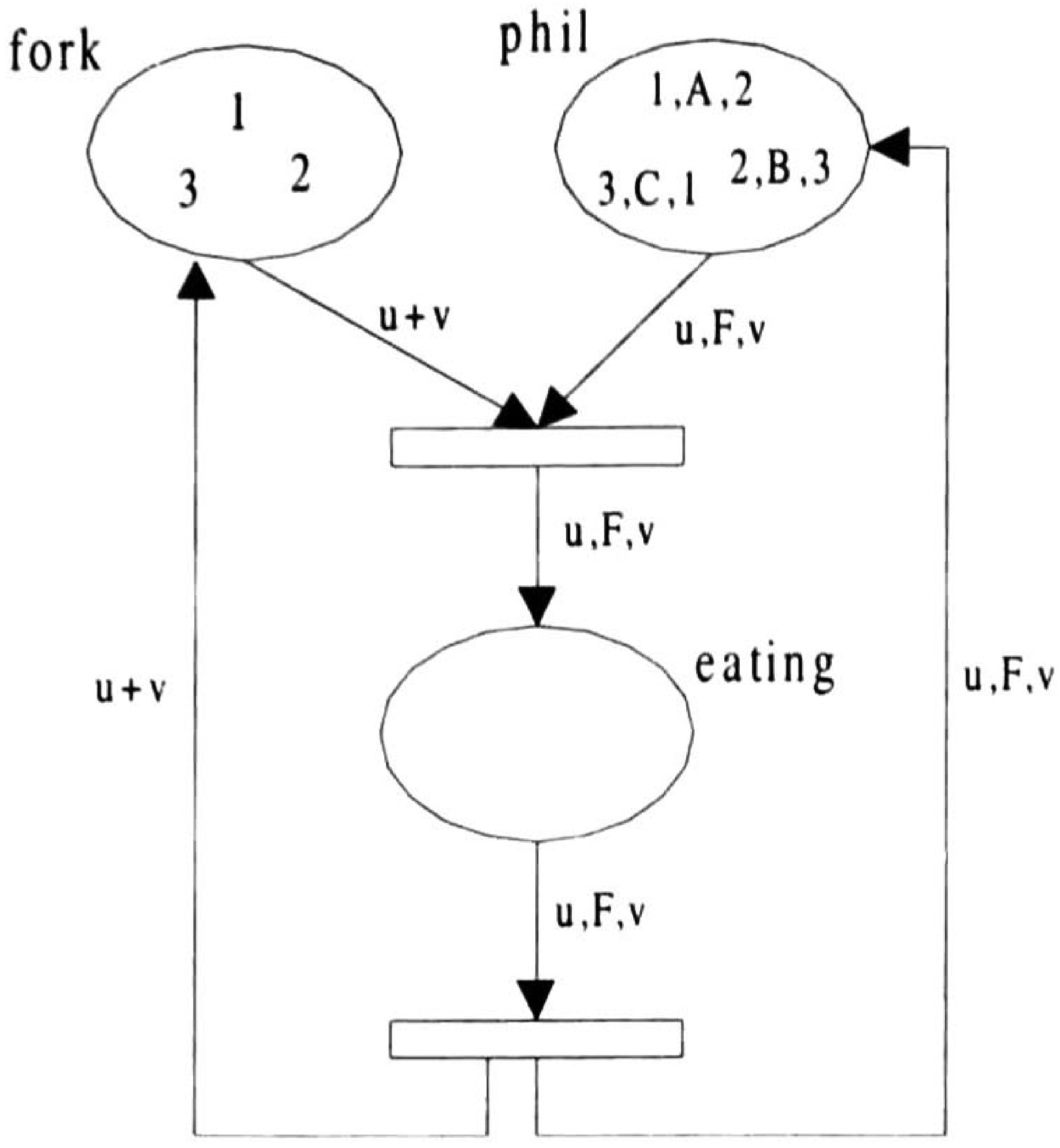


Figure 4.4: The Dinners Philosophers Problem Colored Net

wants to eat and accordingly goes to dining room and other transition for when a philosopher ends to eat. The net shows the case when the number of philosophers is three.

As it is shown, the tokens are tuple of names. Also, the edges are labeled with a tuple of names. For example, the tuple "u,F,v" in an edge means that such tuple could be "instantiated" with any tuple with length three.

The firing of a transition in these nets occurs when a value is given to the edge label in the pre set of a transition. This assignation of values defines a function from the edge labels to the color names into the places; such function is called a "binding" function. For example, in Figure 4.4, it is possible to fire the upper transition with the "binding" function $b : u \rightarrow 1, v \rightarrow 2, F \rightarrow A$. Now, in order to get the post set as result of the firing of this transition, is applied this binding function to its post set. In this particular case, the colored token "$1, A, 2$" is gotten into the "eating" place, meaning that the philosopher A is now eating. Since the problem is reduced to the case of three philosophers, any other philosopher can go to eat at this moment, as it was expected. At this state, the lower transition could be fired to return to the initial state.[1]

Definition 4.1 (Colour Set) *The set of colored tokens C_P over a set P is defined as the set of finite sequences over P, that is:*

$$C_P = \{v_0, ..., v_n | v_i \in P, 0 \leq i \leq n, n \in \mathbb{N}\}$$

It will be used $\bar{u}, \bar{v}$ ranging over C_P. The length of a tuple $\bar{v} \in C_P$, is defined as $|\bar{v}| = |v_0, ..., v_n| = n + 1$, and the selection of the i-th element is $\pi_i(\bar{v}) = v_i, 0 \leq i \leq n$.
For a $v_0, ..., v_n = \bar{v} \in C_P$, $[\bar{v}]$ denotes the set $\{v_0, ..., v_n\}$ rather than multiset..

[1] Note: the edge label "$u + v$" in the pre set of the upper transition means that there are two edges between the "fork" place and this transition: one edge with the "u" label and one edge with the "v" label. This is only for notation conveniences.

Definition 4.2 (Coloured Net) *A Colored Net (CNet) is a bipartite graph with two classes of nodes: places and transitions, represented by a tuple* $N = (T, \partial_-, \partial_+)$, *where* T *is a finite set of transitions,* $\partial_+, \partial_- : T \rightarrow \mu(Nm_\omega \times C_{Nm})$ *are the pre and post functions. Also,* $\partial_-(t)$, *is often called the* **pre set** *of* t, *and* $\partial_+(t)$ *is often called the* **post set** *of* t.

Definition 4.3 (Marked CNet) *A pair* (N, μ_0), *where* N *is a* $CNet$, *and* $\mu_0 \in \mu(Nm_\omega \times C_{Nm})$ *is called a Marked CNet (MCNet). The element* μ_0 *is called the initial marking.*

Definition 4.4 (Bounded names, Binding) *The set of bounded names for a* $\mu' \in \mu(Nm_\omega \times C_{Nm})$ *is denoted:*

$$rc(\mu') = \cup\{[\bar{x}] | \exists p : (p, \bar{x}) \in \mu')$$

A ***binding*** *for* μ' *is a function* $b : rc(\mu') \rightarrow Nm$. *It is defined* $\mu'\langle b\rangle = \{(p, b(\bar{c}) | (p, \bar{c}) \in \mu')\}$, *where* $b(\bar{c} = v_o, ..., v_n) = b(v_0), ..., b(v_n)$.

Definition 4.5 (Transition enabling) *Let* (N, μ) *be a* $MCNet$. *Let* $t \in T$, *it is said that* t *is enabled at* μ *under the binding* b, *denoted* $(N, \mu)[t\rangle_b$ *iff there exist a binding* $b : rc(\partial_-(t)) \rightarrow Nm$, *such that* $\partial_-(t)\ \langle b\rangle \subseteq \mu$.

Remark 3 *A Labeled* $CNet$ *can be obtained associating a transition labeling function* $l : T \rightarrow L$ *to any* $MCNet$ (N, μ) *as in* $MPTNet$.

Definition 4.6 (Firing Rule) *The firing rule for a labeled* $MCNet$, $\_[\_\rangle_b\_ \subseteq MCNets \times L \times MCNets$, *is the smallest substitutive relation generated by:*

$$(N, \mu)[t\rangle_b \Longrightarrow (N, \mu), [l(t)\rangle_b, (N, \mu \backslash \partial_-(t)\ \langle b\rangle + \partial_+(t)\ \langle b\rangle)$$

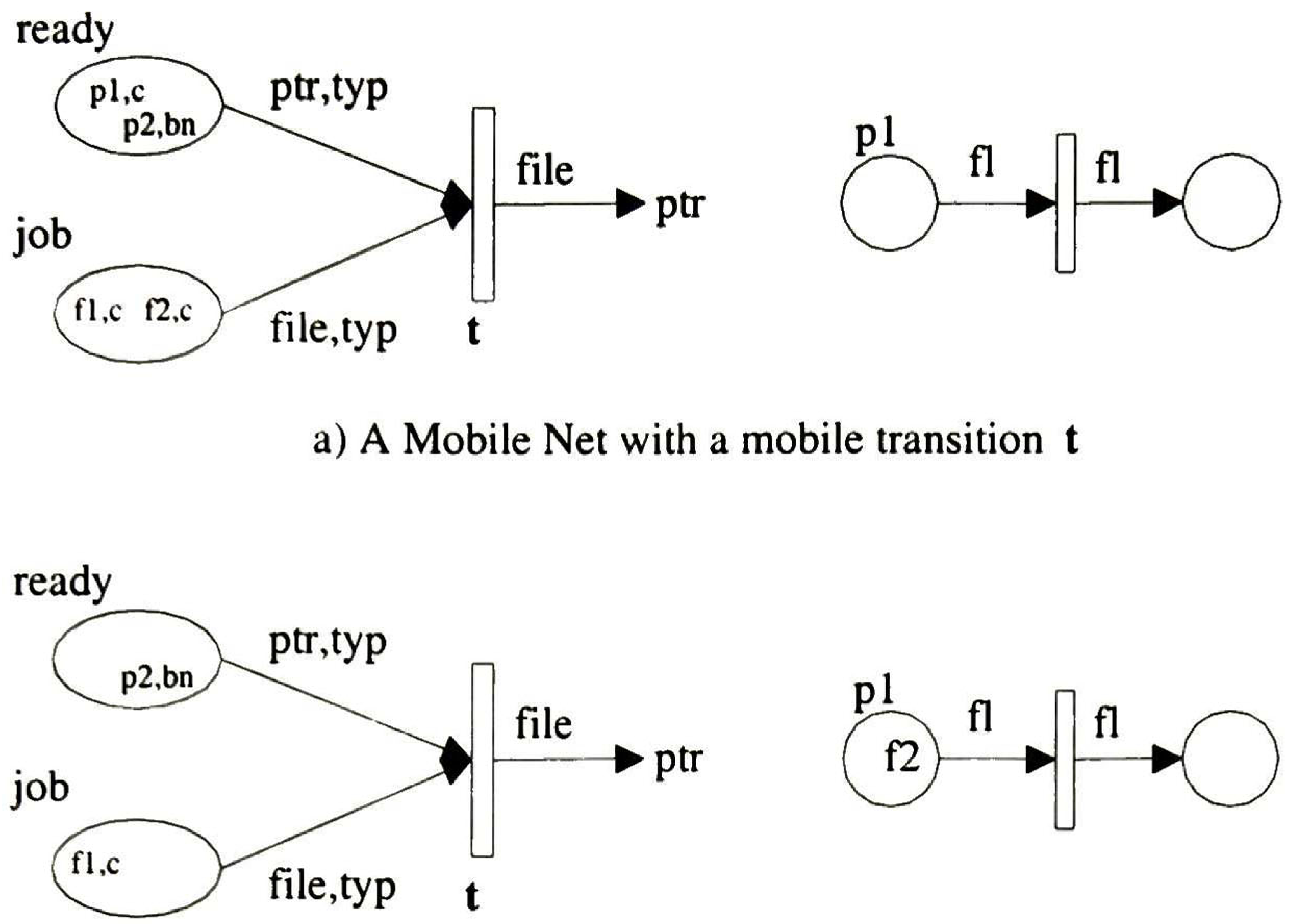
where b *is a binding for* $\partial_-(t)$.

Example 4.1 (A Colored Net) *As example consider the next net:* $N = \{fork(u{+}v), phil(u, F, v)\} \rhd \{eating(u, F, v)\}, \{eating(u, F, v)\} \rhd \{fork(u{+}v), phil(u, F, v)\}$, *and* $\mu_0 = \{fork(1{+}2{+}3), phil(1, A, 2 + 2, B, 3 + 3, C, 1)\}$, *then* $(N, \mu_0) \in MCNets$ *is the net on Figure 4.4.*

In this section was explained the major concepts on Colored Nets. For further details see [Jensen,1981]

4.3.2 Mobile Nets

Mobile Nets, also called Reconfigurable Nets, are a generalization of Colored Nets. In them, the post set of a transition could be delivered to different set of places, i.e., the post set of a transition is "mobile" or reconfigurable. It is achieved using the colored tokens in the pre set of a transition for determining where the post set must be delivered. For example, on Figure 4.5a, it is shown a transition with input places "ready" and "job" and where its output place must be determined in order to the colored tokens linked to its pre set. In this case, Figure 4.5b shows the firing of the "mobile" transition using the binding $b : (ptr, typ \rightarrow p1, c), (file, typ \rightarrow f2, c)$, producing as consequence that the place "$p1$" receives a colored token "$f2$" The net depicted on the Figure 4.5a and b meets partially the behavior for a printer spooler [Forunet, 1995].



a) A Mobile Net with a mobile transition **t**

b) The Mobile Net after the firing transition **t** with the binding:
b:{ (ptr,typ → p1,c) , (file,typ → f2,c) }

Figure 4.5: The Partial Printer Spooler Mobile Net

Definition 4.7 (Mobile Net) *A Mobile Net (MNet) is a bipartite graph with two classes of nodes: places and transitions, represented by a tuple* $N = (T, \partial_-, \partial_+)$*, where* T *is a finite set of transitions,* $\partial_- : T \rightarrow \mu(Nm_\omega \times C_{Nm})$*, and* $\partial_+ : T \longrightarrow \mu(Nm_\omega \times C_{Nm_\omega})$ *are the pre and post functions, respectively. Remember that:* $C_{Nm_\omega} = C_{Nm \cup \omega}$*. Also,* $\partial - (t)$ *is often called the* **pre set** *of* t*, and* $\partial + (t)$ *is often called the* **post set** *of* t*.*

Definition 4.8 (Marked MNet) *A pair* (N, μ)*, where* $N = (T, \partial_-, \partial_+)$ *is a MNet, and* $\mu_0 \in \mu(Nm_\omega \times C_{Nm_\omega})$ *is a Marked MNet (MMNet).*

As it is shown, the pre set of a transition in Mobile Nets has the same form as the pre set of a transition in Colored Nets. Accordingly, the notion of binding and enabling transition are the same for both Colored Nets and Mobile Nets. However, in Mobile Nets the binding is applied to both elements p and $\bar{c}$ for each pair $(p, \bar{c}) \in \partial(t)$, for any transition t, rather than in Colored Nets, where a binding b is applied only to the color element $\bar{c}$ of each pair $(p, \bar{c})$ in the post set of a transition. This is exactly the point where the post function of a transition in Mobile Nets could be delivered to different set of places, that is, since the binding function is applied to both elements in the pair $(p, \bar{c})$, then it is possible that the binding function replace the element p, meaning that the place that must receive the post set of the transition was changed. That is the meaning post set of a transition in Mobile Nets is mobile rather than static, as in Colored Nets. The formal definition is below.

Definition 4.9 (Substituting $\mu \langle\langle b \rangle\rangle$ for Mobile Net) *Let* b *be a binding as in Colored Nets.*

For $\mu \in \mu(Nm_\omega \times C_{Nm_\omega})$, let $\mu\langle\langle b\rangle\rangle = \{(b_\omega(p), b_\omega(\bar{c}) | (p,\bar{c}) \in \mu)\}$, where $b_\omega = b + id_\omega$, and $b_\omega(\bar{c} = v_0, ..., v_n) = b_\omega(v_0), ..., b_\omega(v_n)$. The id_ω function is the identity function over ω.

Definition 4.10 (Transition enabling) *Let (N,μ) be a MMNet. Let $t \in T$, then t is enabled at μ under a binding b, denoted $(N,\mu)[t\rangle_b$ iff there exist a binding $b : rc(\partial_-(t)) \rightarrow Nm_\omega$, such that $\partial_-(t)\langle b\rangle \subseteq \mu$.*

Remark 4 *A Labeled MNet can be obtained associating a transition labeling function $l : T \rightarrow L$, to any MMNets (N,μ) as in MPTNets.*

Definition 4.11 (Firing Rule) *The firing rule for a labeled MNet, $\_[\_\rangle_b\_ \subseteq MLNet \times L \times MLNet$, is the smallest substitutive relation generated by:*

$$(N,\mu)[t\rangle_b \Longrightarrow (N,\mu), [l(t)\rangle_b, (N, \mu \backslash \partial_-(t)\langle b\rangle + \partial_+(t)\langle\langle b\rangle\rangle)$$

where b is a binding for $\partial_-(t)$.

Despite the feature of Mobile Nets to deliver the post set of a transition to a different set of places, Mobile Nets have a static structure, i.e., given a Mobile Net, it is possible to obtain an equivalent Colored Net in a simple way. In fact, Colored Nets and Mobile Nets are just compact representations of Petri Nets.

In the next section, Dynamic Nets -a class of High Level Nets that can increase its places and transitions in every firing of its transitions- are presented.

4.3.3 Dynamic Nets

Dynamic Nets can increase their places and transitions during the firing of a transition. It is achieved allowing to the pre set of a transition to be a net, rather than solely a set of places.

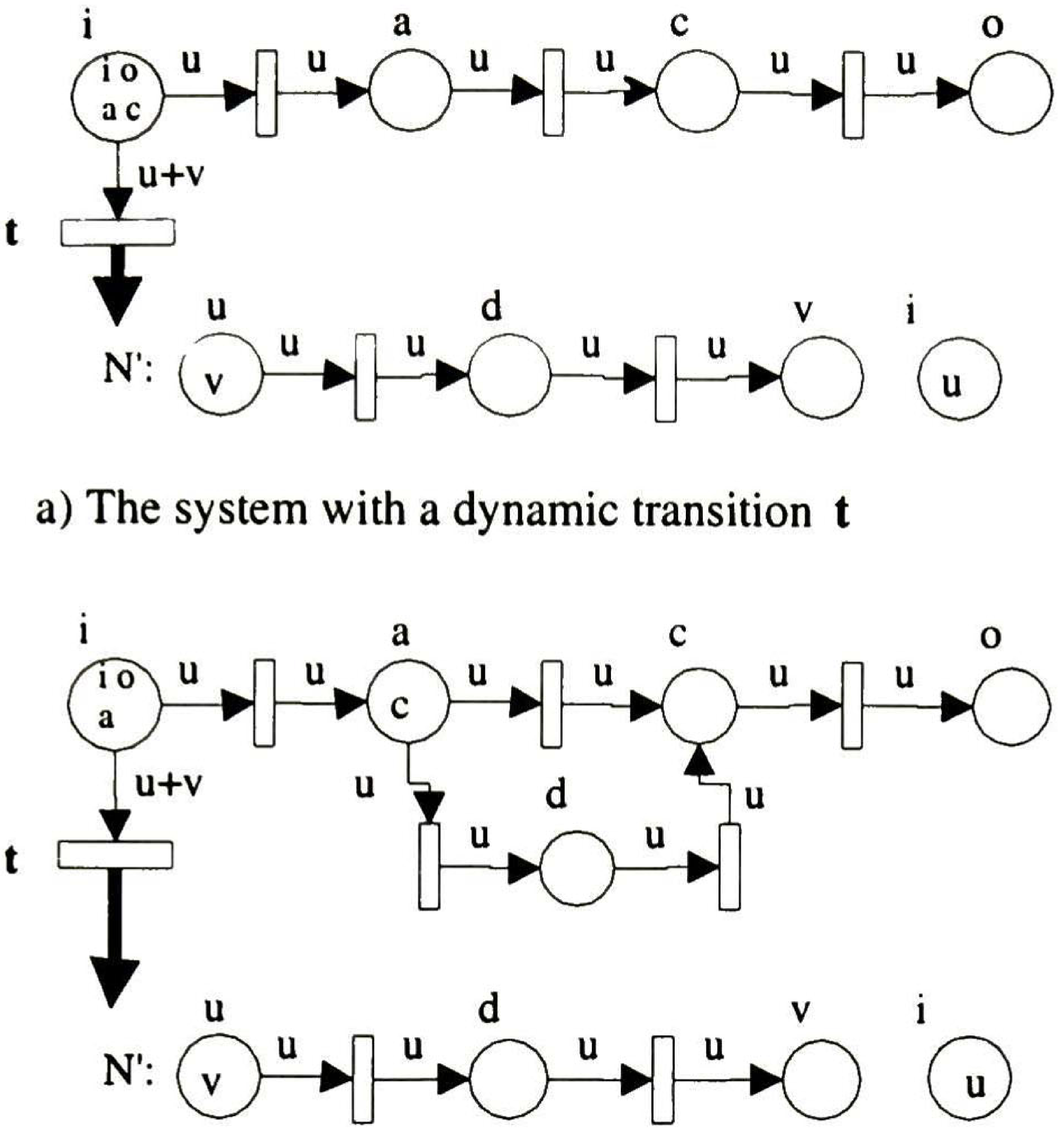
The pre set of a transition in Dynamic nets has the same form as the pre set of a transition in Mobile Nets. So, the enabling of a transition and the binding function have the same meaning. But now, the post set of a transition could be a new net definition.

Intuitively, given a enabled transition t, when it fires, the binding function is applied to its post set, in order to produce the new parametric net. The binding function, maybe, replace several place names; if these place names are public places in both nets, the current net and the new one, then they must be merged.

Obviously, at this stage, the colored tokens linked to the pre set of the transition have been removed; then the nets are merged using the juxtaposed composition of nets defined in the Chapter 2.

As an example, Figure 4.6a shows a net that has a transition that has a new net as post set (see the transition t). Figure 4.6b, also shows this net after the firing of such a transition. Now the net structure changes. On Figure 4.6a, the transition is fired using the binding function $b : (u \rightarrow a), (v \rightarrow b)$. Finally, using the Juxtaposed Composition, the final net looks as the one depicted in Figure 4.6b.

The formal definitions for Dynamic Nets are given below.



a) The system with a dynamic transition **t**

b) The system after the firing of **t** with b:{(u,a),(v,c)}

Figure 4.6: A Dynamic Net

Definition 4.12 (Marked Dynamic Nets) *Let $MDNets$ be the set defined recursively as follows:*
a) A $MPTNet$ is a $MDNet$.

b)The pair (N, μ) such that $N = (T, \partial_-, \partial_+)$, where T is a finite set of transitions, $\partial_- : T \rightarrow (Nm_\omega \times C_{Nm})$ is the pre function, $\partial_+ : T \rightarrow MDNets$ is the post function, and $\mu \in \mu(Nm_\omega \times C_{Nm_\omega})$ is a $MDNet$.

Given a $t \in T$, $\partial_-(t)$ is often called the **pre set** *of t, and $\partial_+(t)$ is often called the* **post set** *of t.*

Definition 4.13 (Subnet in Dynamic Nets) *Let (N, μ) be a $MDNet$. Then a MDNet (N', μ') is a* **subnet** *of (N, μ), denoted $(N', \mu') \subseteq (N, \mu)$, iff $T' \subseteq T$, $\partial'_- \subseteq \partial_-|_{T'}$, $\mu' \subseteq \mu$, and recursively $\partial'_+(t') \subseteq \partial_+(t')$ for all $t' \in T'$*

As the definition of $MDNets$ shows, the pre set of transitions in Dynamic Nets has a similar form as in Colored Nets, but now the post set of a transition could be a new definition of a $MDNets$, with new places and new transitions.

The notion of isomorphism for Dynamic Nets, is a generalization of the isomorphism for $MPTNets$, since the post set of a transition in Dynamic Nets is a generalization of the post set of a transition in $MPTNets$. The definition of isomorphism for $MDNets$ is given below.

Remark 5 *Given two functions g and h, the disjoint coproduct of them, denoted $g + h$, is the function defined by both g or h. For example, if $g : (u \rightarrow 1), (v \rightarrow 2)$, and $h : (w \rightarrow 3)$, then $g + h : (u \rightarrow 1), (v \rightarrow 2), (w \rightarrow 3)$.*

Definition 4.14 (Isomorphism for MDNets) *Let $(N, \mu), (N', \mu')$ be two MDNets, such that $N = (T, \partial_-, \partial_+)$, and $N' = (T', \partial'_-, \partial'_+)$. Then (N, μ), and (N', μ') are isomorphic up to $\alpha -$ conversion, denoted $(N, \mu) \cong (N', \mu')$ if, there exist a pair of isomorphisms $\langle f_t : T \rightarrow T', f_p : \omega \rightarrow \omega \rangle$, such that:*

a) $(id_{Nm} + f_p \times id_{Nm} + f_p)(\mu) = \mu'$,

b) $\forall t \in T$, $\partial'_- \circ f_t(t) = (id_{Nm} + f_p \times id_{Nm}) \circ \partial_-(t)$, and

c) Recursively, $\partial_+(t) \cong \partial'_+(f_t(t))$, $\forall t \in T$

Note that the private places in nets are only ones relevant for isomorphism, as in Petri Nets.

Definition 4.15 (Transition enabling) *Let (N, μ) be a MDNet. Let $t \in T$, t is enabled at μ under a binding b, denoted $(N, \mu)[t\rangle_b$ iff there is a binding $b : rc(\partial_-(t)) \rightarrow Nm_\omega$, such that $\partial_-(t) \langle b \rangle \subseteq \mu$.*

The enabling of a transition in this class of nets, is the same as in Colored Nets. But now, the binding function is used also to define the new net in the post set of a transition. The formal definition is given below

Definition 4.16 (Sustitution for DNets) *Let b be a binding as in Colored Nets. For a MDNet (Y, μ), such that $Y = (T, \partial_-, \partial_+)$, it is defined:*

$$(Y, \mu) \langle\langle\langle b \rangle\rangle\rangle = ((T, \partial'_- : (\forall t \in T : \partial_-(t) \langle\langle b \rangle\rangle), \partial'_+ : (\forall t \in T \ : \textit{recursively}, \ \partial_+(t) \langle\langle\langle b \rangle\rangle\rangle)), \mu \langle\langle b \rangle\rangle)$$

that is, $(Y, \mu) \langle\langle\langle b \rangle\rangle\rangle$ denotes the net after applying the substitution $\langle\langle b \rangle\rangle$ as in Mobile Nets, for both the pre set of all transitions in Y and the marking μ, and recursively applying this renaming, understand $\langle\langle\langle b \rangle\rangle\rangle$, to the post set of all transitions in Y

Remark 6 *A Labeled Dynamic Net can be obtained associating a transition labeling function $l : T \rightarrow L$ to any MDNet (N, μ), as in MPTNets.*

Definition 4.17 (Firing Rule) *The firing rule for a Labeled MDNet $\_[\_\rangle_b\_ \subseteq MDNets \times L \times MDNets$ is the smallest substitutive relation generated by:*

$$(N, \mu)[t\rangle_b \Longrightarrow (N, \mu), [l(t)\rangle_b, (N, \mu \backslash \partial_-(t) \langle b \rangle) \otimes \partial_+(t) \langle\langle\langle b \rangle\rangle\rangle)$$

where b is a binding for $\partial_-(t)$. It is denoted $(N, \mu)[t\rangle_b(N', \mu')$, where $(N', \mu') = (N, \mu \backslash \partial_-(t) \langle b \rangle) \otimes \partial_+(t) \langle\langle\langle b \rangle\rangle\rangle)$. The operation $\otimes$ is the juxtaposed composition of nets defined in Chapter 2.

A *MLDNet* allow to modify the structure of a net; however the change is always incremental; thus successive changes grow the size of the model. Moreover, when a transition is fired, all the transitions in its post set are always new. So, for example, it is no possible to "rewrite" a pair of sequential transitions in such way that they have a parallel relationship.

So it is necessary a more general mechanism that allows to "rewrite" in some way a part of the system structure, preserving some important information, such as, transition, edges, places, causal relationships, etc.

One evident way is to generalize the pre set of a transition in such way that it allows to consume a subnet of the current net structure and to produce a new subnet, in the parametric order of the consumed subnet. In this way, it is possible to define a rewriting mechanism for Petri Nets that preserves the structure, increases it, and reduces it.

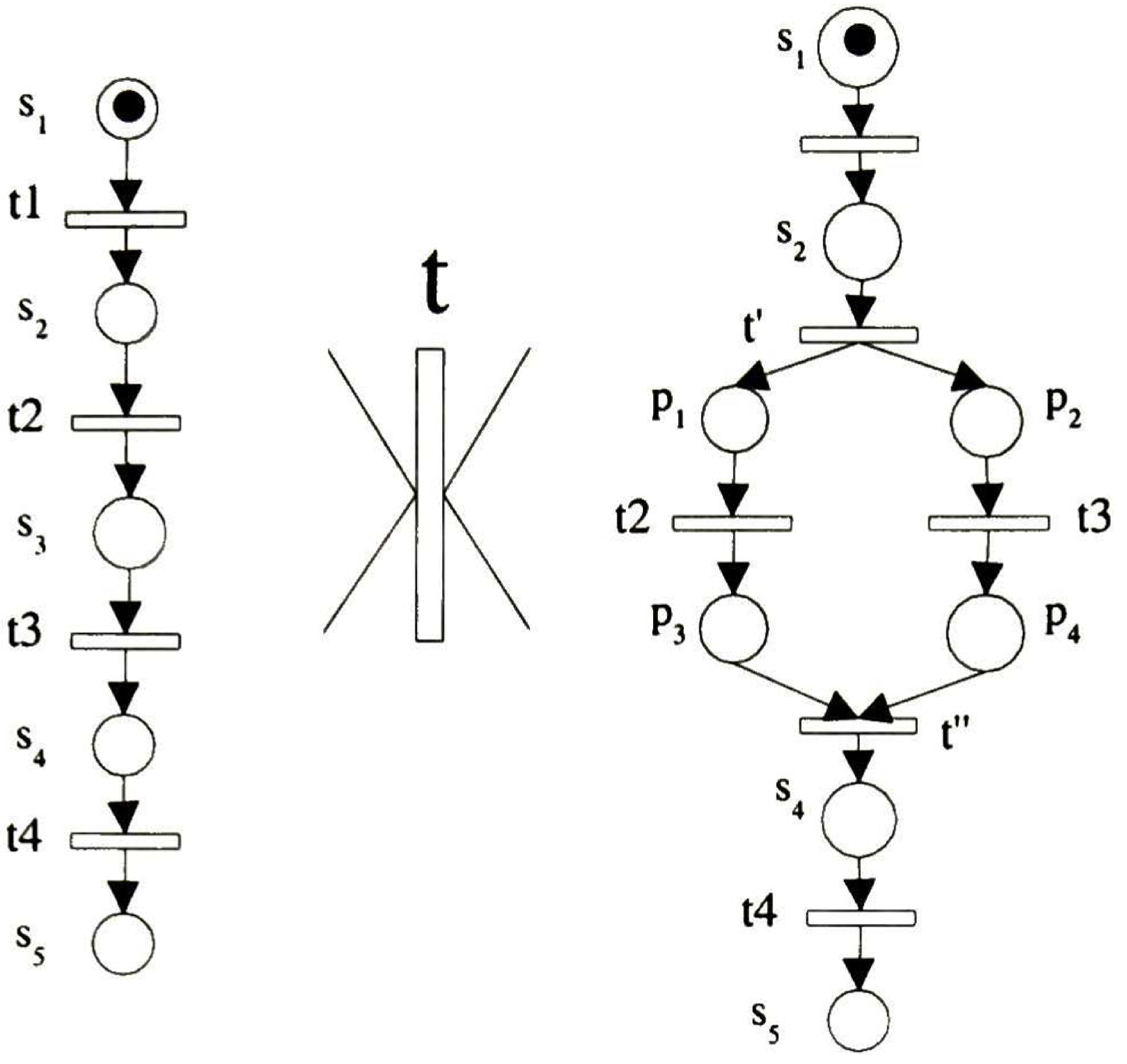


Figure 4.7: A Rewriting Net

4.3.4 RW-Nets

Dynamic Nets offer some flexibility for Petri Nets. However, they have some constraints. For example, it is not possible to obtain the parallel representation of a given pair of sequential transitions.

Dynamic Nets generalize the pre set of a transition; following this approach, Figure 4.7 shows a transition, in which, the pre set is a net, rather than a set of places. This transition could "consume" a subnet -in this particular case, four transitions in sequence-, and then produce a new "parametric" net, where the two central transitions are executed in parallel.

In the rest of this section Rewriting Nets are presented, a class of High Level Nets that rewrite part of its structure at the firing of its transition.

Definition 4.18 (Marked Rewrite Nets) *Let $MRWNets$ be the set defined recursively as follows:*

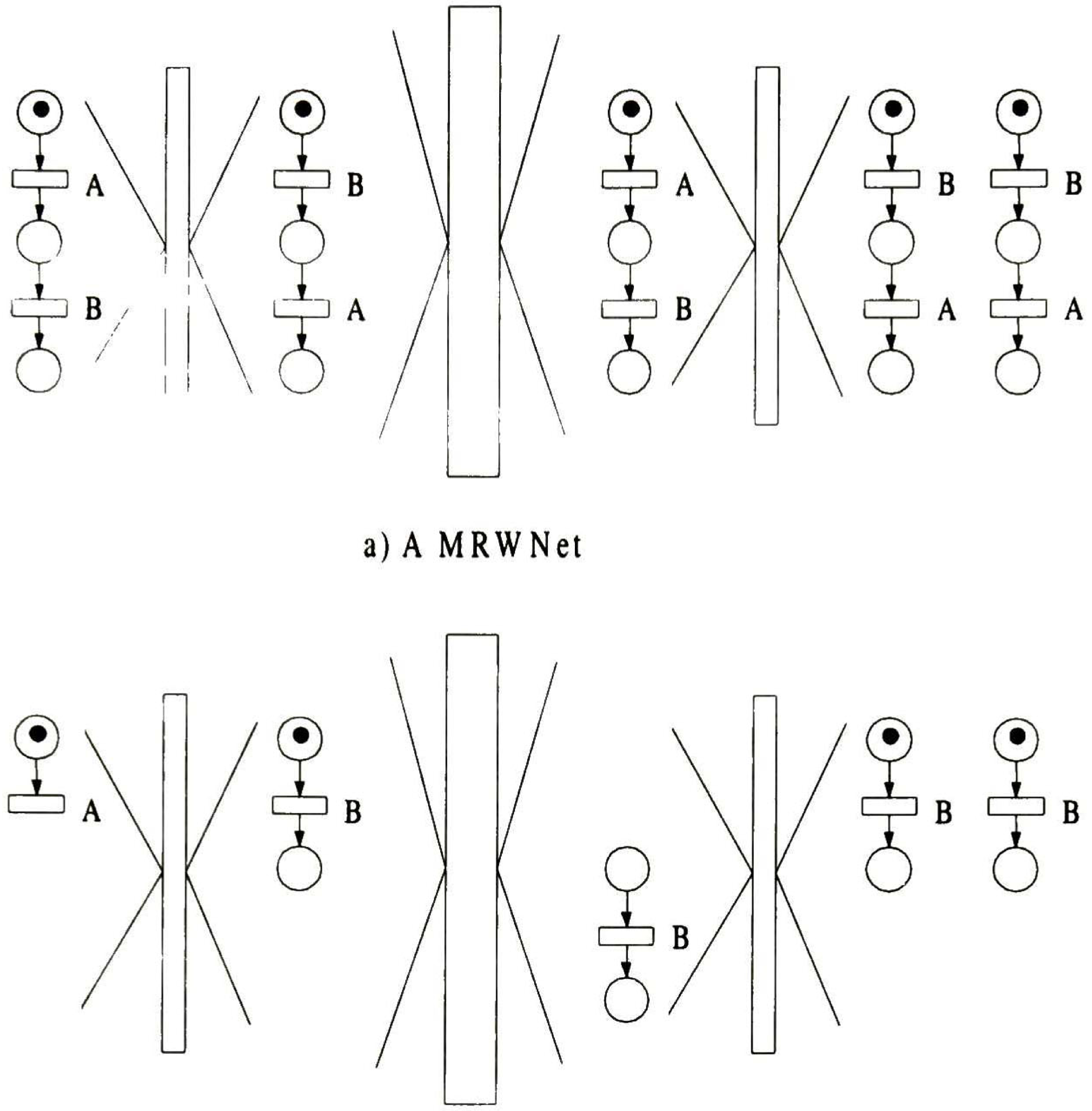
a) A $MDNet$ is a $MRWNet$,

*b) The pair (N, μ), such that $N = (T, \partial_-, \partial_+)$, where T is a finite set of transitions, $\partial_-, \partial_+ : T \rightarrow MRWNets$ are pre and post functions, respectively, and $\mu \in MRWNet$ is a $MRWNet$. Given a $t \in T$, $\partial_-(t)$ is often called the **pre set** of t, and $\partial_+(t)$ is often called the **post set** of t.*

Definition 4.19 (Subnet in MRWNets) *Let $(N, \mu) \in MRWNets$, such that $N = (T, \partial_-, \partial_+)$. The pair $(N', \mu') \in MRWNets$, where $N' = (T', \partial'_-, \partial'_+)$, is a subnet of (N, μ), denoted $(N', \mu') \subseteq (N, \mu)$, iff:*

$T' \subseteq T$, $\partial'_- \subseteq \partial_-|_{T'}$, $\partial'_+ \subseteq \partial_+|_{T'}$, and recursively $\mu' \subseteq \mu$, $\partial'_i(t') \subseteq \partial_i(t'), i = -, +$, $\forall t' \in T'$

Since both, the pre and post set of a transition in $RWNets$, the element could be nets, then in order to two $RWNets$ be isomorphic, it is necessary that they have a isomorphic structure at the top level and at all of their subsequent levels.



a) A MRWNet

b) A Subnet of the above net

Figure 4.8: The Subnet concept in RWNets

For example, Figure 4.8b shows a *RWNet* that is subnet of the *RWNet* depicted in Figure 4.8a.

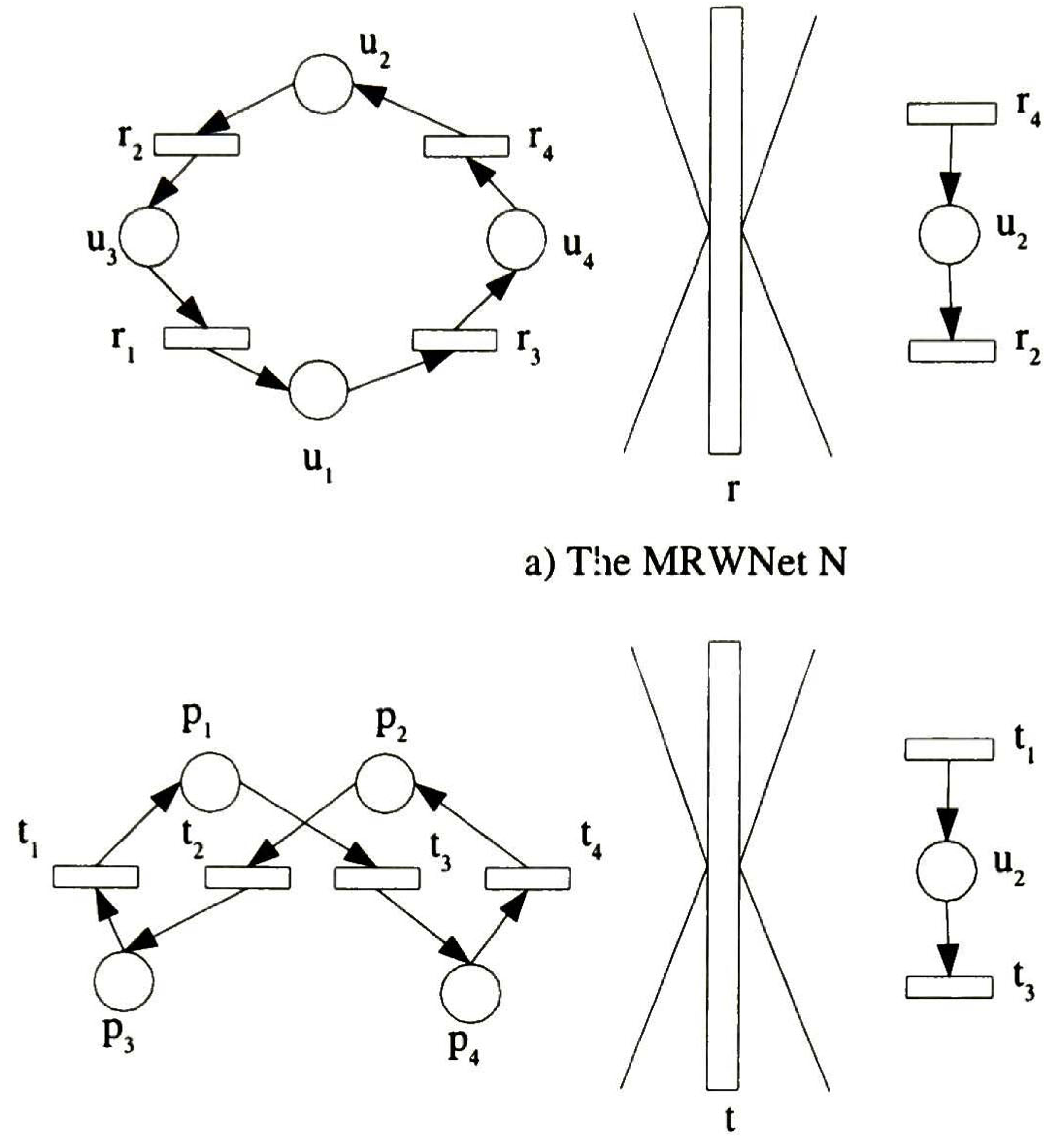
In *RWNets* there exist transitions that can rewrite part of the current net structure. In order to a transition be enabled in RWNets, it is necessary that there exist a matching for its pre set on the current net. The matching between the pre set of a transition and a given subnet requires an isomorphism test as performed in many cases in graph rewriting [Blostein, 1996]. A definition of isomorphism for *RWNets* is given below.

Definition 4.20 (Isomorphism por MRWNets) *Let* $(N, \mu), (N', \mu') \in MRWNets$, *where* $N = (T, \partial_-, \partial_+)$ *and* $N' = (T', \partial'_-, \partial'_+)$; *then they are isomorphic, denoted* $(N, \mu) \cong (N', \mu')$ *iff:*

There exist a pair $\langle f,: T \rightarrow T', g : MRWNets \rightarrow MRWNets \rangle$, *such that,* $\partial'_i \circ f = g \circ \partial_i, i = -.+$, *and recursively* $\mu \cong \mu'$, *and* $\partial'_i \circ f(t) \cong \partial_i(t), i = -, +$ *for all* $t \in T$.

Note that this definition of isomorphism defines a "family" of isomorphisms rather than a single isomorphism.

Figure 4.9 shows two isomorphic *RWNets*. Remember that, it is known that the two nets in the pre set of the two transitions t and r in Figure 4.9 are isomorphic (see Figure 3.2). So this



a) The MRWNet N

a) The MRWNet N'

Figure 4.9: Two Isomorphic RWNets

isomorphism in $RWNets$ is quite natural.

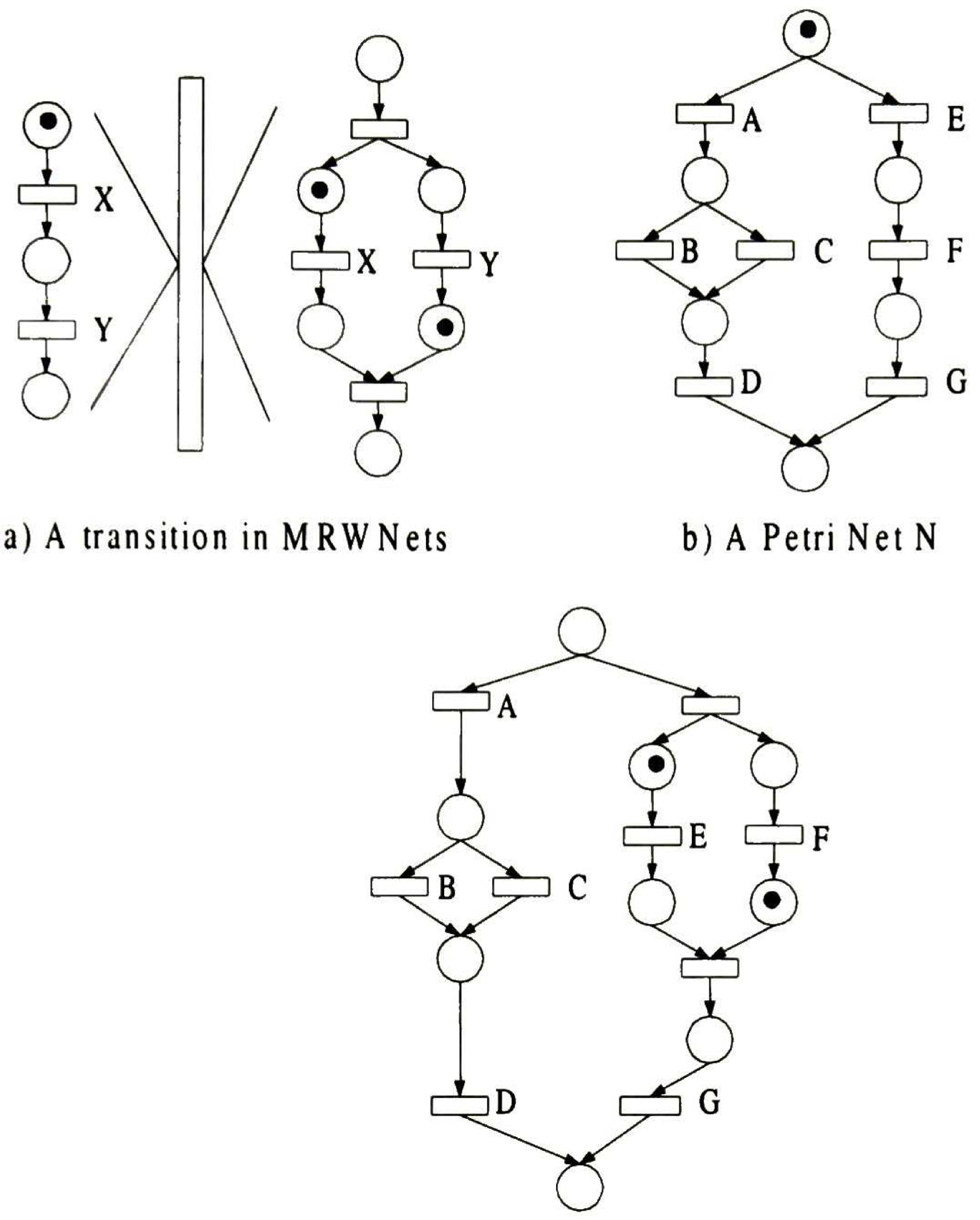
Definition 4.21 (Bounded subnet, binding) *Let $(N,\mu) \in MRWNet$, where $N = (T, \partial_-, \partial_+)$. Let $t \in T$, a* ***matching*** *(or* ***binding****) for $\partial_-(t) \in MRWNets$ is a function $b : MRWNet \rightarrow MRWNet$ such that $\partial_-(t) \cong b(\partial_-(t))$. The net $\partial_-(t)$ is called a* ***bounded net****.*

Once given a matching for the pre set of a transition, the next step is to apply the "family" of isomorphisms defined by such matching to the post set of such transition, in order to get the parametric subnet that must be added to the current net. The next definition is related to the application of this "family" of isomorphisms to a given $RWNet$.

Definition 4.22 (Substitution for MRWNets) *Let b be a binding in $MRWNets$. Let $\langle\langle\langle f,g\rangle\rangle\rangle$ be the family of isomorphisms defined by b. For a $(X,\mu) \in MRWNets$, $(X,\mu)\,\langle\langle\langle\langle b\rangle\rangle\rangle\rangle$ denotes $(X,\mu)\,\langle\langle\langle f,g\rangle\rangle\rangle$, i.e., the application of the family of isomorphisms $\langle\langle\langle f,g\rangle\rangle\rangle$ to (X,μ).*

Now, it is defined the concept of transition enabling in Rewriting Nets. As an intuitive example, the nets depicted in Figure 4.10 show the firing of a transition in $MRWNet$. It works as follows:

Figure 4.10a shows a transition in $MRWNets$. Figure 4.10b shows an ordinary Petri Net, and Figure 4.10c shows the Petri Net resulting of the firing of the "Rewriting Transition" over the ordinary Petri Net.



a) A transition in MRWNets

b) A Petri Net N

c) The Petri Net N after the application of the transition in a)

Figure 4.10: The Firing in RWNets

Intuitively, a transition in Rewriting Nets is enabled if there exist a subnet on the current net that matches its pre set in an isomorphic way. Then, all elements -places, edges, and transitions- that are present in the pre set but are not in the post set, must be removed from the current net; also, all elements -places, edges, and transitions- that are present in the post set but are not in the pre set must be added to the current net. It is easy to see that all these elements that are present in both, the pre set and the post set, must be preserved in the current net. The firing rule, defined later, states how the firing of a transition changes the net structure.

Other example, is shown in Figure 4.11; it is the same Petri Net of the Figure 4.10b, but now after applying the "Rewriting Transition" on Figure 4.10a, that produces a non-sound workflow net.

Definition 4.23 (Transition enabling) *Let $(N, \mu) \in RWNets$, such that $N = (T, \partial_-, \partial_+)$. Let $t \in T$, such that $\partial_-(t) = (N', \mu')$, then t is enabled at μ, under binding b, denoted $(N, \mu)[t\rangle_b$ iff there exist a binding $b : \partial_-(t) \rightarrow MRWNets$, such that $b(N', \mu') \subseteq \mu$.*

Remark 7 *A Labeled Rewriting Net can be obtained by associating a transition labeling function $l : T \rightarrow L$ for any $(N, \mu) \in MRWNets$, as in $MPTNets$.*

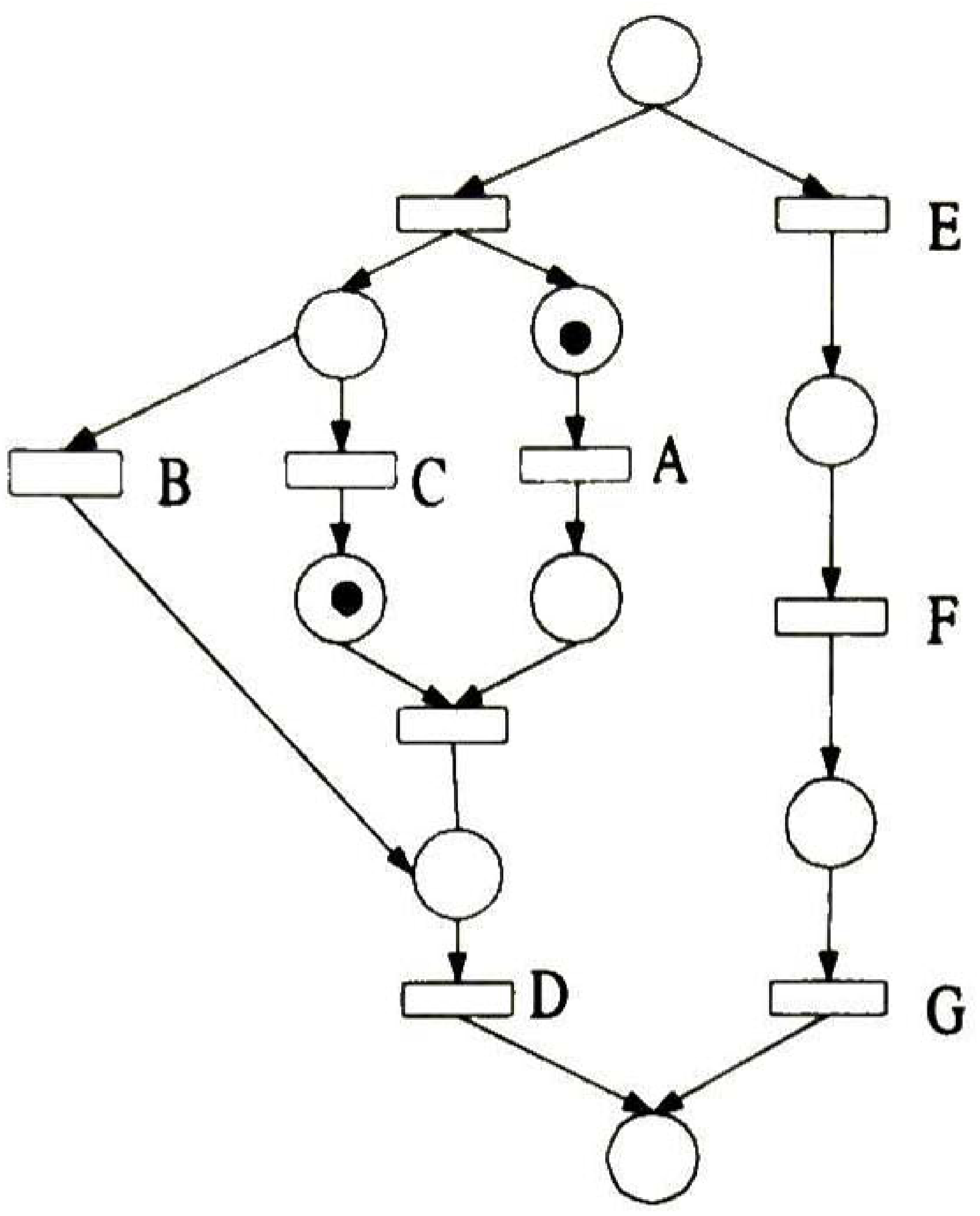


Figure 4.11: A no sound WFNet by the wrong application of a rewriting rule

Definition 4.24 (Firing Rule) *The firing rule for a Labeled* $MRWNet$, $\_[\_\rangle_b\_ \subseteq MRWNet \times L \times MRWNet$*, is the smallest substitutive relation generated by:*

$$(N,\mu)[t\rangle_b \Longrightarrow (N,\mu), [l(t)\rangle_b, (N, \mu\backslash(\partial_-(t)\langle\langle\langle\langle b\rangle\rangle\rangle\rangle \backslash \partial_+(t)\langle\langle\langle\langle b\rangle\rangle\rangle\rangle)) \otimes (\partial_+(t)\langle\langle\langle\langle b\rangle\rangle\rangle\rangle \backslash \partial_-(t)\langle\langle\langle\langle b\rangle\rangle\rangle\rangle)))$$

where b *is a binding for* $\partial_-(t)$. *It is denoted* $(N,\mu)[t\rangle_b(N,\mu')$, *where* $\mu' = \mu\backslash(\partial_-(t)\langle\langle\langle\langle b\rangle\rangle\rangle\rangle \backslash \partial_+(t)\langle\langle\langle\langle b\rangle\rangle\rangle\rangle \otimes (\partial_+(t)\langle\langle\langle\langle b\rangle\rangle\rangle\rangle \backslash \partial_-(t)\langle\langle\langle\langle b\rangle\rangle\rangle\rangle))$

Definition 4.25 (Sound "rewriting" transition) *Let* $(N,\mu) \in MRWNets$*, such that* $N = (T, \partial_-, \partial_+)$. *Let* $t \in T$*, then* t *is called sound iff:* t *produces a sound net* μ' *whenever* μ *is sound.*

Intuitively, each "rewriting" transition t, defines a partial correspondence between elements in its pre set into elements in its post set. Such partial relation determines which elements places, transitions, and arcs- must be deleted, and which ones must be created. For example, the "rewriting" transition on Figure 4.7 determines that the place $s3$ must be deleted, and that the transitions t' and t'' must be created.

Definition 4.26 (Marked Rewriting Workflow Nets) *Let* $MRWflNets \subseteq MRWNets$ *be the set defined as follows:*
a) If $(N,\mu) \in \mathcal{W}$*, then,* $(N,\mu) \in MRWflNets$.
b) The pair (N,μ)*, such that* $N = (T, \partial_-, \partial_+)$*, where* T *is a finite set of transitions,* $\partial_-, \partial_+ : T \rightarrow MRWflNets$ *are pre and post sets, and* $\mu \in MRWflNets$ *is a* $MRWflNets$.

Using Rewriting Workflow Nets, it is possible to support structural changes over a Workflow process definition. Moreover, it is possible to define Rewriting Workflow Nets that preserve soundness through the firing of its transitions. This is presented in the next section.

4.3.5 Using MRWflNets

In this section, it is presented how to use $MRWNets$ in order to support the dynamic changes enunciated in the introduction to this Chapter.

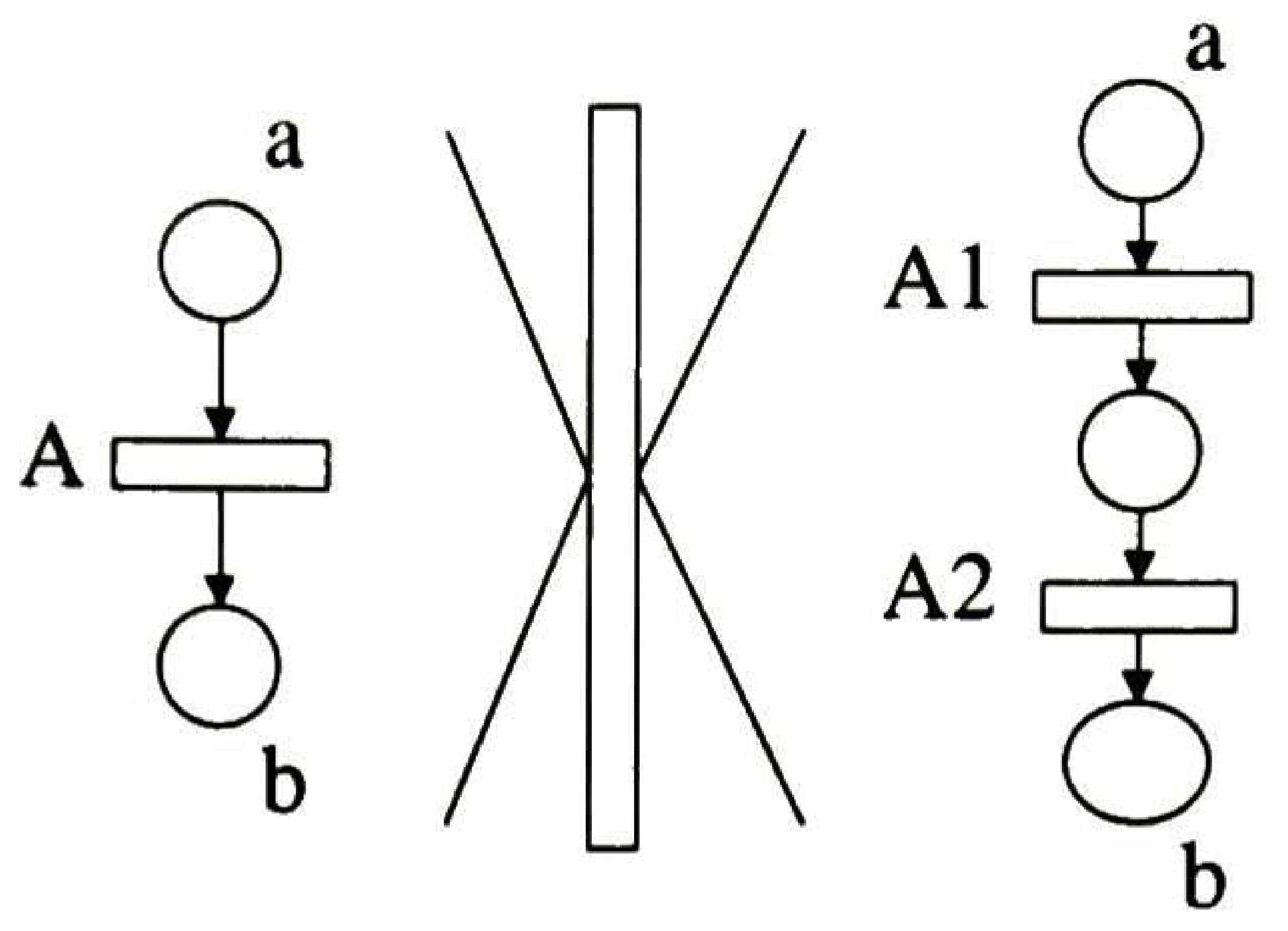


Figure 4.12: The Extend Rewriting Transition

For example, "Extend" change was the first dynamic change presented, Figure 4.2a. Also, "Extend" was defined as: a kind of refinement on the actual set of activities. The next is the case for the "Extend" dynamic change.

Case 12 ("Extend" dynamic change) *Given a current inactive task A in a workflow process, with its causal relationships, pre conditions, and post conditions, it is necessary to refine it into two sub tasks A1 and A2, preserving its causal relationship, pre conditions and post conditions, and adding a causal relationship from A1 to A2.*

Solution 13 ("Extend" rewriting transition) *In agreement with the previous case, it is possible to determine the next topics: the inactive task A, could be represented as a Petri Net transition with no tokens in its input places; each of its pre conditions and post conditions must be represented as a Petri Net place. The preservation of its causal relationship, pre conditions, and post conditions, at the refinement, means that its pre conditions must be the pre conditions of the task A1, and that its post conditions must be the post conditions of the task A2. The addition of a causal relationship from A1 to A2 means that the end of the task A1 activates the task 2.*

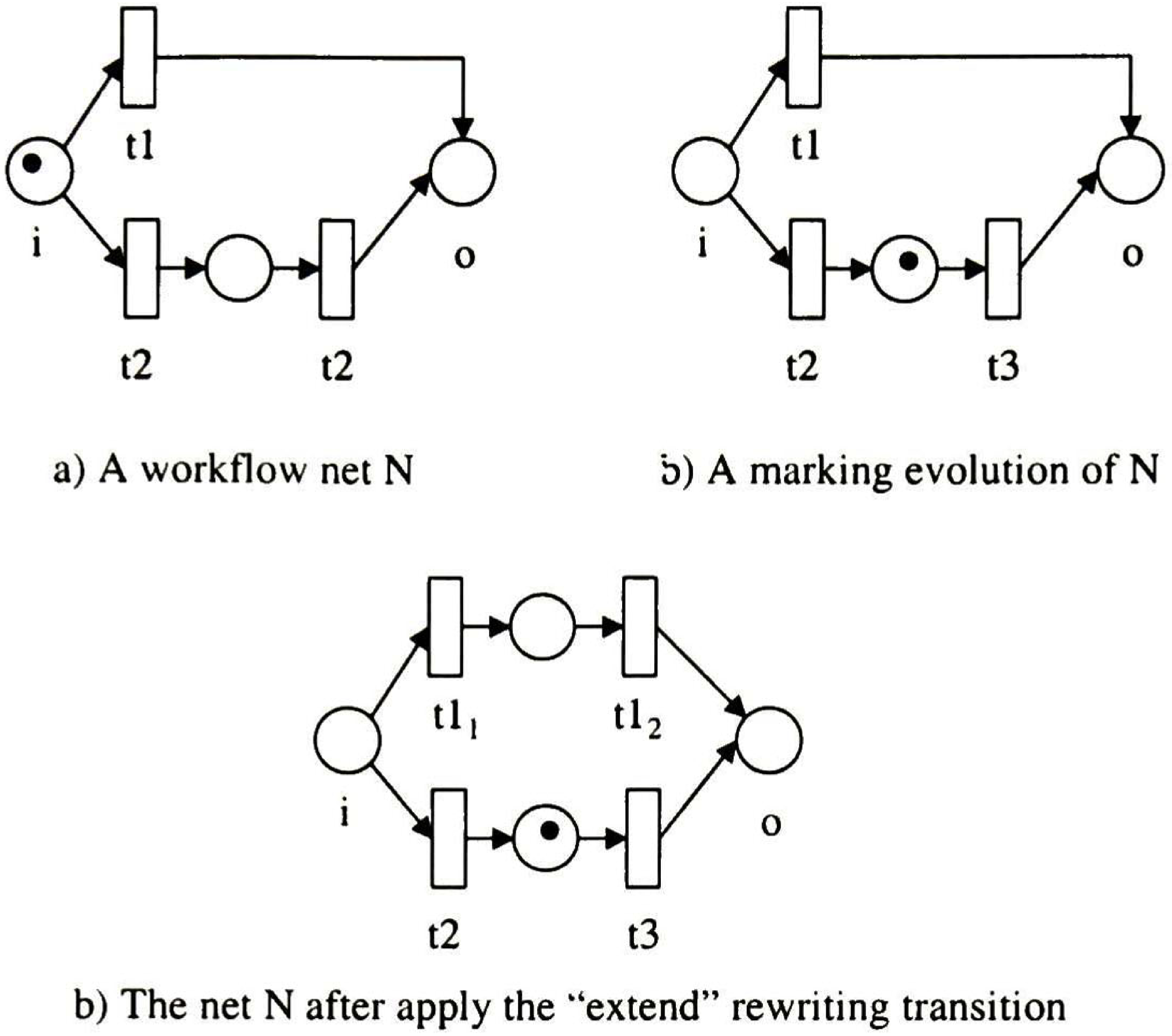
Using this information, the rewriting transition that solves this dynamic change is depicted in Figure 4.12.

Figure 4.13a shows a workflow net. The rewriting transition depicted in Figure 4.12 is not already enabled. However, Figure 4.13b shows a marking evolution of the workflow net. Now, the rewriting transition is already enabled. Finally, Figure 4.13c shows the net after the application of the rewriting transition. Note that all properties required for this dynamic change are satisfied.

The next all, are the remaining changes depicted in Figure 4.2 and their solution as rewriting transitions.

Case 14 ("Replace" dynamic change) *Replace has the meaning of the substitution of a task by another one*

Solution 15 ("Replace" rewriting transition) *Figure 4.14 shows a "generic" rewriting transition that interchanges any workflow pattern by another one. Of course, in any particular situation, this "generic" rewriting transition must be defined in detail.*



a) A workflow net N

b) A marking evolution of N

b) The net N after apply the "extend" rewriting transition

Figure 4.13: The structural change produced by the "extend" rewriting transition

Case 16 ("Re-order" dynamic change) *Re-order has the meaning of a new causal order between the current set of tasks.*

Solution 17 ("Re-order" rewriting transition) *See Figure 4.15.*

Case 18 ("Restart" dynamic changes) *Restart means that the current workflow process must be abruptly restarted.*

Solution 19 ("Restart" rewriting transition) *See Figure 4.16.*

Case 20 ("Version" dynamic change) *Version means that the current workflow process must be readapted into a new business re-engineering rule, such as a refinement of tasks, a reordering of tasks, etc.*

Solution 21 ("Version" rewriting transition) *This dynamic change could be achieved using one of the other rewriting transitions presented. Only, it is necessary to understand the sens of "Version" in each particular scenario; for example, in a particular situation, "Version" could means "congruent to top", i.e., all task in the actual workflow process must be congruent in its new version -that is, if in the current workflow process there exist the task A, then it is non congruent that in the new version of the workflow process exists the undo-task A-. Any way, the "Version" dynamic change depend on the particular situation.*

Case 22 ("Transfer" dynamic change) *Transfer means that all cases present in the current workflow process must be present in the new workflow process version.*

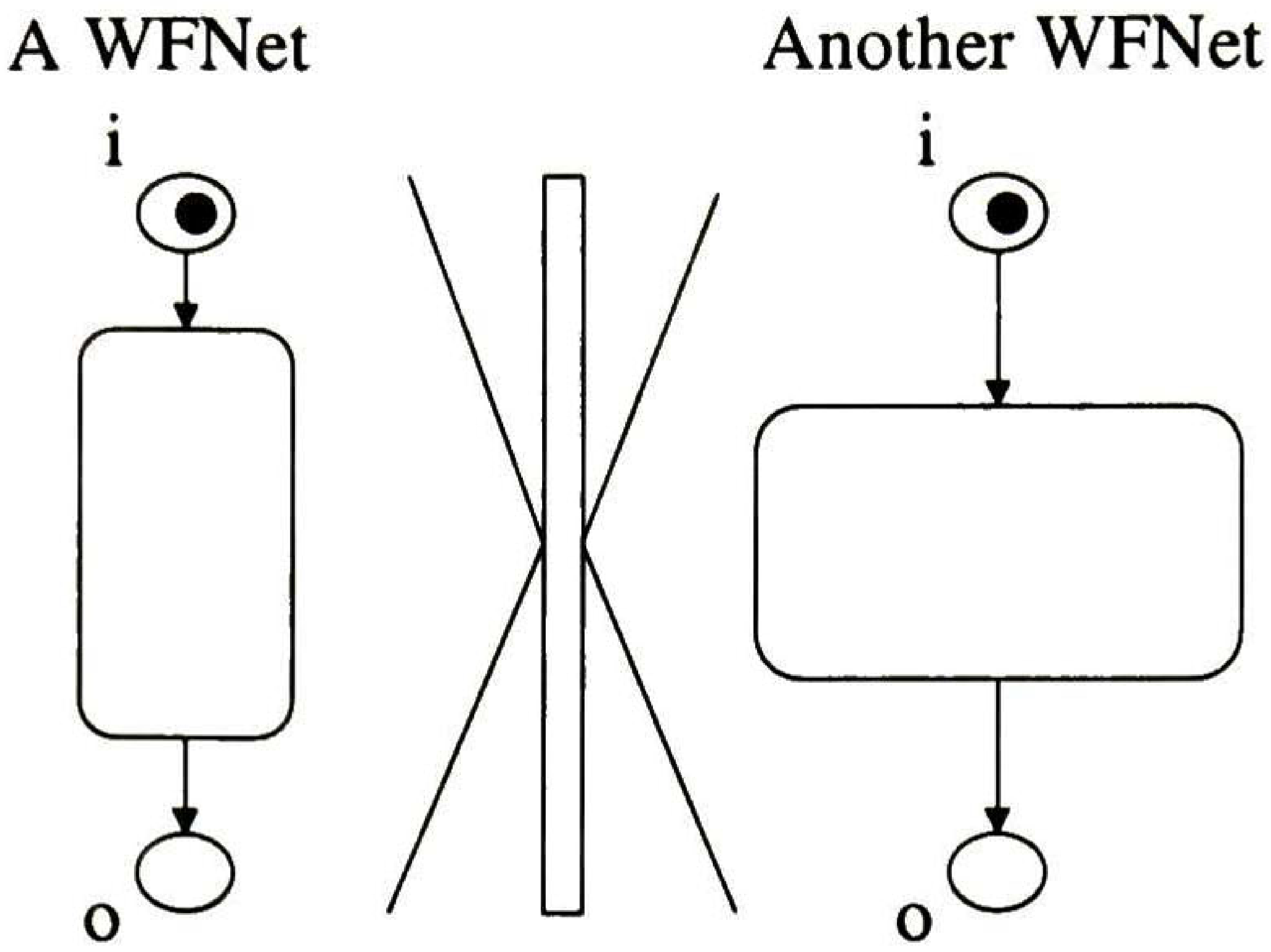


Figure 4.14: The Pattern-by-Pattern Rewriting Net

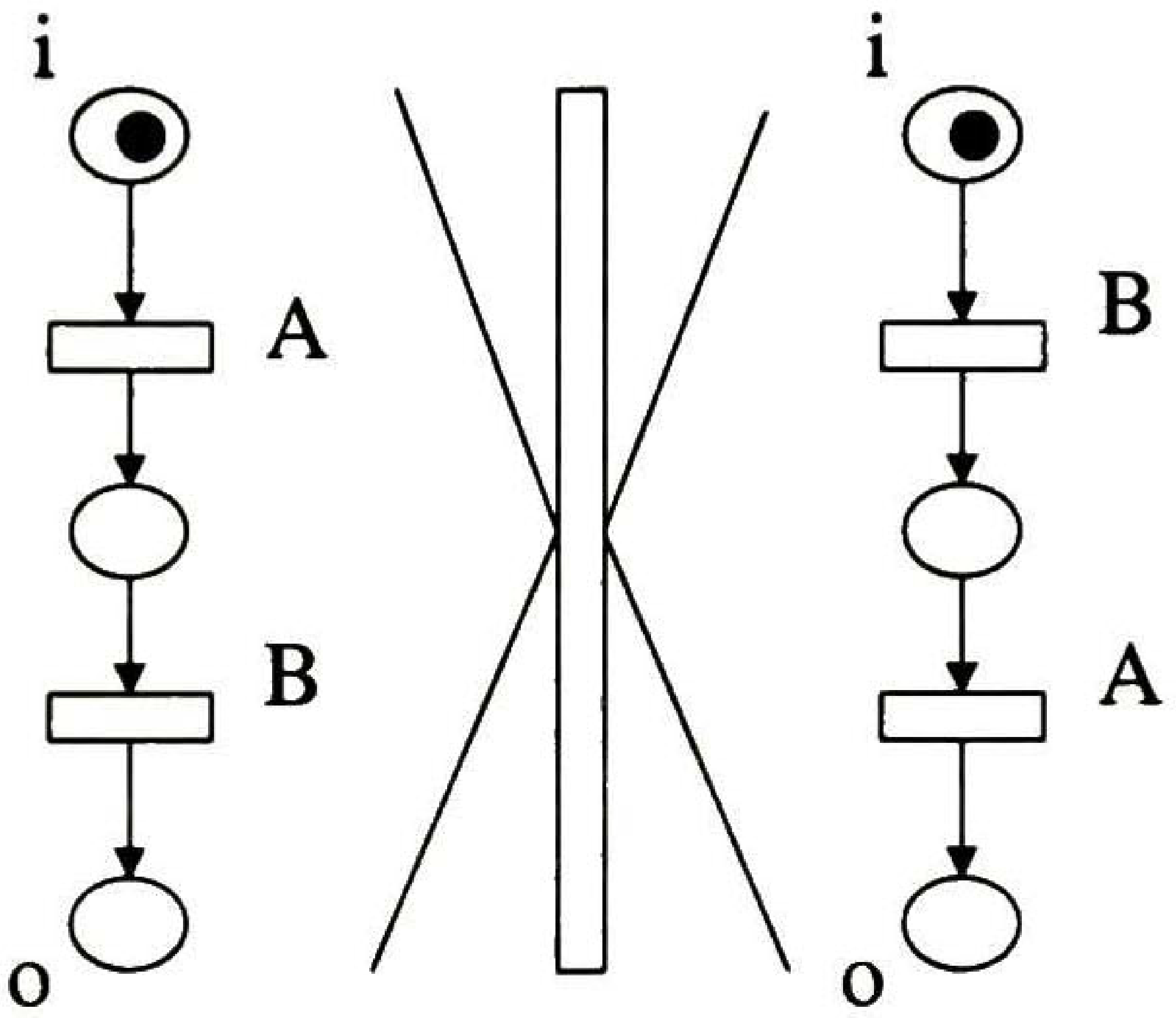


Figure 4.15: The 2-Interchange-Transition Rewriting Net

Solution 23 ("Transfer" rewriting transition) *See Figure 4.17.*

As another example, the next case combines some dynamic changes previously mentioned. Some Rewriting transitions give solution to this particular dynamic workflow change.

Case 24 (Other alternatives and Re-Ordering tasks) *Suppose the next scenario: In a workflow process definition is necessary that there exist the possibility of new flow of activities for some ones already on the system. Also, it is necessary to change the order between existing sequential activities.*

Solution 25 *Suppose the next net:* $N = (A = \{i\} \rhd \{s1\}, B = \{s1\} \rhd \{s2\}, C = \{s2\} \rhd \{s3\}, D = \{s3\} \rhd \{o\}), \mu = \{i\}$. *Now, if it is necessary that this net meets the first condition given above, then it is necessary to add the next rule (or "rewriting" transition) to the net:*

Let $t' = (\{u\}, \{v\}) \rhd N'$ *where* $N' \in \mathcal{W}' \subseteq \mathcal{W}$, *and* $i' = u, o' = v$, *where* i', *and* o' *are the input place and output place of* N' *respectively.*

This transition works in the following way: given a pair of places u, and v on the actual net, it rewrites them into a Workflow process definition that has the place u as its input place and the place v as its output place. Since the places u, and v are present in both the pre set and post set of

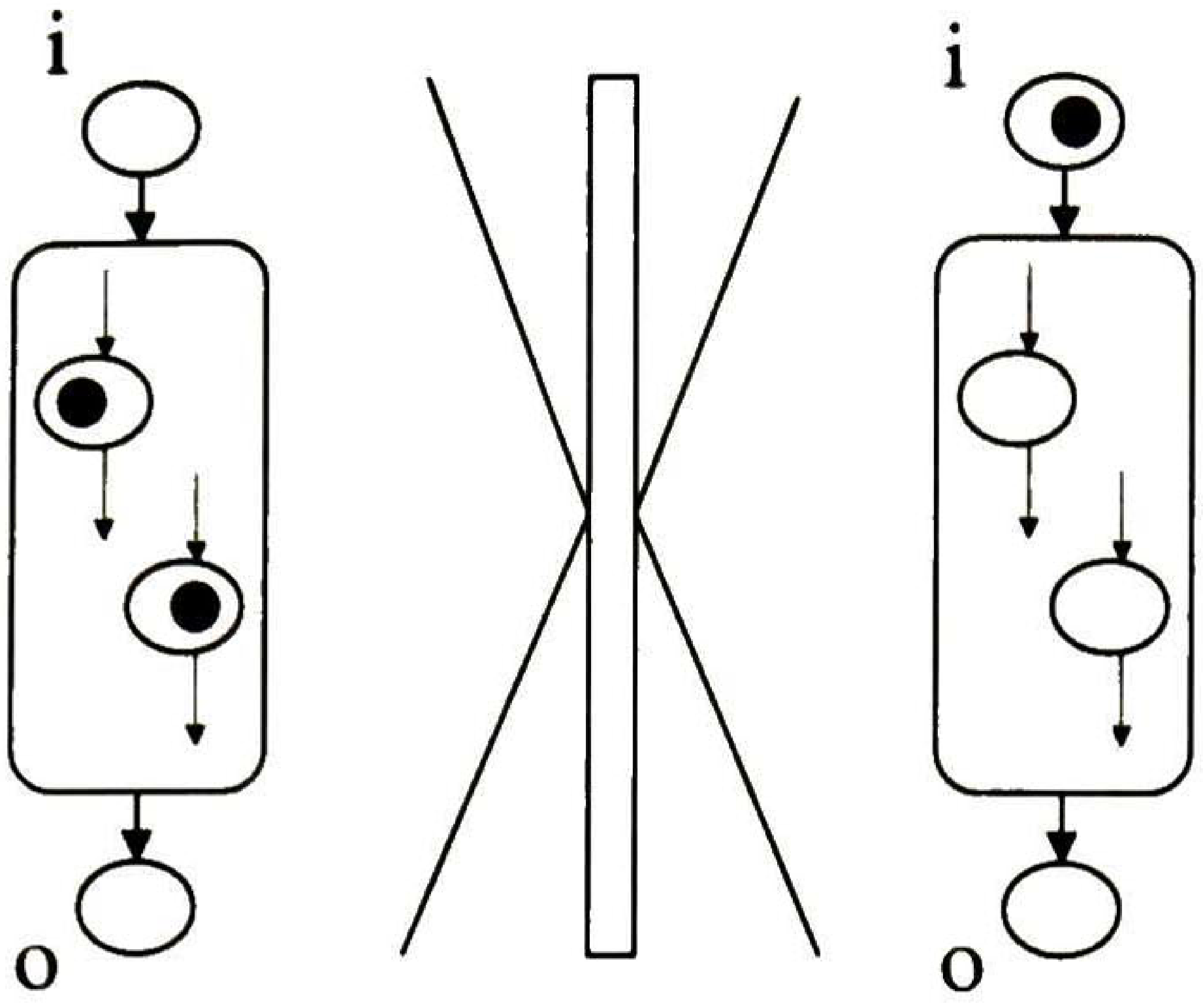


Figure 4.16: The Reset Case Rewriting Net

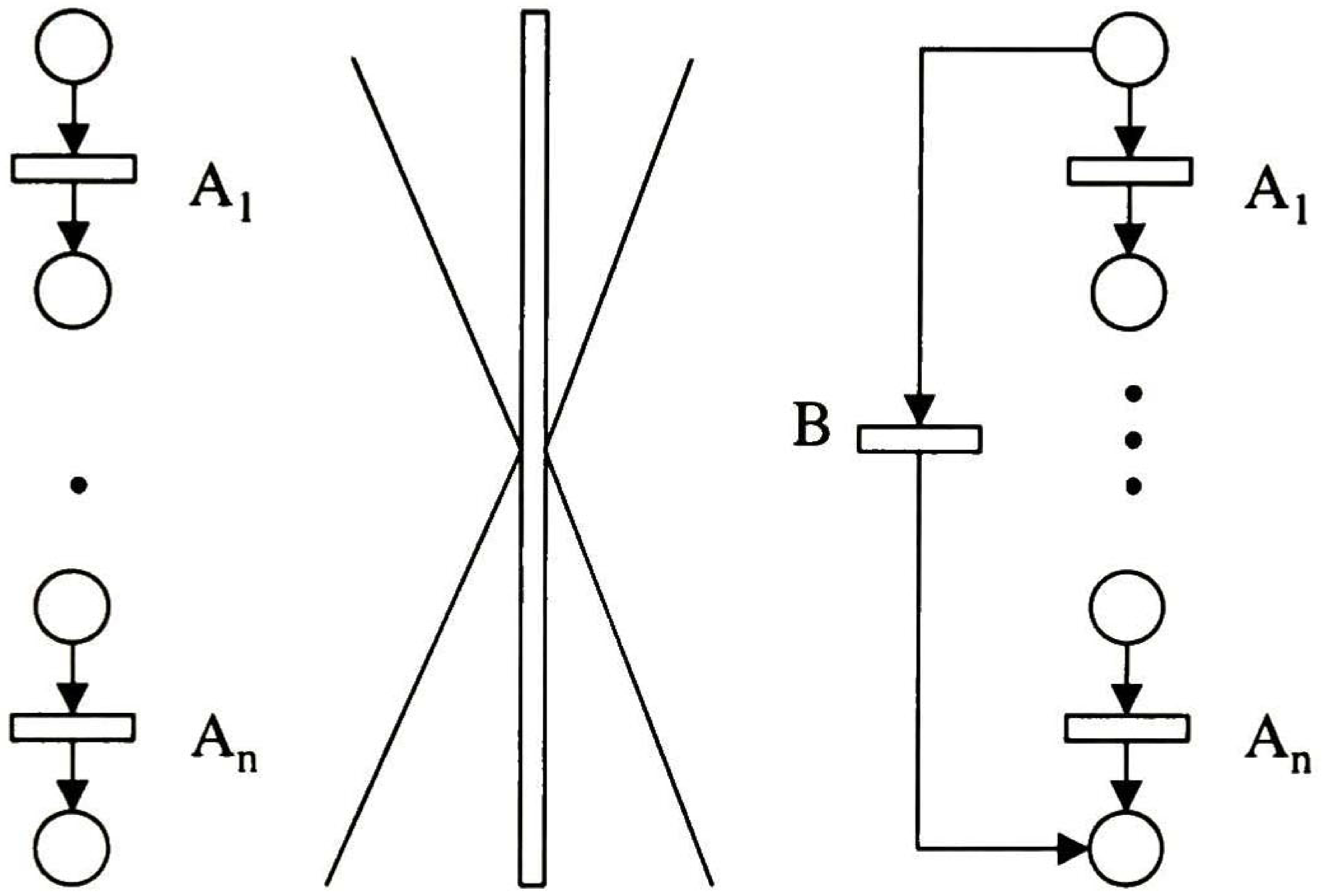


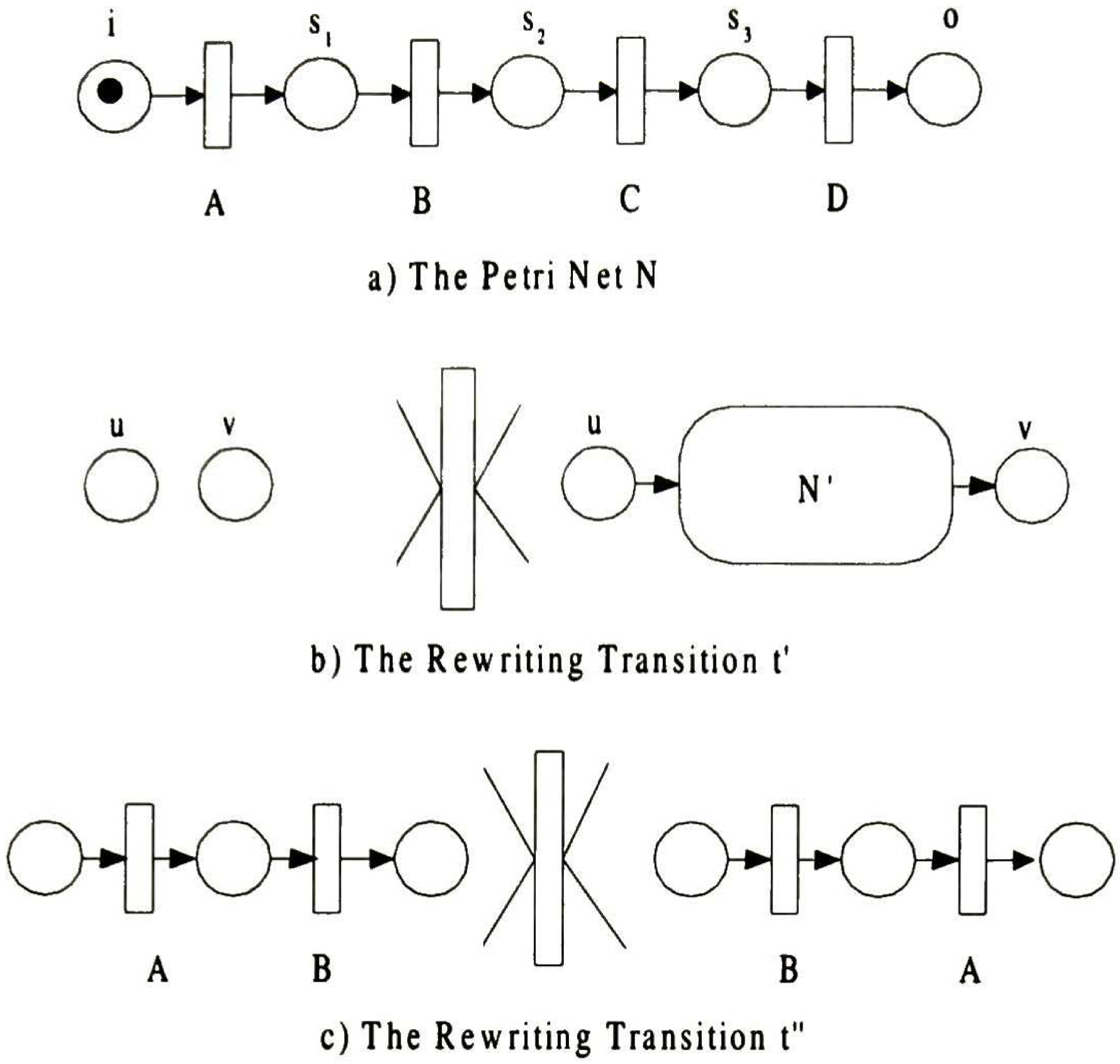
Figure 4.17: A rewriting transition with the transfer condition

the transition, then, this transitions "pastes" a new flow of activities between the places u and v. In this way, the net can perform an alternative "flow-of-work" between any pair of places.

Now, let $t'' = (A = \{u\} \rhd \{v\}, B = \{v\} \rhd \{w\}) \rhd (B = \{u\} \rhd \{v\}, A = \{v\} \rhd \{w\})$*, then it is easy to see that* t'' *performs a reorder between two transitions A and B. So, the net* $(N \cup \{t', t''\}, \mu)$ *meets the problem mentioned above, and it is depicted in Figure 4.18. This net is closed in* $\mathcal{W}$*, i.e., if* $N \in \mathcal{W}$*, then any new net produced by the "rewriting" transitions will be ever in* $\mathcal{W}$*.*

4.4 Graph Grammars and Petri Net Grammars

In this section some concepts on Graph Grammar are presented. An apparent relationship between graph grammar and RWNets is suggested. Moreover, since [Corradini, 1995] suggests graph grammars as a generalization for Petri Nets, it is natural to place some concepts in graph grammars into Rewriting Nets. In short, this section aims to glimpses a suitable relationship between Rewriting Nets and Graph Grammars, and then take advantage of them.



a) The Petri Net N

b) The Rewriting Transition t'

c) The Rewriting Transition t''

Figure 4.18: The RWNet solution for the given planted problem

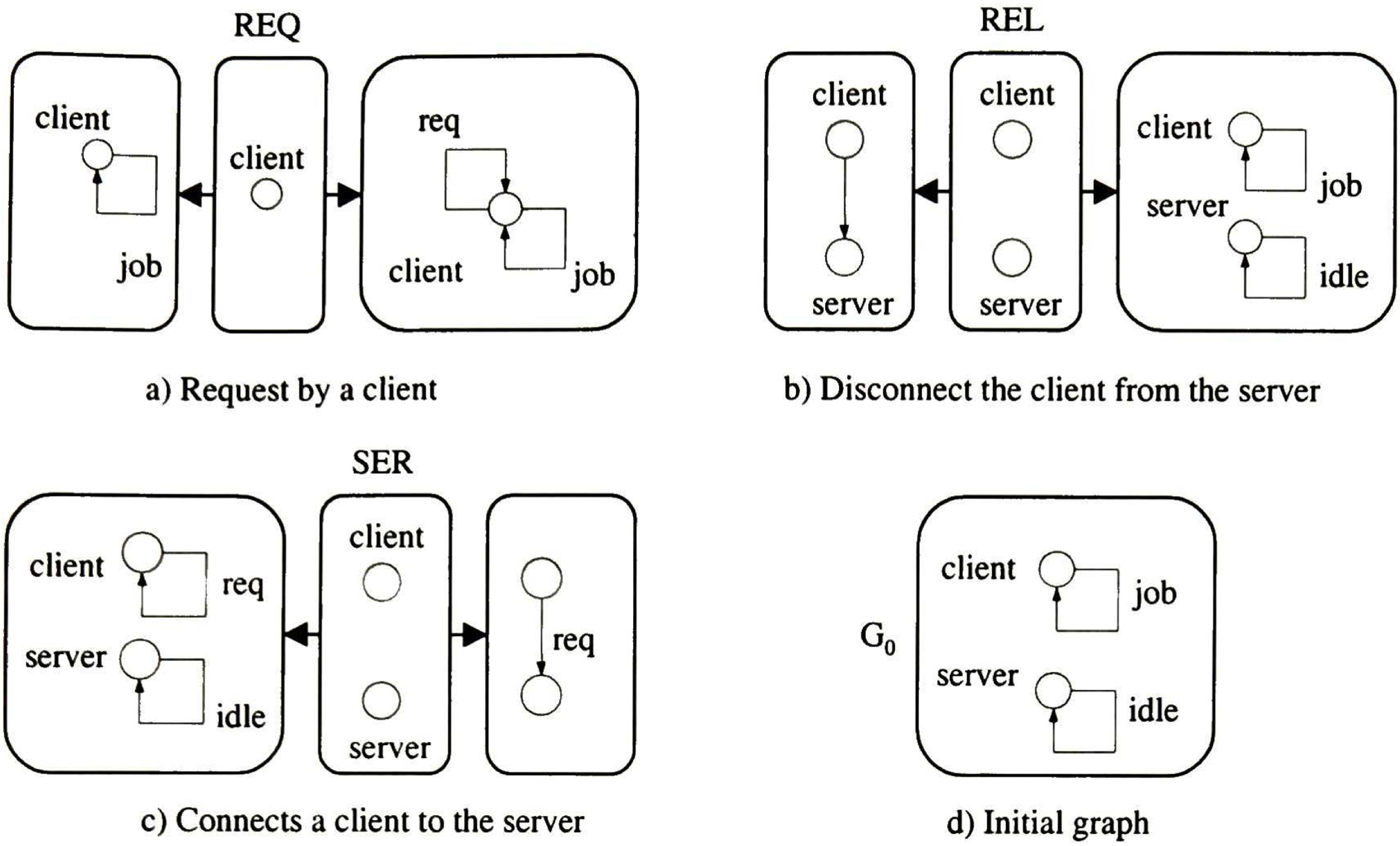
4.4.1 Graph Grammar

It is well know that graphs are an expressive and versatile mathematical tool. In many practical situations, nodes represents static elements, but there are many others practical situations when the vertexes must be dynamic entities. So, it is desirable some ways for the graph manipulation. In many cases this manipulations is implicit, embedded in a software program, for example. However, explicit graph manipulations through well defined graph rewriting rules offers some advantages and allows interesting properties analysis [Blostein, 1996]. The explicit graph manipulation is called graph rewriting and it is useful in many research areas [Blostein, 1996].

In general there exist two approaches for Graph Transformation: the double-pushout (DPO) approach and the single-pushout (SPO) approach [Corradini, 1997].

Definition 4.27 (lebeled graph) *Given two fixed alphabets Ω_V and Ω_E for node and edge labels, respectively, a labeled graph, over Ω_V and Ω_E, is a tuple $G = (G_V, G_E, \partial_-^G, \partial_+^G, lv^G, le^G)$, where G_V is a set of nodes, G_E is a set of edges, $\partial_-^G, \partial_+^G : G_E \longrightarrow G_V$ are the source and target functions, and $lv^G : G_V \longrightarrow \Omega_V$ and $le^G : G_E \longrightarrow \Omega_E$ are the node and edge labeling functions, respectively.*

Definition 4.28 (graph morphism) *Given two graphs G and G' a graph morphism $f : G \longrightarrow G'$ is a pair $f = \langle f_V : G_V \longrightarrow G'_V, f_E : G_E \longrightarrow G'_E \rangle$ of functions which preserve source, target, and labels, i.e., $f_V \circ \partial_+^G = \partial_+^{G'} \circ f_E$, $f_V \circ \partial_-^G = \partial_-^{G'} \circ f_E$, $lv^{G'} \circ f_V = lv^G$, and $le^{G'} \circ f_E = le^G$ If both f_V and f_E are bijections, the f is an isomorphism. If there exist an isomorphism $f : G \longrightarrow H$, we write $G \cong H$; also $[G] = \{H | G \cong H\}$. An automorphism of a graph G is an isomorphism $\phi : G \longrightarrow G$; if $\phi \neq id_G$, we say that ϕ is non-trivial.*



a) Request by a client

b) Disconnect the client from the server

c) Connects a client to the server

d) Initial graph

Figure 4.19: A Cliente/Server System Graph Grammar

Definition 4.29 (Graph productions) *A graph production is an element $p : (L \xleftarrow{l} K \xrightarrow{r} R)$, where L, K, and R are graphs, called left-hand, interface, and right-side of the p production. The elements $l : K \to L$, and $r : K \to R$ are graph morphisms; p is the name of the graph production.*

Suppose, without lose of generality, that in a production $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ morphisms l and r are inclusions. Then, intuitively, the production could be interpreted as follows: all elements in K must be preserved; all elements in L which are not present in K must be deleted, and all elements in R which are not present in K must be created.

Definition 4.30 (span-isomorphic productions) *Given two productions $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ and $p' : (L' \xleftarrow{l'} K' \xrightarrow{r'} R')$ a span-isomorphism $r : p \to p'$ is a tuple of isomorphisms $r = \langle \phi_L : L \to L', \phi_K : K \to K', \phi_R : R \to R' \rangle$ such that $\phi_L \circ l = l' \circ \phi_K$ and $\phi_R \circ r = r' \circ \phi_K$.*

Definition 4.31 (Graph Grammar) *A Graph Grammar ıs a pair $\mathcal{G} = \left\langle (p : L \xleftarrow{l} K \xrightarrow{r} R)_{p \in P}, G_0 \right\rangle$ where the first component is a finite set of production names over P, and G_0 is a graph, called the start graph.*

Definition 4.32 (pushout and pushout complement) *Given a category **C** and two arrows $b : A \to B$, and $c : A \to C$ in **C**, a triple $\langle D, g : B \to D, f : C \to D \rangle$ is called a pushout of $\langle b, c \rangle$ if*

- *$g \circ b = f \circ c$, and*
- *for all objects D' and arrows $g' : B \to D'$ and $f' : C \to D'$ with $g' \circ b = f' \circ c$, there exist a unique arrow $h : D \to D'$ such that $h \circ g = g'$ and $h \circ f = f'$*

$$\begin{tikzcd}
(L \arrow[d, "m"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] & R) \arrow[d, "m*"] \\
(G & D \arrow[l, "l*"] \arrow[r, "r*"'] & H)
\end{tikzcd}$$

Figure 4.20: Direct Derivation as double-pushout construction

For more concepts on category theory see [Lane, 1971].

Definition 4.33 (direct derivation) *Given a graph G, a graph production $p : (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$, and a match $m : L \rightarrow G$, a direct derivation from G to H using p, based on m, exist iff the diagram in Figure 4.20 can be constructed, requiring that both squares be pushouts in Graph. In this case, D is called the* ***context graph****, and we write $G \overset{p,m}{\Longrightarrow} H$, in short $G \overset{p}{\Longrightarrow} H$.*

Given a production $p : (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ and a graph G, it is possible to apply p over G, if there exist an occurrence of L in G, i.e., a morphism $m : L \rightarrow G$, called a match. If p is applicable to G through m, then it is produced the graph H, as shown Figure 4.20, it is denoted $G \overset{p.m}{\Longrightarrow} H$.

Example 4.2 (A Graph Grammar) *As an example of a graph grammar, let $C/S = (\{REQ, SER, REL\}, G($ be the graph grammar on Figure 4.19, then C/S is the graph grammar for a Client/Server system. This graph grammar works in the following way:*

In this case, a client could perform an internal activity when it decides to make an asynchronous request to the server. This is represented by the production REQ depicted in Figure 4.19a.

Now, if the client make a request to the server and the server is idle, then the connection between them take place. This is represented by the production SER depicted in Figure 4.19c.

If it is the case when the communication between the client and the server must be done, then connection is released. This is represented by the production REL depicted in Figure 4.19b.

Finally the Figure 4.19d shows the start graph: a client performing an internal activity and the server in the idle state.

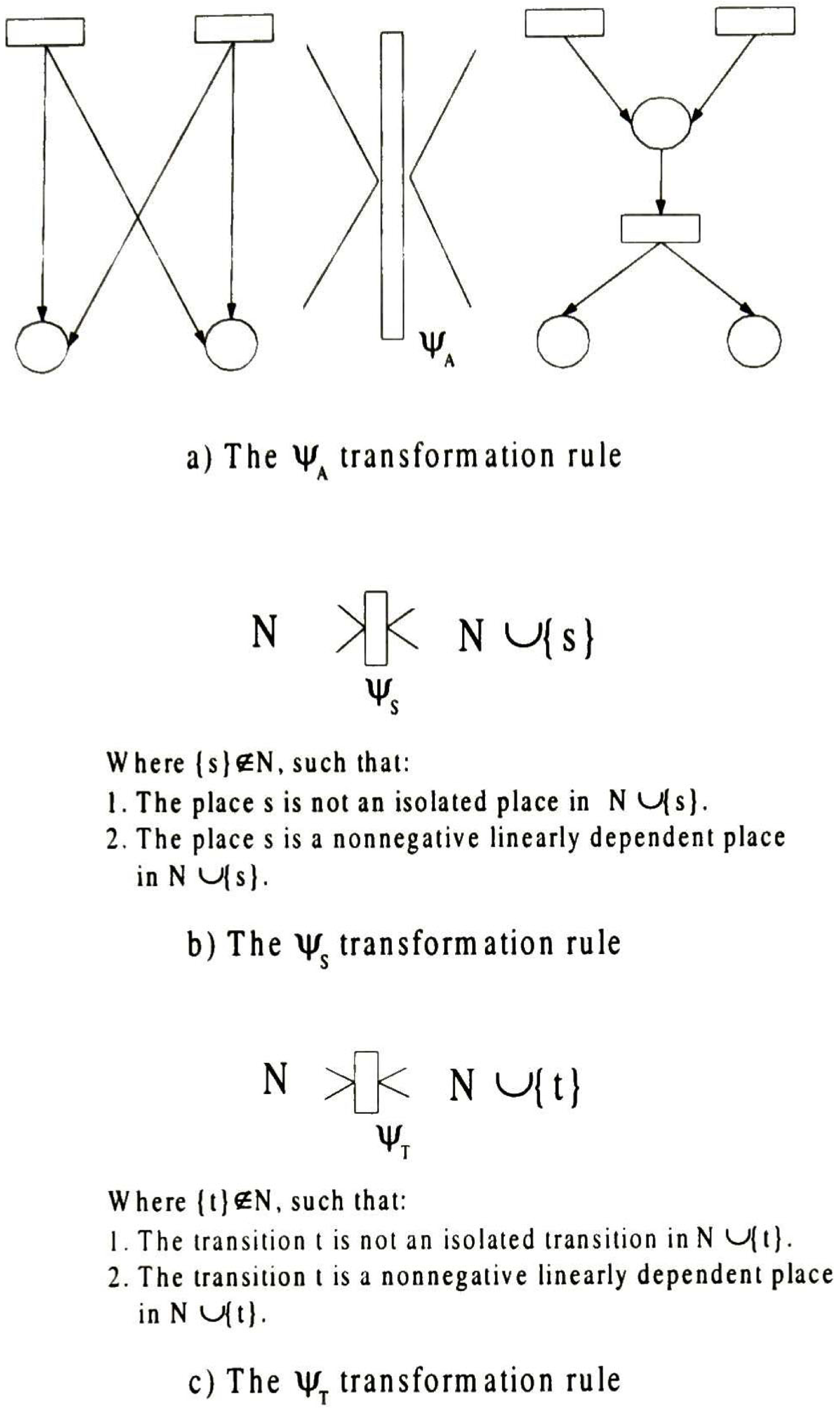
Finally, there exist many interesting results of the algebraic approach of the graph transformation, such as: independence and parallelism of productions, amalgamation and distributions of direct derivations, etc. For more see [Corradini, 1997]

4.4.2 Petri Net Grammar

Now, in this section it is defined Petri Net Grammar, a special case in Rewriting Nets. Basically, a Petri Net Grammar is a simple restriction in Rewriting Nets that allow to transitions rewrites at most ordinary Petri Nets. It is very similar to the case of Graph Rewriting, with some finesses.

In general, a graph rewriting rule r is applicable over a graph G iff there exist a match for its lest-side graph; then the graph transformation could take place.

On the other hand, a "rewriting" transition t in RWNets is enabled iff there exist a match for its pre set, which may includes a marking -i.e., a certain state on the system-; then the Petri Net transformation could take place. The formal definition for a Petri Net grammar is given below.



a) The Ψ_A transformation rule

Where $\{s\} \notin N$, such that:
1. The place s is not an isolated place in N $\cup\{s\}$.
2. The place s is a nonnegative linearly dependent place in N $\cup\{s\}$.

b) The Ψ_S transformation rule

Where $\{t\} \notin N$, such that:
1. The transition t is not an isolated transition in N $\cup\{t\}$.
2. The transition t is a nonnegative linearly dependent place in N $\cup\{t\}$.

c) The Ψ_T transformation rule

Figure 4.21: A Petri Net Grammar

Definition 4.34 (Petri Net Grammar) *Let (N, μ) be a $MRWNets$, where $N = (T, \partial_-, \partial_+)$. Then (N, μ) is a Petri Net grammar (PTGrammar) iff for all $t \in T$, $\partial_i(t)$ is a $MPTNet$, $i = -, +$, and μ is a $MPTNets$.*

Definition 4.35 (Isomorphic Rewriting Transition) *Let $(N, \mu) \in MRWNets$, where $N = (T, \partial_-, \partial_+)$. Let $t, t' \in T$ Then, t and t' are isomorphic, denoted $t \cong t'$, iff there exist a pari of isomorphisms $f : \partial_-(t) \longrightarrow \partial_-(t')$, and $g : \partial_+(t) \longrightarrow \partial_+(t')$.*

As in graph grammars, where a pair of span-isomorphic production determines that they produce a similar transformation over a graph model, in RWNets, a pari of isomorphic rewriting transitions determines that it rewrites the Petri net model in a similar way.

Intuitively, in both cases, graph grammars and RWNets, it is possible to remove one of these isomorphic rewriting elements, giving a more light model of the system with a similar behavior.

Figure 4.21 shows three rewriting transitions. Such rules are called: ψ_A, ψ_S, ψ_T, respectively. Given a MPTNet (N, μ), such that $N = (T, \partial_-, \partial_+)$, then they works as follows:

- The ψ_A rule transforms a set of transitions, places, and arcs -as it is shown in its pre set-, into the net of its post set. It is easy to see that no places and no transitions are missed after to apply this rewriting transition. Moreover, a new place and a new transition are added. It is depicted in Figure 4.21a.

- The ψ_S rule adds a new place s into a given net and then connects it in such way that this place becomes a nonnegative linearly dependent place. Moreover, the new place s is not an isolated place in the compounded net. This rule is depicted in Figure 4.21b.

- The ψ_T rule is quite similar to the ψ_S one, in the way that the ψ_T rule add a new transition t to a given net and then connect it to the net such way that t becomes a nonnegative linearly dependent transition. Also, the new transition t is not an isolated transition. This rule is depicted in Figure 4.21c.

The last three rewriting transitions are so important since it is proved that this rules, using the *loop net* -Petri Net with one place and one transition and a loop between them- as a start net, produce the set of all well-formed Free-Choice Petri Nets! [Desel, 1995].

Since the Petri Net grammar $(\{\psi_A, \psi_S, \psi_T\}, \mu_0)$, where μ_0 is the *loop net*, produce all well-formed Free-Choice nets, it is easy to see that these three rewriting transitions are sound.

As an application example for Petri Net grammar, look the next grammar:

$$(\{P - by - P, Restart, Cancel, \psi_A, \psi_S, \psi_T\}, \mu_0)$$

If it is constrained the $P - by - P$ to rewrite only workflow nets that are Free Choice Nets and μ_0 be the loop net, then we have a Petri Net grammar that produce all well-formed workflow nets that are FC Nets. This is easy to see since the ψ_A, ψ_S, and ψ_T are yet mentioned, they are closed under the set of all FC Nets, and the $P - by - P$ rewriting rule was constrained to FC Nets. The *Restart* and *Cancel* rewriting rules does not perform any change in the net structure, but they perform a change on the net markings. Then, we starting with the loop net μ_0 we could construct any well-formed free choice workflow net that could change any workflow subnet by other workflow net at any time. Also, this net could transfer any reachable marking into both the START state or the END state.

For the Dynamic Change in Workflow Management, it is possible to construct a Petri Net grammar that meets the specific class of dynamic change.

A similar approach of Petri Net Grammars was studied in [Padberg, 1998] and [Padberg, 1998 LNCS].

4.4.3 Conclusions

Dynamic Nets proposed by [Buscemi, 2001] and [Asperti, 1996] allow a degree of flexibility. Despite this flexibility, Dynamic Nets have some restrictions: for example, given two sequential transitions, it is no possible to perform its parallel representation. Rewriting Nets represent an extension for

Dynamic Nets. Rewriting Workflow Nets describe a framework for the dynamic change within Workflow Management, using Petri Nets.

Chapter 5

Implementation of RW-Nets

This chapter introduces the implementation of some rewriting transitions applicable over all Petri Nets, whenever there exist a match for its pre set; a module integrating a set these rewriting transitions is built into Petri Net Kernel (PNK) -a software tool that allows to construct Petri net software tools-. First, an overview of PNK is presented, then the main features of the set of rewriting transitions is described, through their application over a workflow net.

5.1 Implementing dynamic changes

A workflow model expressed as an ordinary PN is the core of a software implementing workflow management system (WFMS). The aim of this chapter is to demonstrate that the use of RWNets provide dynamic change capabilities to a PN based WFMS.

For this propose, an software implementation over Petri Net Kernel (PNK) was developed in order to perform structural changes over Petri Nets, using RWNets concepts.

Several rewriting transitions were implemented as Java classes:

- cancel-case rewriting transition: this transition transform any reachable marking into the END state by a given workflow net.
- reset-case rewriting transition: this transition transform any reachable marking into the START state by a given workflow net.
- pattern by pattern rewriting transition: this transition replace a workflow pattern by another one.
- interchange two transitions: this transitions interchange two sequential transitions.
- sequence to parallel rewriting transition: this transition transform two sequential transitions into a pair of parallel ones.
- parallel to sequence rewriting transition: this transition transform two parallel transitions into a pair of sequential ones.

Before to describe the module for these rewriting transitions implementation, an overview of the PNK is presented.

5.2 The Petri Net Kernel

The PNK provides an framework for the construction of Petri Net tools [Kindler, 2001], and is divided into the next main modules: the PNK Kernel, the PNK Application Control, the PNK net element extensions, and the PNK exceptions, Figure 5.1. The next section explores some of these modules.

The PNK uses the PNML language -a language for the description of petri nets based on XML- [Weber, 2002], that provides a generic interchange mechanism between Petri Net software applications.

The PNK provides a graphical net editor, that allows to construct a net, dragging and dropping places and transitions; also it allows to the user to save, to open and to modify files containing nets. Figure 5.2 shows a window of the PNK editor in which is depicted a workflow net.

The PNK also provides a net simulator, that allows two operation modes: it works with or without the user interaction. Figure 5.3 shows the case where the net simulator interacts with the user. In this case, the PNK simulator requests to the user for the selection of one emphasized transition. Of course, the set of emphasized transitions are the enabled transitions in the current marking.

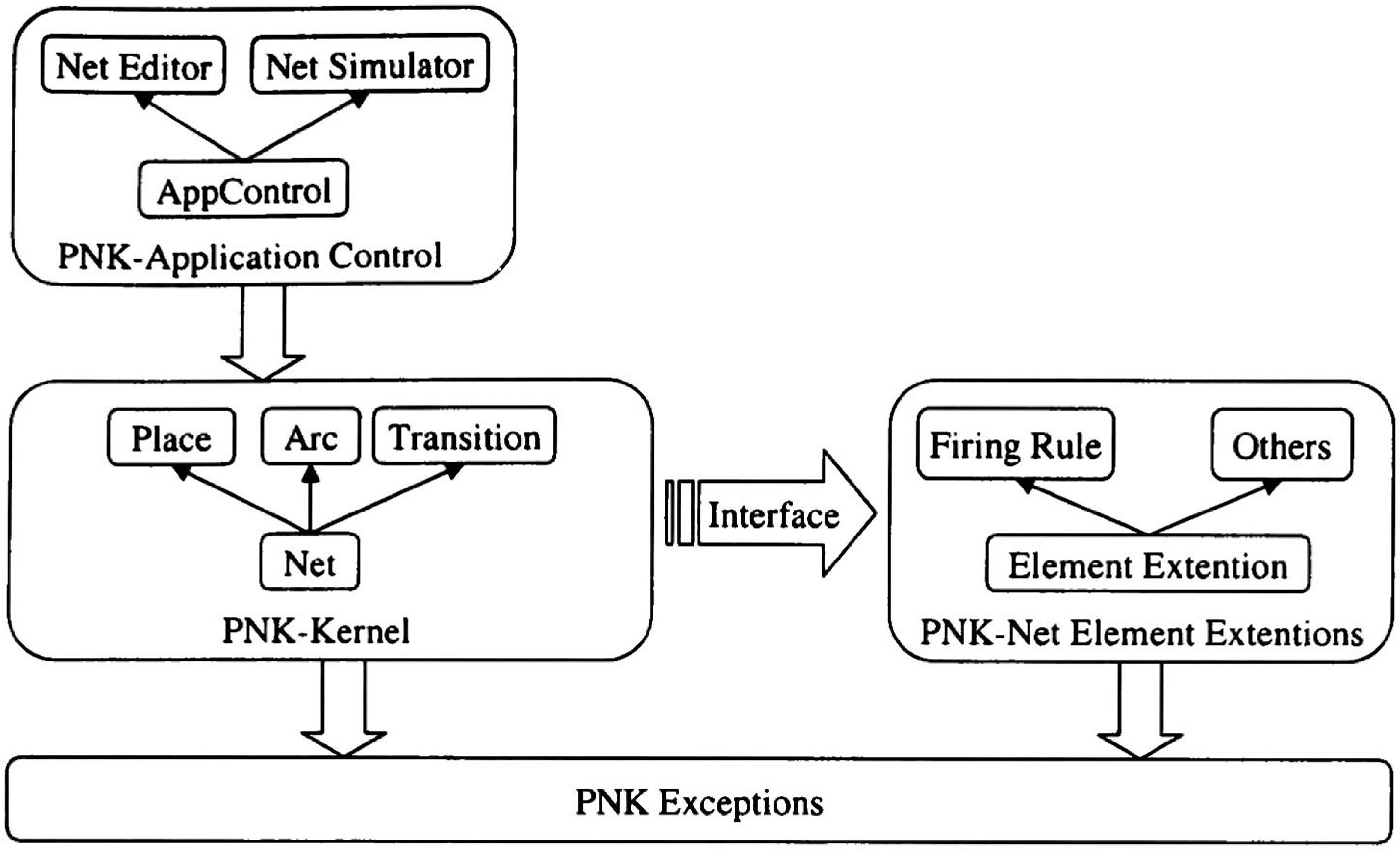


Figure 5.1: The Petri Net Kernel

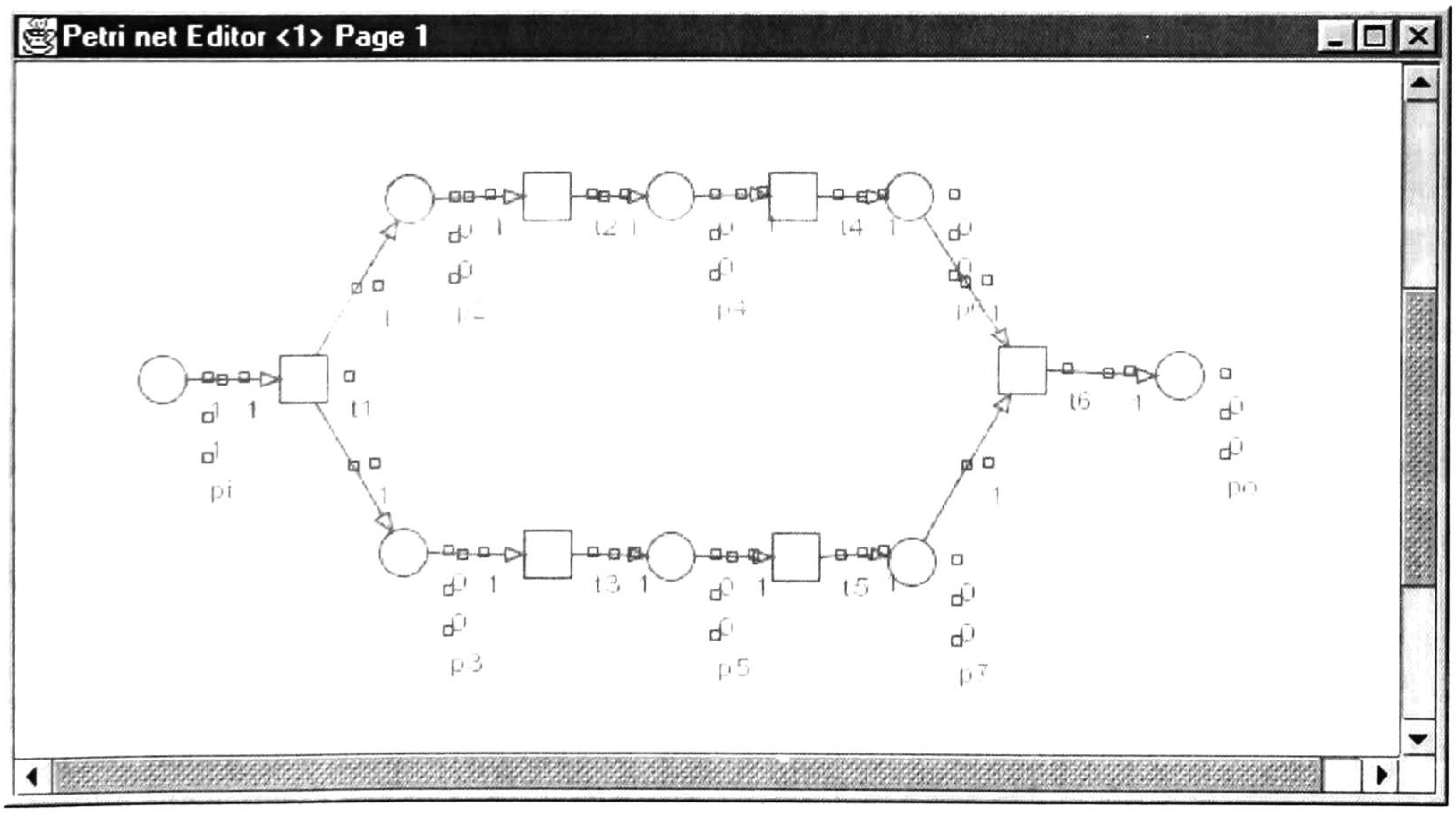
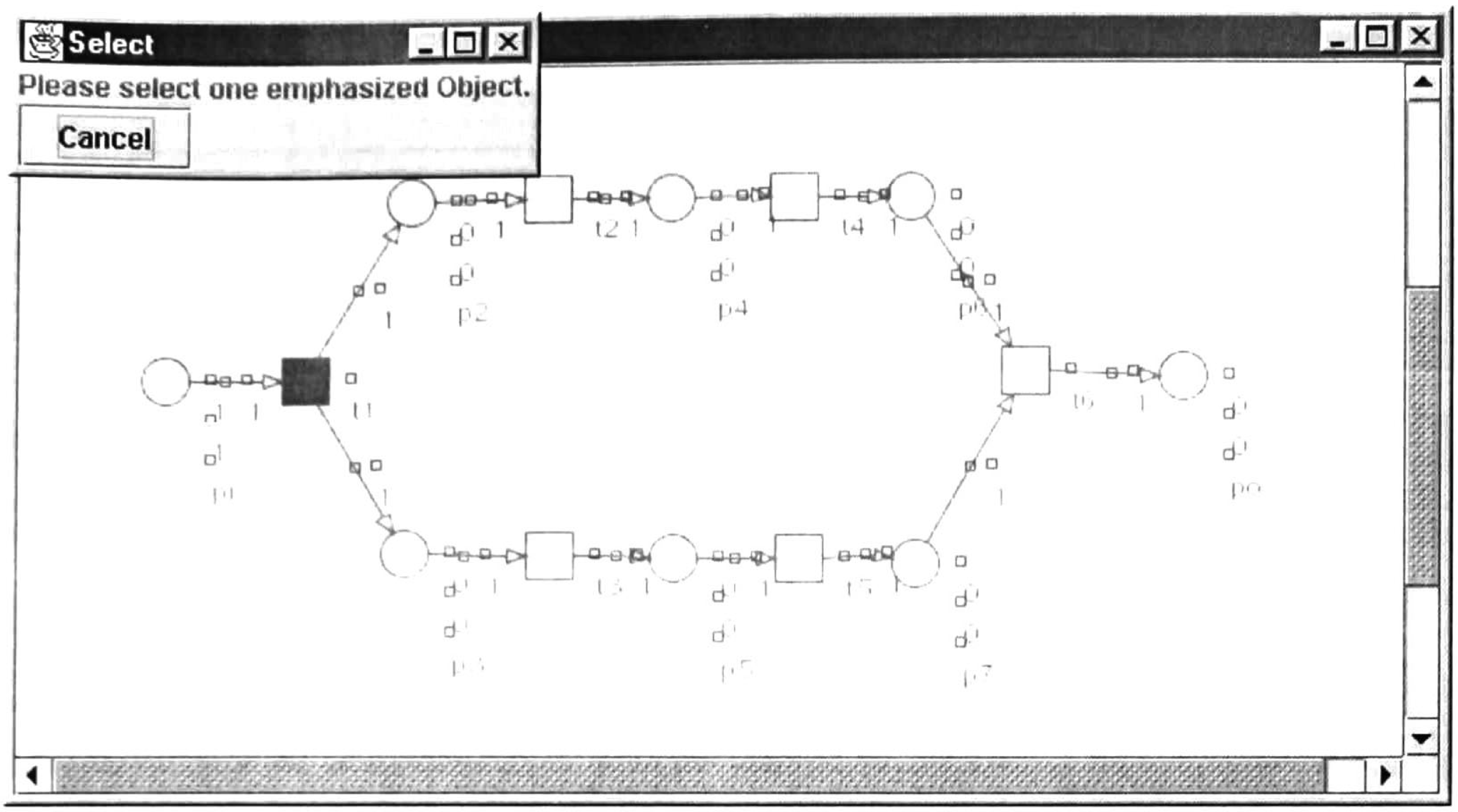


Figure 5.2: The PNK Editor



Select

Please select one emphasized Object.

Figure 5.3: The PNK Simulator

The PNK Kernel is the core of the PNK. It provides a complete set of Java classes that meet the major concepts of Petri Net It has mainly the next Java classes:

- Net : This class provides a complete description for a Petri Net. It offers methods to access their arcs, places and transition.

```
public final class Net extends Graph {
   public Vector getArcs()
   public Vector getPlaces()
   public Vector getTransitions()
   public FiringRule getFiringRule()
} // class Net
```

- Arc: This class describes an arc in Petri Nets. When it is created a new arc it register itself into the net passed as constructor's parameter. It is possible to assign an inscription, or weight, to a certain arc. Also, it is possible to know the source and target nodes for a given arc, etc.

```
public class Arc extends Edge {
   public Arc(Net net, Node source, Node target, Object initiator)
   public Inscription getInscription()
   public Place getPlace()
   public Transition getTransition()
   /// METHODS INHERITED FROM class Edge:
   public Node getSource()
   public Node getTarget()
} // class Arc
```

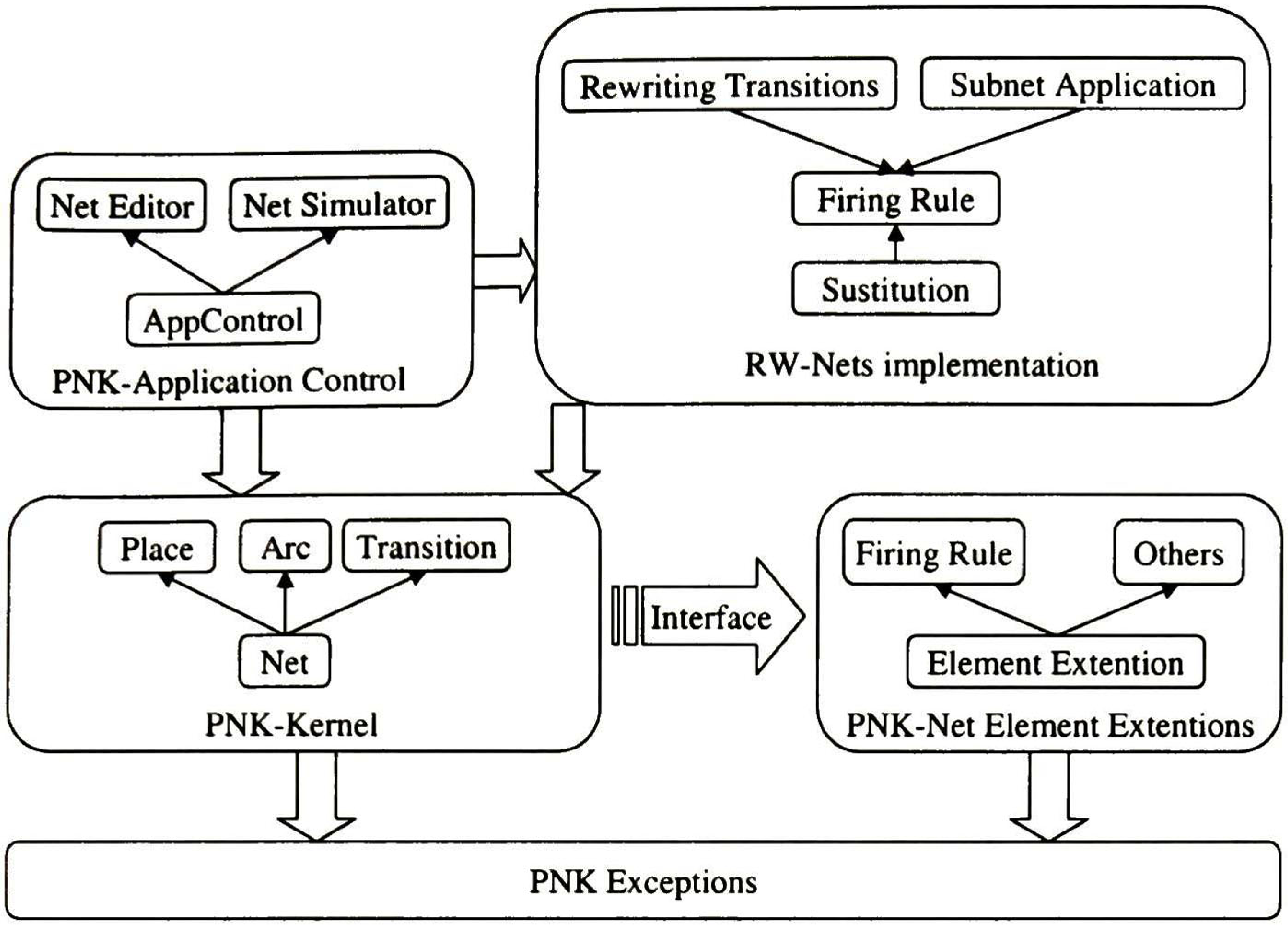


Figure 5.4: The RW-Nets implementation

- Place: This class describes a place in Petri Nets. When it is created a new place, it register itself into the net passed as constructor's parameter. It is possible to know the actual marking for a place, the input and output edges, etc.

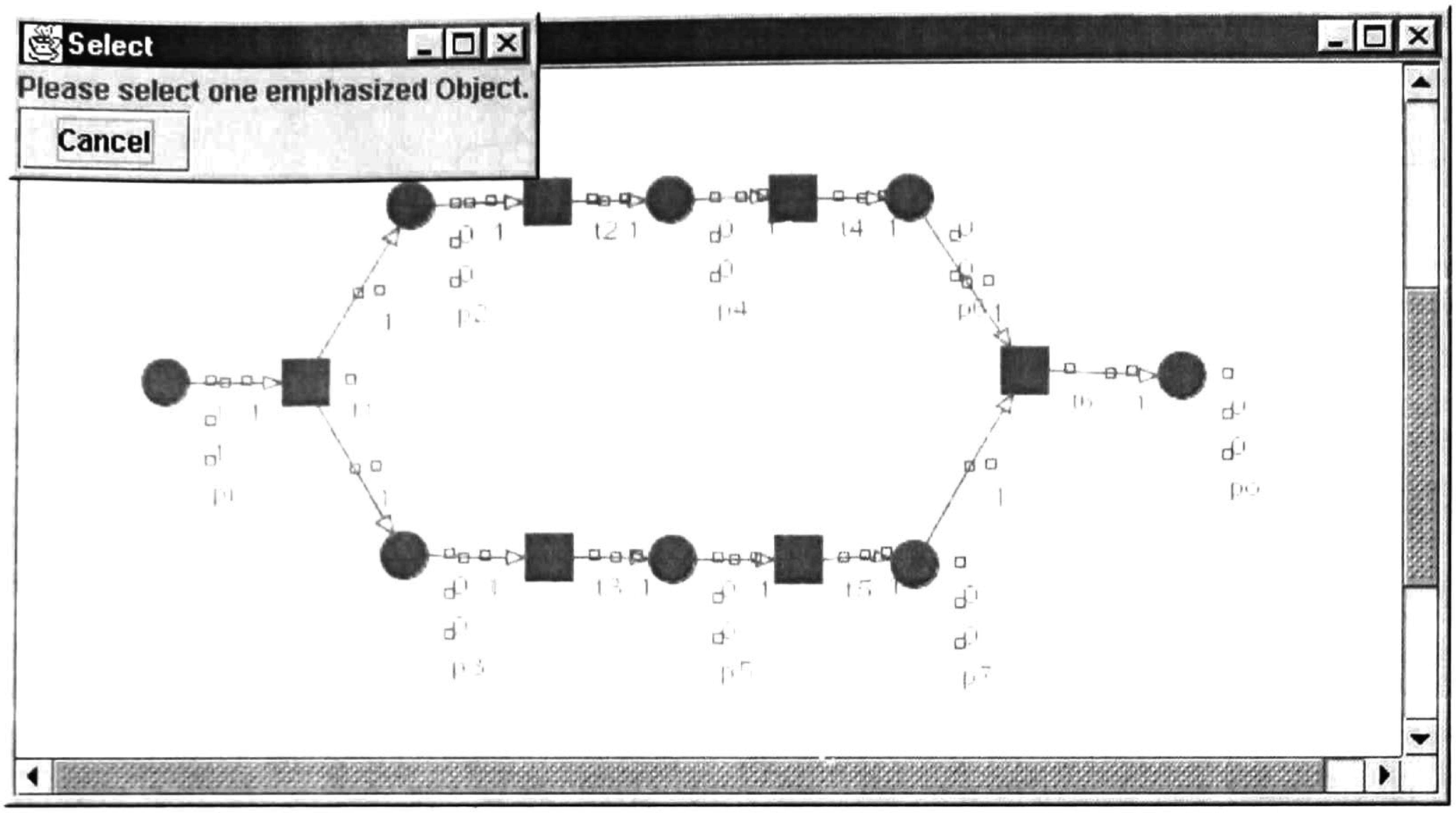
```
public final class Place extends Node
  public Place(Net net, String name, Object initiator)
  public void delete(Object initiator)
  public Marking getMarking()
  public void setMarkingAsInitial()
} // class Place
```

- Transition : This class describes a transition in Petri Nets. As in the place class, when it is created a new place, it register itself into the net passed as constructor's parameter. It is possible to know the input and output edges of a transition, etc.

```
public final class Transition extends Node {
  public Transition(Net net, String name, Object initiator)
  public void delete(Object initiator)
  public Mode getMode()
} // class Transition
```

5.3 A module for implementing RWNets

The software developed in this work is mainly divide into: the set of rewriting transitions, the substitution for RWNets, and the Firing rule for RWNets, Figure 5.4.


Select

Please select one emphasized Object.

Figure 5.5: The PNK Subnet application

- The set of rewriting transitions: this set of rewriting transitions were introduced at the beginning of this chapter, and were implemented as Java classes.

- The substitution for RWNets: this function implements the Substitution for RWNets definition (see the RWNets section in the previous chapter). Note that this function implementation is a simplified version of the original definition, since the rewriting transitions implemented in this section rewrite only ordinary Petri Nets.

- The Firing Rule for RWNets: this function implements the Firing Rule definition for RWNets (see the RWNets section in the previous chapter); in short, this function makes the structural changes after the firing of a transition in RWNets.

Also, as an auxiliary application, it was developed the Subnet Application. This application allows the user to select the subnet of the current net in which the user wants to work, Figure 5.5. The PNK Subnet application requests the user for the selection of one emphasized object, then it keeps the subnet selected by the user -simulating in this way, the matching for the pre set of a rewriting transition-.

The set of transformation rules that allows to perform structural changes over a Petri Net, are implemented as follows:

Problem 26 (Cancel Case) *This problem deals with the cancellation of a workflow case, i.e., one workflow instance currently executed must be abruptly finished. Figure 5.6 shows a rewriting transition that meets this problem. As it is shown, in a symbolic way, any marking on the net is mapped to its END state.*

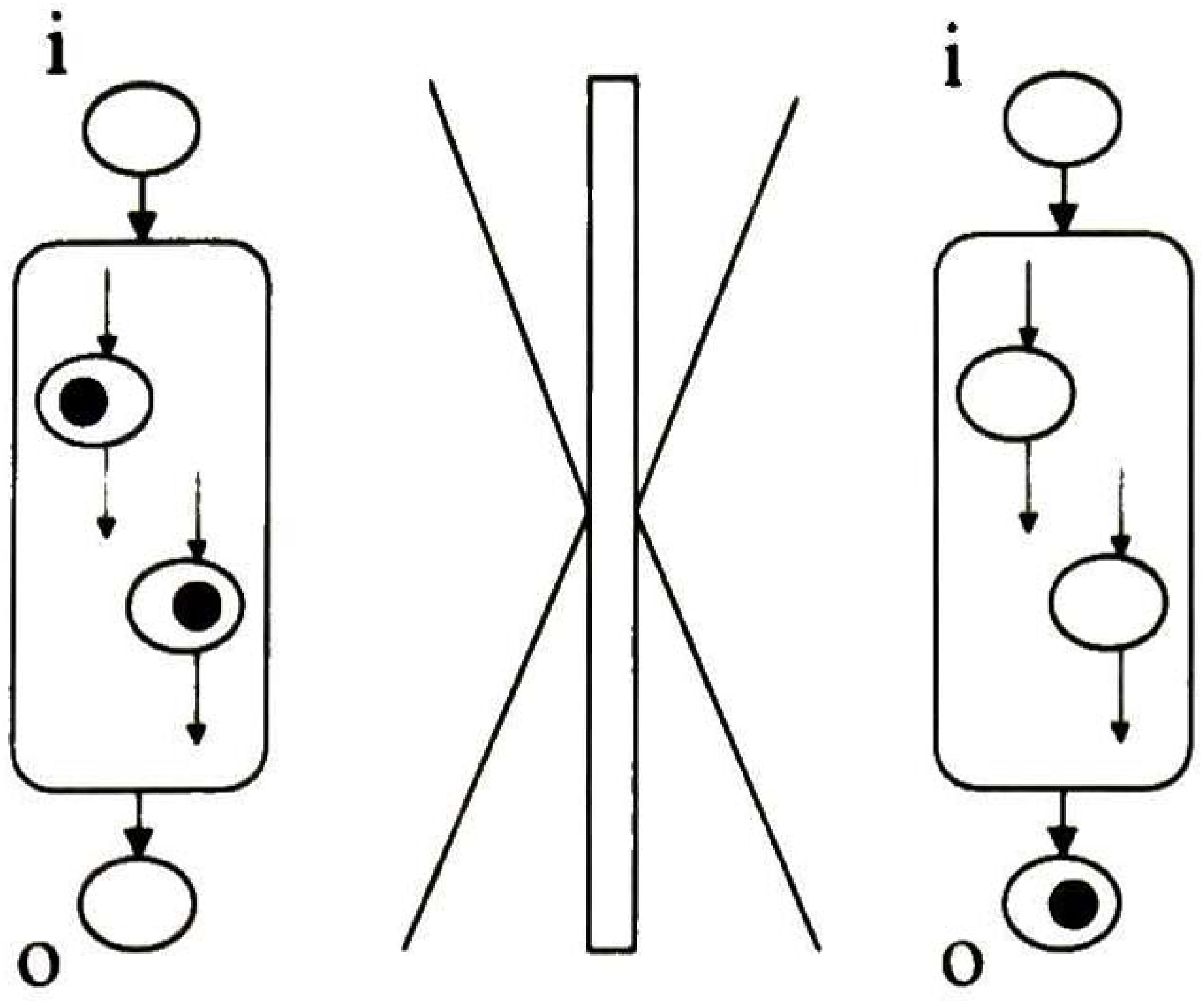


Figure 5.6: The Cancel Case Rewriting Rule

Solution 27 (Cancel Case Rewriting Rule) *CancelRule.java: This Java class sends any reachable marking over the actual net to its END state, i.e., the marking with one token in its output place "o" Figure 5.7 shows the effect produced by the application of this rule over the net on the PNK editor of the Figure 5.2. As it is shown, the workflow net was "canceled"*

Problem 28 (Reset Case) *This problem deals with the re-initialization of a workflow case, i.e., one workflow process currently executed must be abruptly restarted. Figure 4.16 shows a rewriting transition that meets this problem. As it is shown, in a symbolic way, any making on the net is mapped to its START state.*

Solution 29 (Reset Case Rewriting Rule) *ResetRule.java: This Java class sends any reachable marking over the actual net to its START state, i.e., the marking with one token in its input place "i" Figure 5.8 shows the effect produced by the application of this rule over the net on the PNK editor of the Figure 5.2. As it is shown, the workflow net was "restarted"*

Problem 30 (Pattern by Pattern) *This problem deals with the replacement of a Workflow pattern by another one. Figure 4.14 shows a rewriting transition that meets this problem. As it is shown, a certain workflow pattern, on the pre set of the transition, is be changed by another workflow pattern, on the post set of the transition.*

Solution 31 (PatternByPatternRule) *PatternByPatternRule.java: This java class replace a certain Workflow pattern present in the actual net by another one elected by the user. Figure 5.9 shows the effect produced by the application of this rule over the net on the PNK editor of the Figure 5.2. As it is shown, a Simple Activity pattern has been changed by the XOR-Split pattern.*

Problem 32 (Interchange two Transition) *This problem deals with the change of order between two sequential transitions. Figure 4.15 shows a rewriting transition that meets this problem. As it is shown, two sequential transitions are changed by the same two activities but in inverse order.*

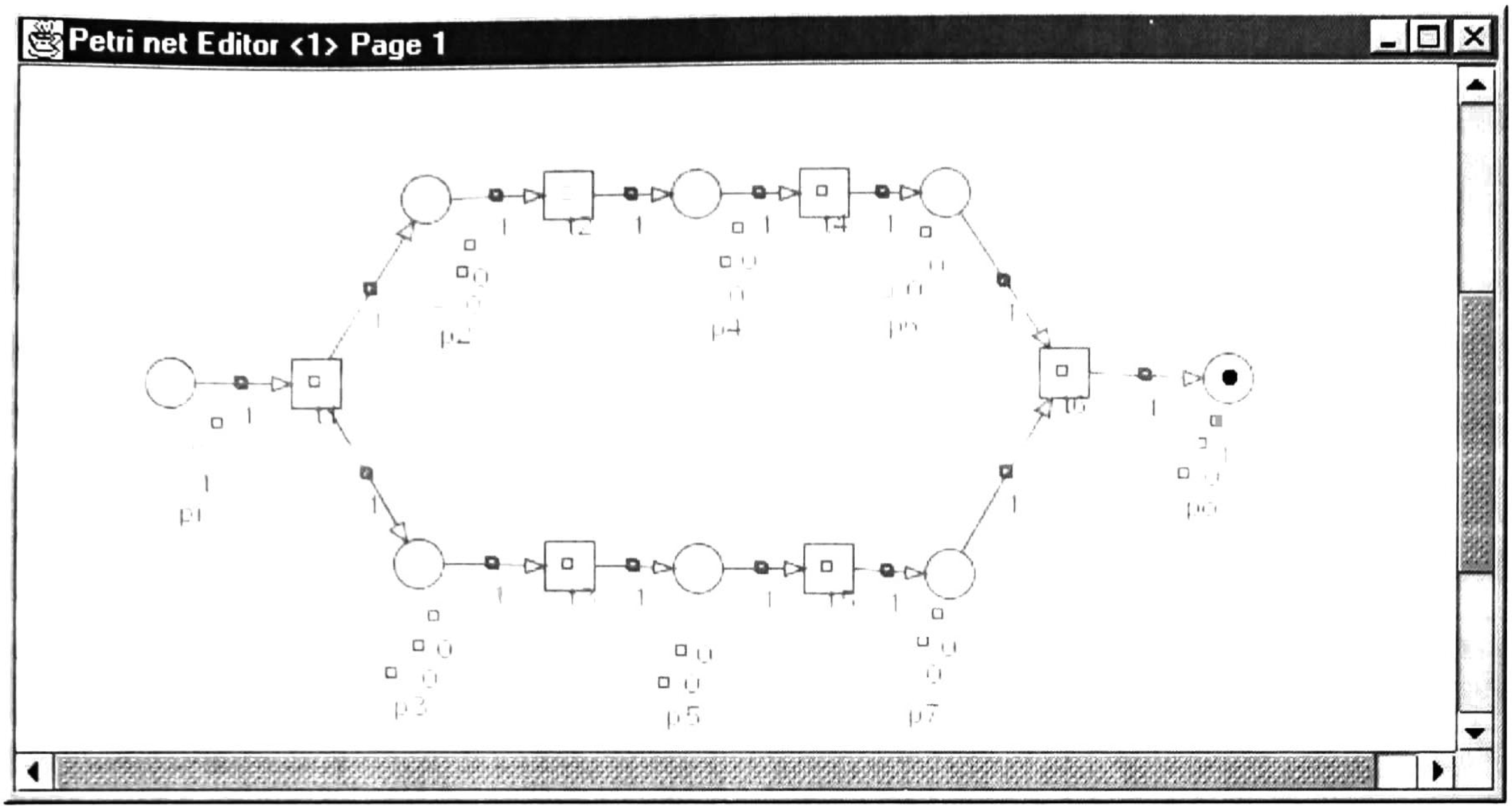


Figure 5.7: The Cancel Case Rewriting Rule efect

Solution 33 (Interchange2Transitions) *Interchange2Transitions.java: This java class interchanges two sequential activities, i.e., if it is given two sequential activities A ⟶ B, after apply this rewriting transition these transitions must have the new sequential relation B ⟶ A. Figure 5.10 shows the effect produced by this rule over the net on the PNK editor of the Figure 5.2. As it is shown, transitions t2 and t4 have now an inverse order.*

Problem 34 (Sequence to Parallel) *This problem deals with the change of two sequential transitions for its parallel execution. Figure 5.11 shows a rewriting transition that meets this problem. As it is shown, two sequential transitions are changed into its parallel representation.*

Solution 35 (SequenceToParallelRule) *SequenceToParallel.java: This java class performs the next action: given two sequential transitions, it performs its parallel representation. Figure 5.12 shows the effect produced by the application of this rule over the net on the PNK editor of the Figure 5.2. As it is shown, the transitions t2 and t4 have now a parallel relationship.*

Problem 36 (Parallel to Sequence) *This problem deals with the change of two parallel transitions into its sequential execution. Figure 5.13 shows a rewriting transition that meets this problem. As it is shown, two parallel transitions are changed into its sequential representation.*

Solution 37 (ParallelToSequenceRule) *ParallelToSequence.java: This java class, obviously, performs the sequential representation for two given parallel ones. Figure 5.14 shows the effect produced by by the application of this rule over the net on the PNK editor of the Figure 5.12. As it is shown, transitions t2 and t4 have now a sequential relation.*

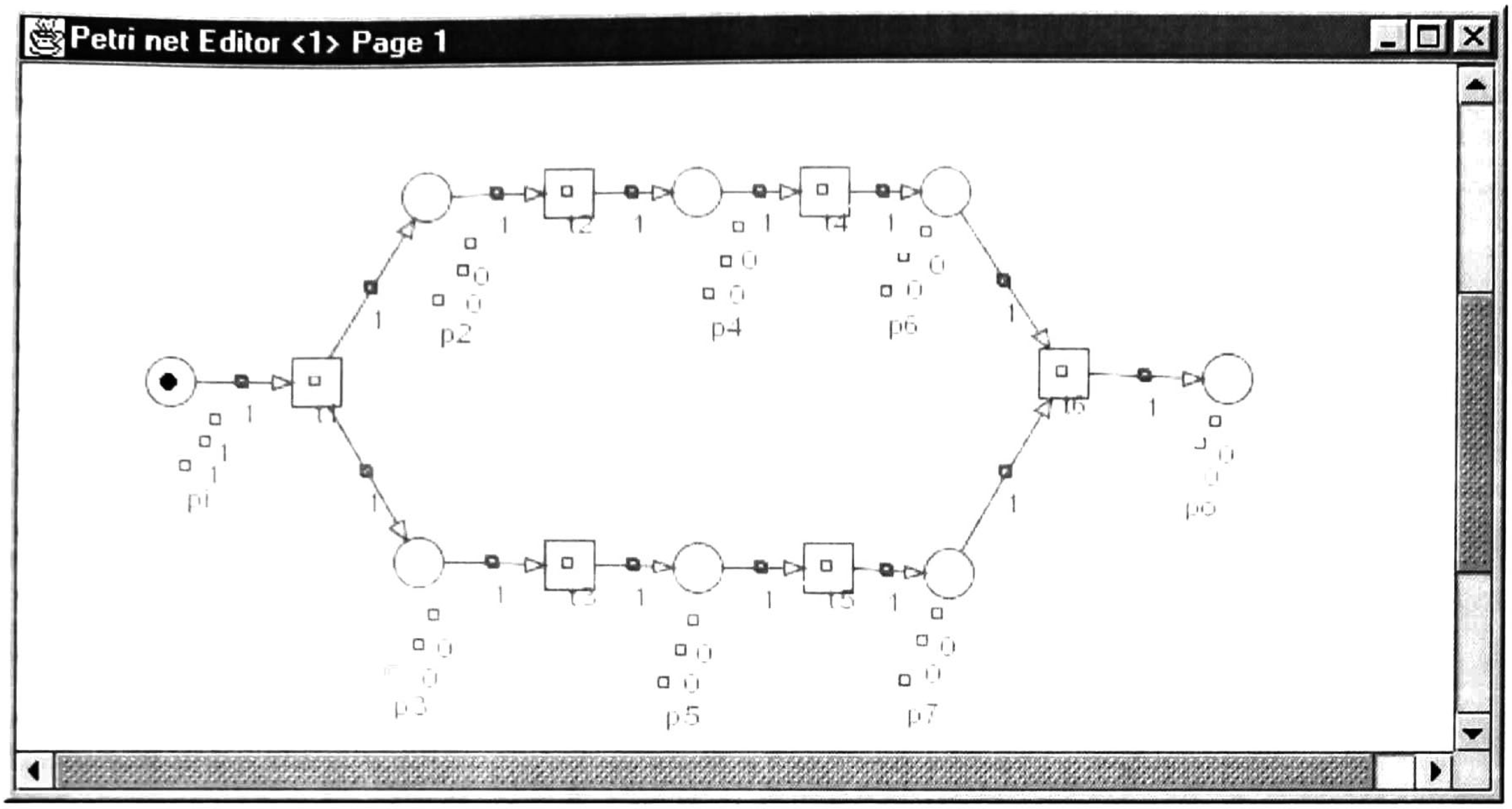


Figure 5.8: The Reset Case Rewriting Rule efect

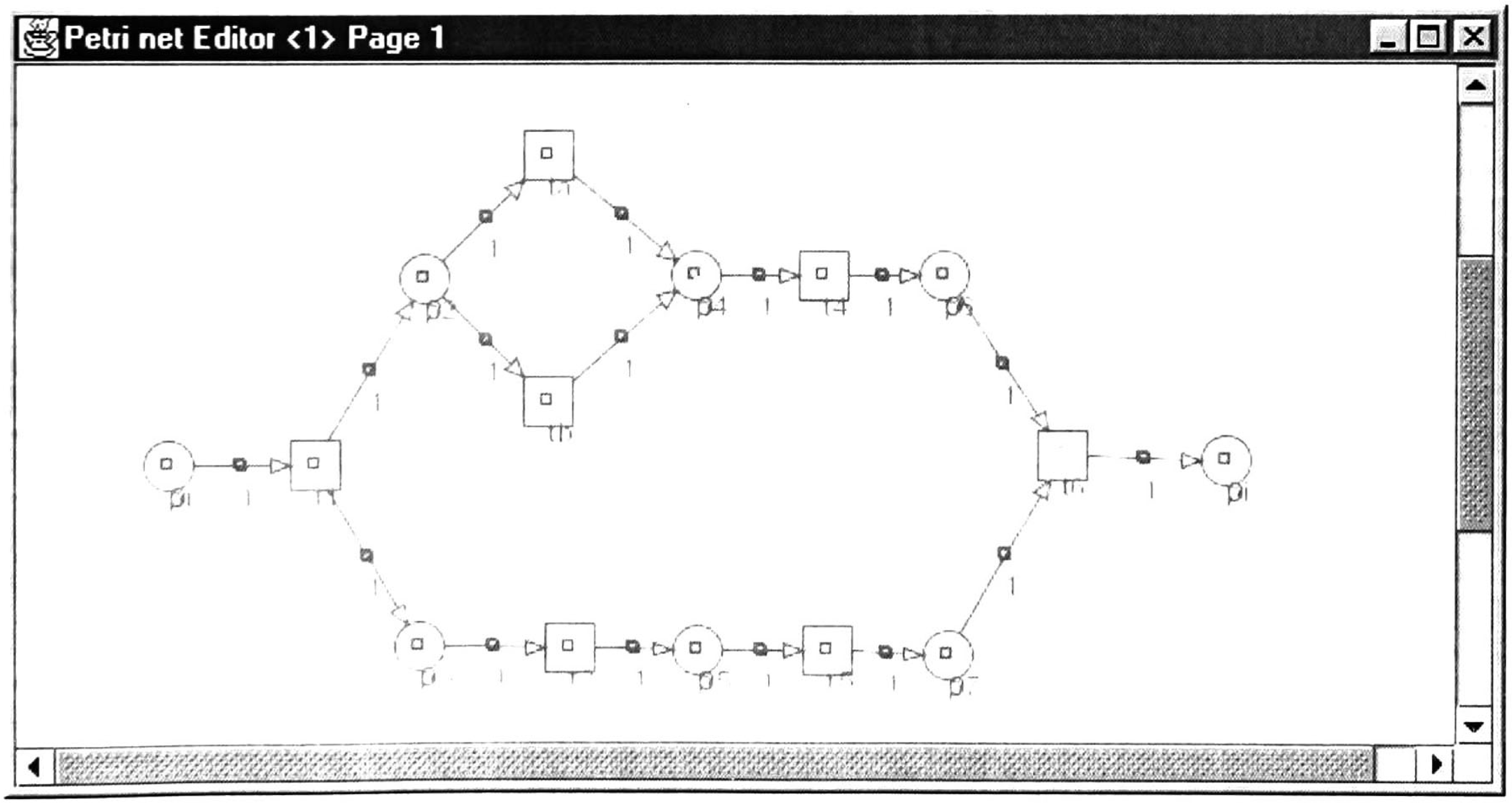


Figure 5.9: The Pattern-by-Pattern Rewriting Rule

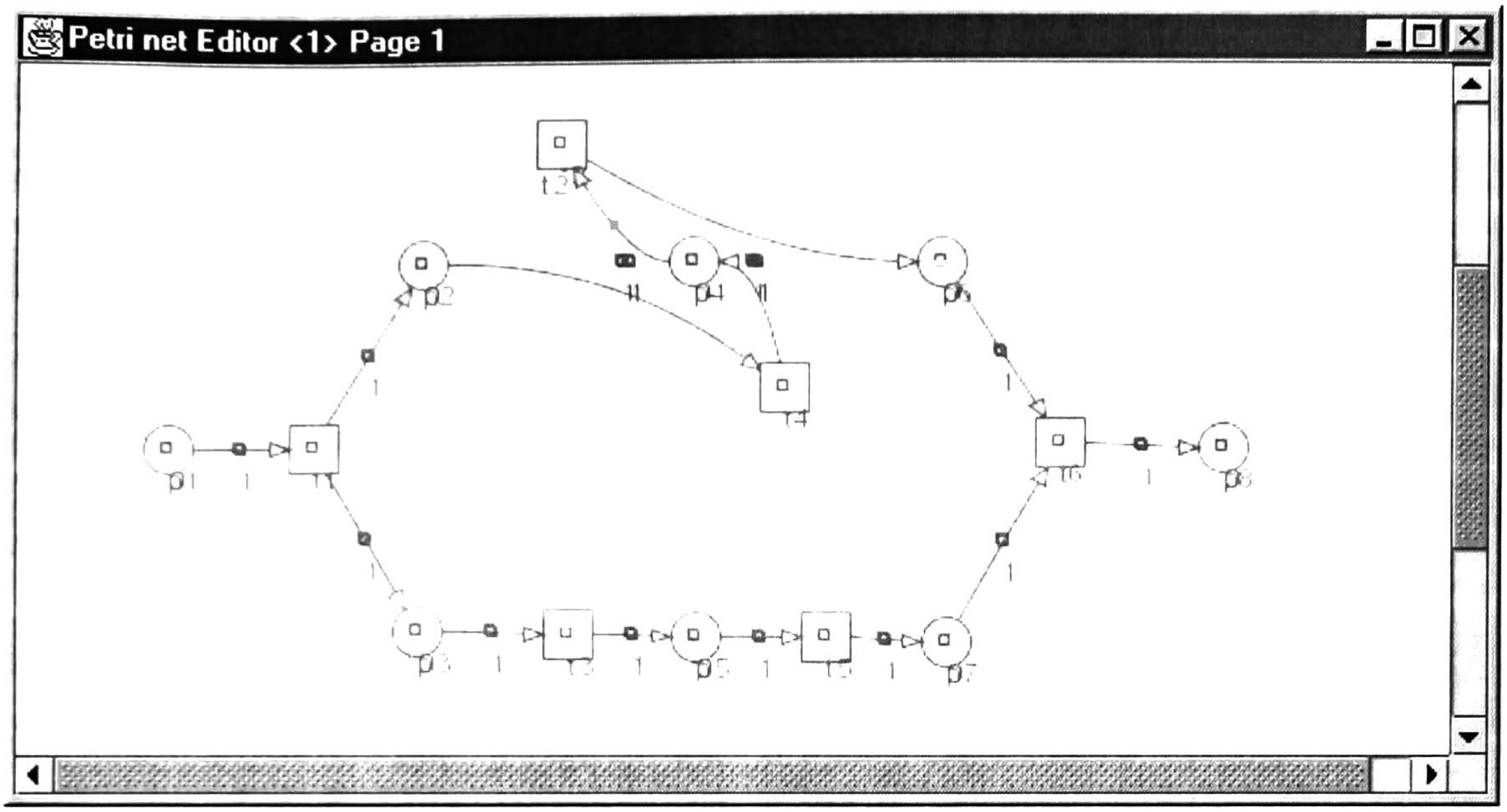


Figure 5.10: The 2-Interchange-Transitions Rewriting Rule

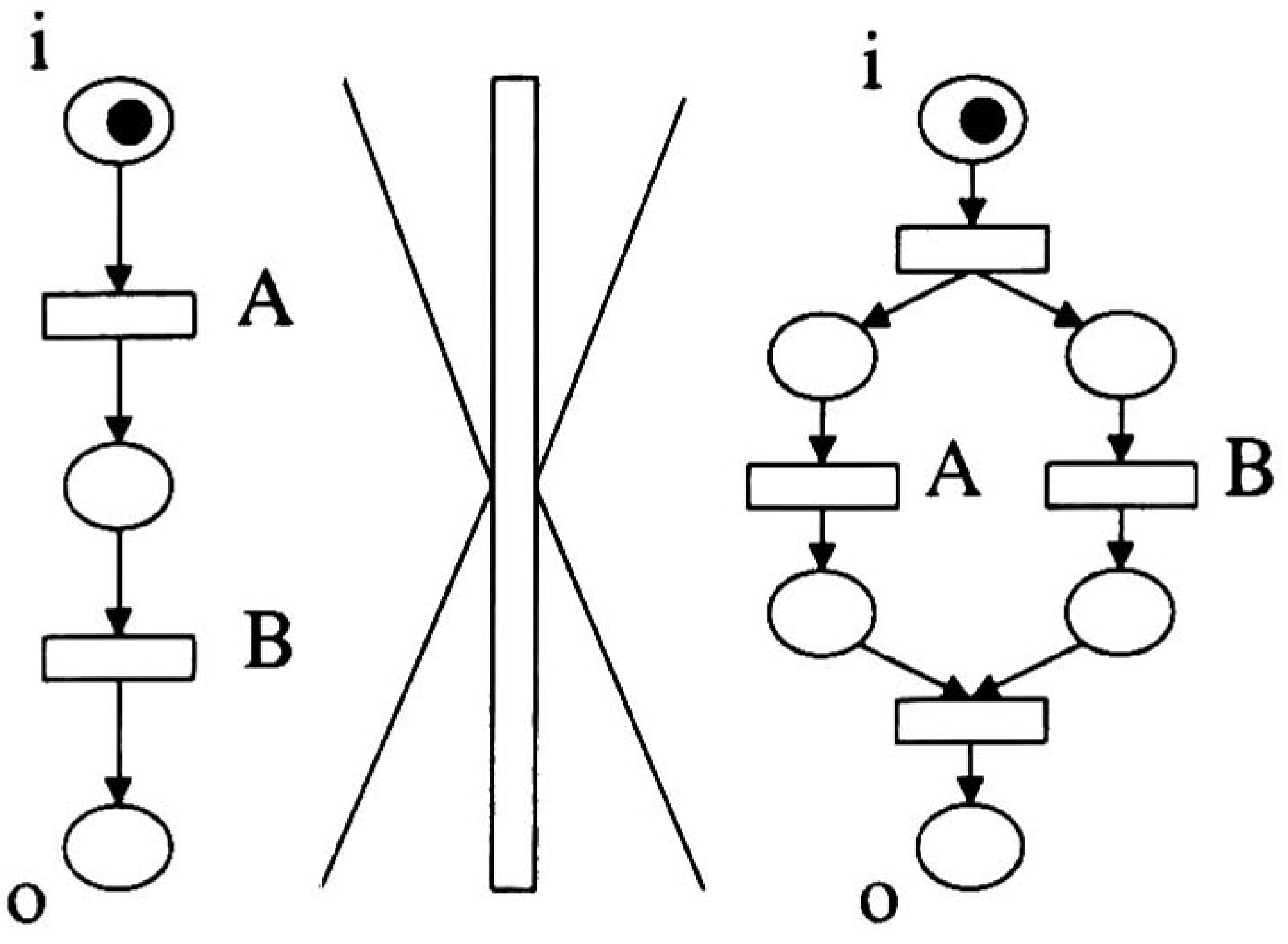


Figure 5.11: The Sequence-to-Parallel Rewriting Net

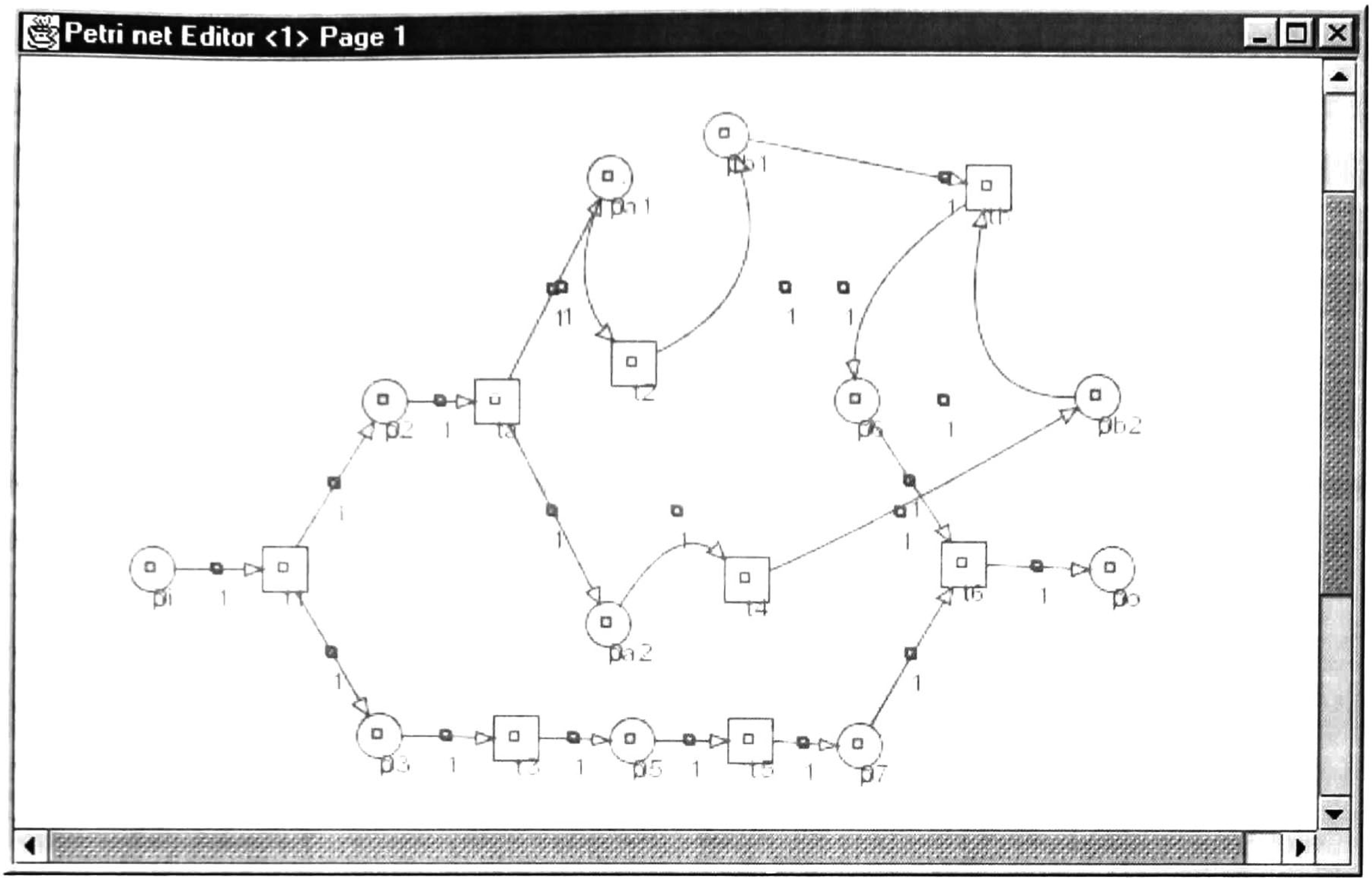


Figure 5.12: The Sequence-to-Parallel Rewriting Rule

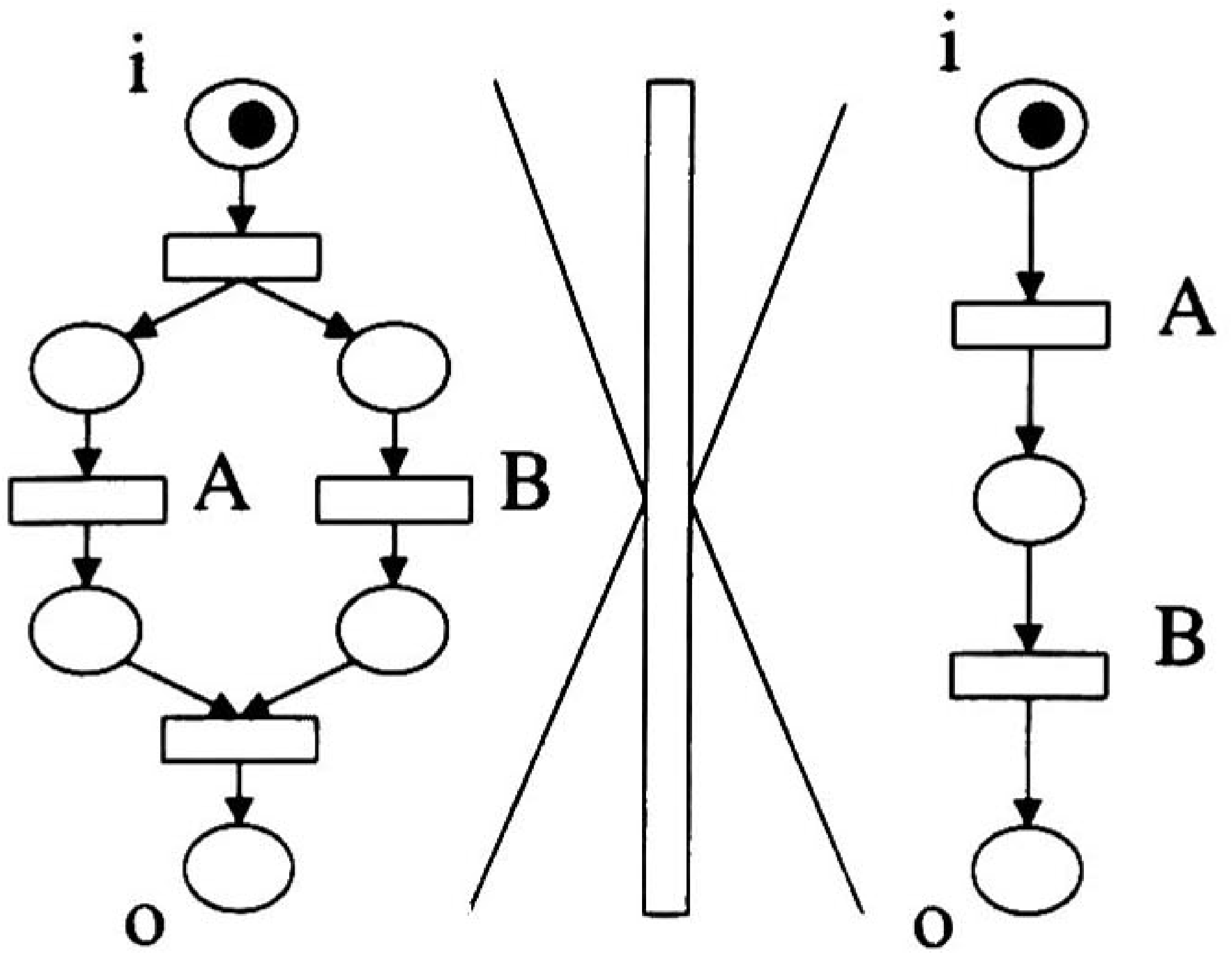


Figure 5.13: The Parallel-to-Sequence Rewriting Net

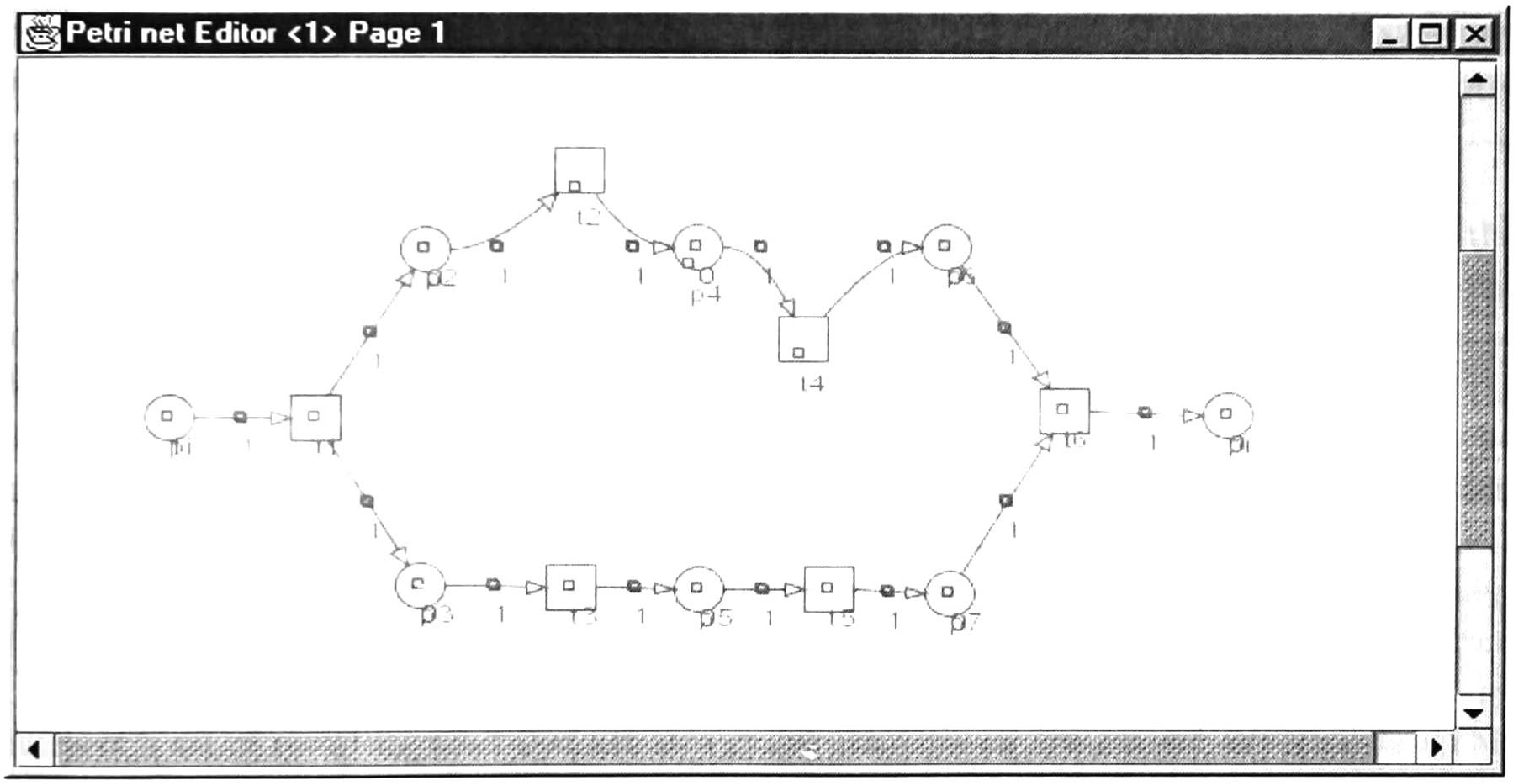


Figure 5.14: The Parallel-to-Sequence Rewriting Rule

Chapter 6

Conclusions

This thesis addressed the problems of modular modeling and dynamic change in workflow management systems (WFMS). For the first problem, synthesis method for the incremental construction of complex workflow models, expressed as workflow nets (WFN), a subclass of Petri nets, was proposed. In order to cope with the dynamic change problem, it was proposed Rewriting nets (RWN), an extension to dynamic nets that allows structural changes on WFN models.

This adaptive synthesis method allows to build progressively sound WFN models avoiding the analysis of properties of large and complex final models. The method is suitable for the initial definition of a WFMS and it is easy to handle since it is based on the successive composition of WFN models that equivalent to well known workflow patterns.

The strategy of the synthesis method is based on the refinement of places; this strategy may be complemented with refinement of transitions, yielding a more complete method for the construction of complex workflow models.

Regarding the dynamic change problem, it has been shown that RWN supports the ability of a WFMS for on-line adapting of its internal WFN model. Thus modifications issued from environmental changes, new strategies, or process reengineering can be easily implemented by means of a set of rewriting rules that act as a graph grammar on the current WFN model. This has been illustrated through an implementation of a reconfigurable WFMS; the prototype built into PNK shows that the core of a WFMS based on a WFN model can be modified during the execution.

In this work the workflow model and the set rewriting rules (expressed as RWNets) are separately handled; however it could be interesting to consider the case in which both model and rules are specified using RWNets. On the other hand, we feel that the relationship between RWNets and graph grammars should be more deeply studied exploiting existing results for this last class of formalisms.

Bibliography

[Aalst, 1997] W.M.P. van der Aalst. Three Good Reasons for Using a Petri-net-based Workflow Management System. In S. Navathe and T. Wakayama, editors, Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96), pages 179-201, Cambridge, Massachusetts, Nov 1996.

[Aalst, 1998] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, 8(1):21-66, 1998.

[Aalst, Agust 1999] W.M.P. van der Aalst, T. Basten, H.M.W. Verbeek, P.A.C. Verkoulen, and M. Voorhoeve. Adaptive Workflow: An Approach Based on Inheritance. In M. Ibrahim and B. Drabble, editors, Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business, pages 36-45, Stockholm, Sweden, August 1999.

[Aalst, 1999] W.M.P. van der Aalst. Flexible Workflow Management Systems: An Approach Based on Generic Process Models. In T. Bench-Capon, G. Soda, and A. Min-Tjoa, editors, Proceedings of the 10th International Conference on Database and Expert Systems Applications (DEXA'99), volume 1677 of Lecture Notes in Computer Science, pages 186-195. Springer-Verlag, Berlin, 1999.

[Aalst, 1999 UGA] W.M.P. van der Aalst. How to Handle Dynamic Change and Capture Management Information: International Journal of Computer Systems, Science, and Engineering, 16(5):295-318, 2001.

[Aalst, 2000 IS] W.M.P. van der Aalst. Process-oriented Architectures for Electronic Commerce and Interorganizational Workflow. Information and Systems, 24(8):639-671, 2000.

[Aalst, 2000, WP 47] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. QUT Technical report, FIT-TR-2002-02, Queensland University of Technology, Brisbane, 2002.

[Aalst, 2000 WP 50] W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An approach to tackling problems related to change. Computing Science Reports 99/06, Eindhoven University of Technology, Eindhoven, 1999.

[Aalst, 2001 WP 51] W.M.P. van der Aalst. Exterminating the Dynamic Change Bug: A Concrete Approach to Support Change. BETA Working Paper Series, WP 51, Eindhoven University of Technology, Eindhoven, 2000.

[Agostini, 1998] A. Agostini and G. De Michelis. Simple Workflow Models. In W.M.P. van der Aalst, G. De Michelis, and C.A. Ellis, editors, Workflow Management: Net-based Concepts, Models, Techniques, and Tools (WFM98), Proceedings, pages 146164, Lisbon, Portugal, June 1998. Eindhoven University of Technology, Eindhoven, The Netherlands, Computing Science Report 98/7, 1998.

[Asperti, 1996] Asperti, A., and Busi, N., Mobile Petri Nets. Technical Report UBLCS-96-10, University of Bologna, Italy (1996).

[Badouel, 1998] E. Badouel and J. Oliver. Reconfigurable nets: a class of high level Petri nets supporting dynamic changes with workflow systems. Inria Research Report PI1163, 1998.

[Blostein, 1996] D. Blostein and H. Fahmy and A. Grbavec. Practical use of graph rewriting. pages: 95-373, CDN, 1995.

[Buscemi, 2001] Maria Grazia Buscemi and Vladimiro Sassone. High-Level Petri Nets as Type Theories in the Join Calculus. FOSSACS01. Ed. F. Honsell and M. Miculan. Num. 2030. Serie: LNCS. Pags: 104-120. Springer 2001.

[Casati, 1996] Fabio Casati, Stefano Ceri, Barbara Pernici, and Giuseppe Pozzi. Workflow Evolution. International Conference on Conceptual Modeling / the Entity Relationship Approach, pages: 438-455, 1996.

[Corradini, 1995] Andrea Corradini. Concurrent computing: from Petri nets to graph grammars, Electronic Notes in Theoretical Computer Science, Vol. 2, Elsevier Science Publishers, A. Corradini and U. Montanari, editors, 2000.

[Corradini, 1997] Corradini, Andrea and Ehrig, Hartmut and Heckel, Reiko and Korff, Martin and Loewe, Michael and Ribeiro, Leila and Wagner, Anika. Algebraic Approaches to Graph Transformation Part {I}: Single Pushout Approach and Comparison with Double Pushout Approach. Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations. World Scientific, Rozenberg, G., pages 247–312, 1997.

[Desel, 1995] Jörg Desel, and Javier Esparza. Free Choice Petri nets. Cambridge Tracts in Theoretical Computer Science No. 40. Cambridge University Press, 1995.

[Ellis, 1995] Ellis, C., Keddara, K., and Rozenberg, G., Dynamic Change within Workflow Systems. Proceedings of the Conference on Organizational Computing Systems, ACM Press, New York (1995)10–21.

[Forunet, 1995] Cedric Fournet and Georges Gonthier. The Reflexive CHAM and the Join-Calculus. TProceedings of the 23rd {ACM} Symposium on Principles of Programming Languages, ACM Press, pages: 372–385, 1996.

[Han, 1998] Yanbo Han, Amit Sheth, and Christoph Bussler. A Taxonomy of Adaptive Workflow Management. CSCW-98 Workshop: Towards Adaptive Workflow Systems.

[Jeng, 1992] Jeng, M., F.DiCesare. (1993) A review of synthesis techniques for Petri nets with applications to Automated Manufacturing systems. IEEE Transactions on Systems, Man, and Cybernetics. Vol. 23, No. 1, pp. 301-312, Jan/Feb.

[Jensen,1981] K. Jensen. Colored Petri Nets and the Invariant Method. Theoretical Computer Sciences, 1981, Vol. 14, pp. 317-336.

[Jiang, 2001] Jiang Zhibin, Zuo Ming J., Fung Richard Y.K., Tu Paul Y.L., Colored Petri Nets with changeable structures (CPN-CS) and their applications in modeling one-of-a-kind production (OKP) systems, Computers And Industrial Engineering (41)3 (2001) pp. 279-308.

[Koh, 1991] Koh, I., F. DiCesare. (1991) Modular transformation methods for generalized Petri Nets and their Application to Automated Manufacturing Systems. IEEE Trans. on Systems, Man, and Cybernetics, Vol. 21, No. 6, pp. 1512-1522, Nov/Dec.

[Kindler, 2001] E. Kindler, M. Weber: The Petri Net Kernel An Infrastructure for Building Petri Net Tools. In: Software Tools for Technology Transfer (STTT) 3(4), pages 486-497, September 2001.

[Lane, 1971] S. Mac Lane. Categories for the working mathematician. Springer Verlag, 1971.

[Lopez, 1997] Ernesto Lopez M. Introduction a las Redes de Petri. Facultad de Ciencias Fisico-Matematicas, Universidad Autonoma de Nuevo Leon, Octubre de 1997.

[Padberg, 1998] Julia Padberg and Magdalena Gajewsky and Kathrin Hoffmann. Incremental Development of Safety Properties in Petri Net Transformations, TAGT, pages: 410-425, 1998.

[Padberg, 1998 LNCS] J. Padberg and M. Gajewsky and C. Ermel. Rule-Based Refinement of High-Level Nets Preserving Safety Properties, Lecture Notes in Computer Science 1382, pages: 221–??, 1998.

[Petri, 1962] Petri, C.A., Kommunikation mit Automaten" Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962. In German.

[Tsalgatidou, 1996] Tsalgatidou, A., Louridas, P., Fesakis, G. & Schizas, T. 1996. Multilevel Petri Nets for Modeling and Simulating Organizational Dynamic Behavior, Simulation & Gaming, Special Issue on Simulation of Information Systems, 27 (4), pp. 484-506.

[Weber, 2002] Michael Weber, Ekkart Kindler. The Petri Net Markup Language. "Petri Net Technology for Communication Based Systems" within the LNCS series "Advances in Petri Nets". April 2002.

[Workflow Handbook, 2001] Rob Allen, Open Image Systems Inc., UK. Section 1-The World of Workflow, Workflow: An Introduction, The Workflow Handbook 2001. Published in association with the Workflow Management Coalition (WfMC), Edited by Layna Fischer, OCTOBER 2000, pag. 15-38. ISBN 0-9703509-0-2.

[WFMC-TC-1011] Workflow Management Coalition, Terminology & Glossary, Document Number WFMC-TC-1011, Issue 3.0, February 1999.

[WfMC-TC-1016] Workflow Management Coalition. Interface 1: Process Definition Interchange Q&A and Examples. Document Number WfMC TC-1016-X. Document Status: Draft 7.01. Issued on January 1, 1999. Author: Work Group 1.

[Zhou, 1992] A Hybrid Methodology for Synthesis of Petri Net Models for Manufacturing Systems, MengChu Zhou, Frank DiCesare, Alan A. Desrochers, IEEE Trans. on Robotics and Automation, Vol. 8, No.3, 1992; pages: 350-361.

Centro de Investigación y de Estudios Avanzados del IPN

Unidad Guadalajara

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó la tesis: Modelado Adaptativo y Modular de Sistemas de Gestión de Flujo de Trabajo del(a) C. Raúl CAMPOS RODRÍGUEZ el día 18 de Octubre de 2002

DR. LUIS ERNESTO LÓPEZ MELLADO
INVESTIGADOR CINVESTAV 3A
CINVESTAV GDL
GUADALAJARA

DR. ANTONIO RAMIREZ TREVIÑO
INVESTIGADOR CINVESTAV 2A
CINVESTAV GDL
GUADALAJARA

DR. FÉLIX FRANCISCO RAMOS CORCHADO
INVESTIGADOR CINVESTAV 2A
CINVESTAV GDL
GUADALAJARA