Verifica Formale in Spin di WF-nets e Diagrammi delle Attività UML

Seminario per il corso di Metodi Formali nell'Ingegneria del Software

Professore: Toni Mancini

Autore: Stefano Menotti

Obiettivi Principali

- Sviluppare una metodologia per la verifica formale in Spin delle ***Workflow Nets*** (formalismo utilizzato per la modellazione dei processi di workflow; costituiscono una classe delle più generali ***Reti di Petri***).

- Applicare le idee sviluppate per la verifica di una Workflow Net ai ***Diagrammi delle Attività UML***.

Reti di Petri (1)

- Formalismo proposto nel 1962 da Carl Adam Petri per descrivere ed analizzare sistemi concorrenti.
- Consentono di dare al modello di un sistema il rigore formale necessario sia per eliminare ogni fonte di ambiguità nella rappresentazione, sia per effettuare analisi e verifiche sul comportamento del sistema.
- Rappresentazione grafica semplice.

Reti di Petri (2)

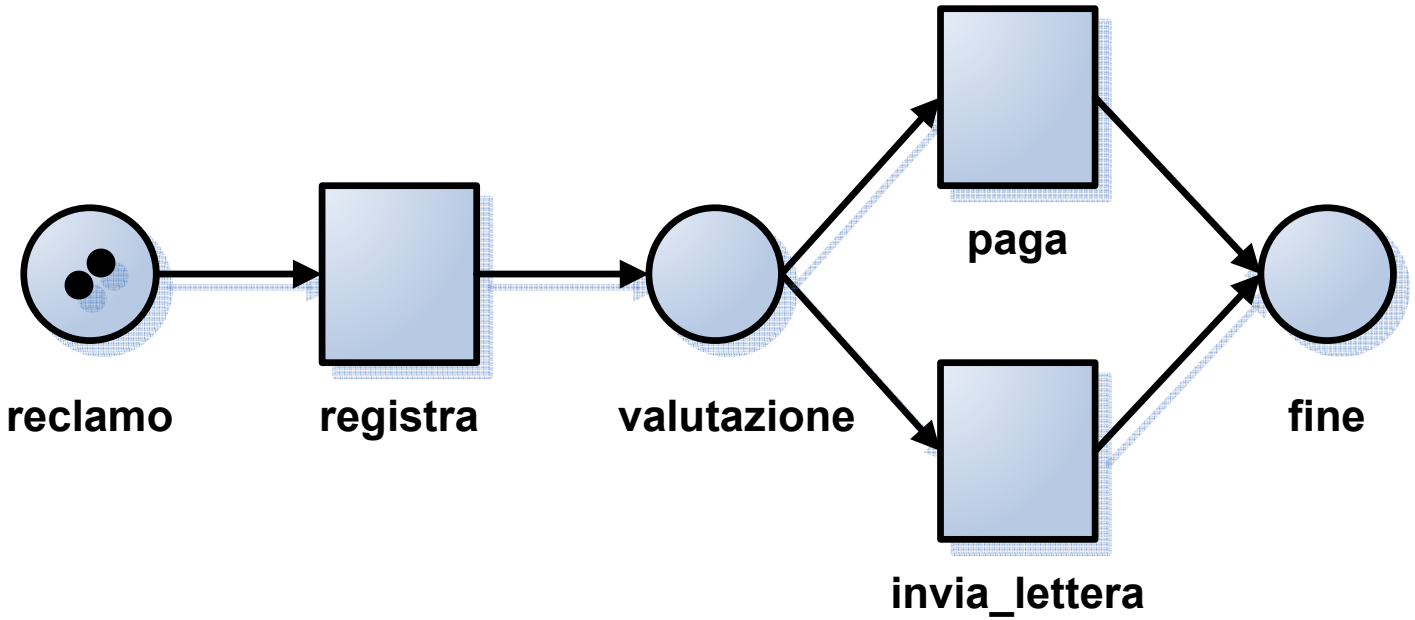
- ***Place****:* definiscono lo stato del sistema. Rappresentati mediante un cerchio.
- ***Transizioni***: componenti attive della rete; modificano lo stato del sistema. Rappresentati mediante un rettangolo.
- ***Archi***: da place a transizione e da transizione a place.
- ***Token:*** rappresentano oggetti, informazioni. Rappresentati mediante un cerchietto nero.

Reti di Petri (3)

- Un place *p* è un ***input place*** per una transizione *t* se e solo se esiste un arco diretto che collega *p* a *t*.
- Un place *p* è un ***output place*** per una transizione *t* se e solo se esiste un arco diretto da *t* a *p*.
- Una transizione "***scatta***" quando è abilitata, cioè nel momento in cui è presente almeno un token in ciascuno dei suoi input place.
- Quando scatta, una transizione "***consuma***" un token da ciascuno dei suoi input place e "***produce***" un token in ciascuno dei suoi output place.

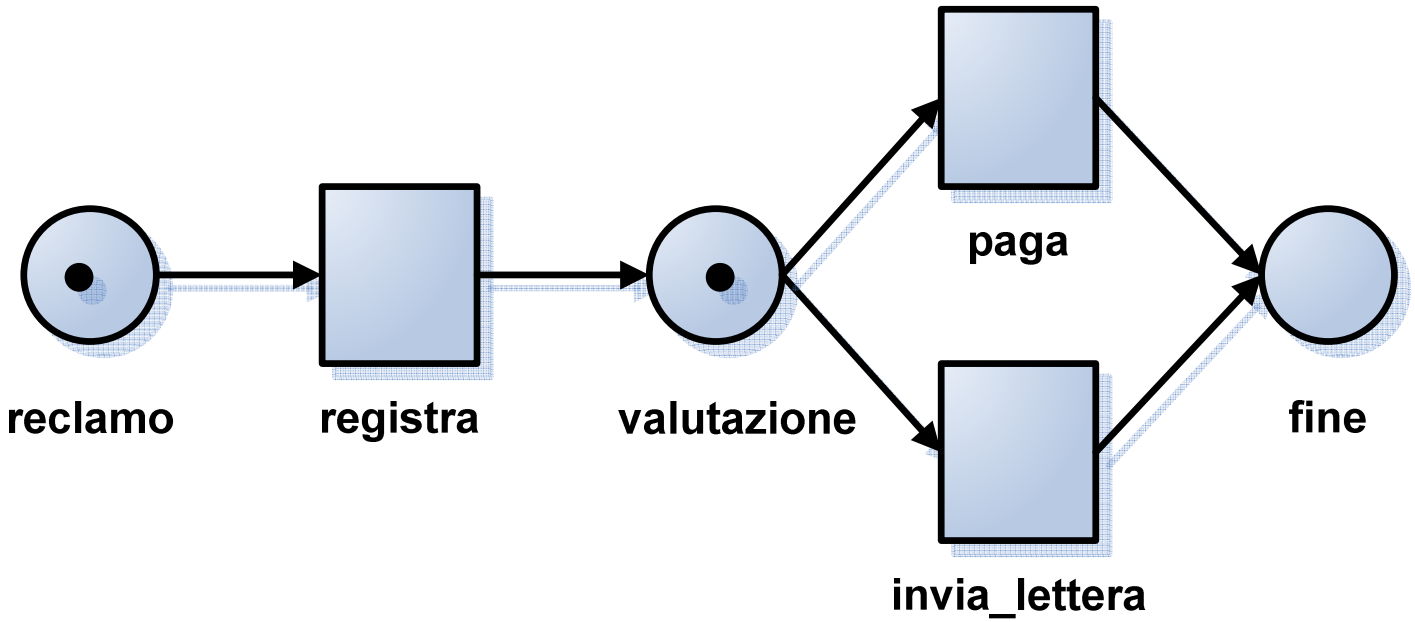
Reti di Petri - Esempio (1)

- Rete di Petri per il processo di gestione di un reclamo in una compagnia di assicurazioni.
- Stato: (2, 0, 0)



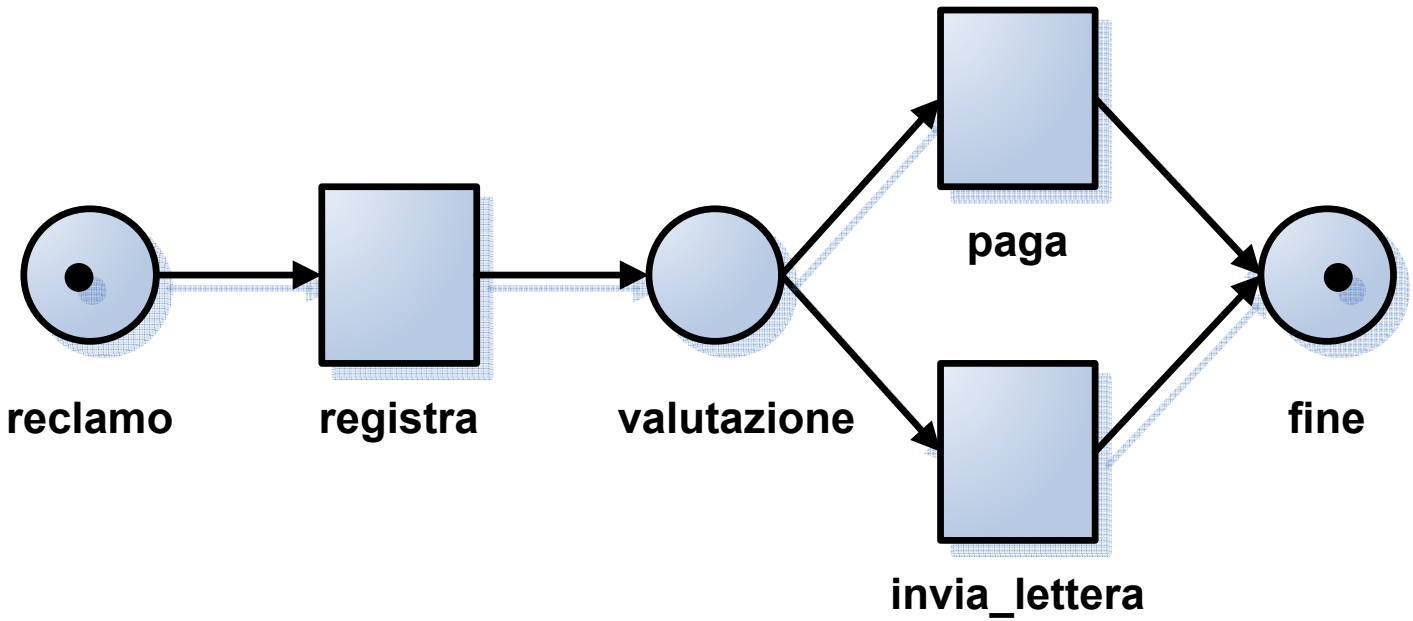
Reti di Petri - Esempio (2)

- Stato: (1, 1, 0)



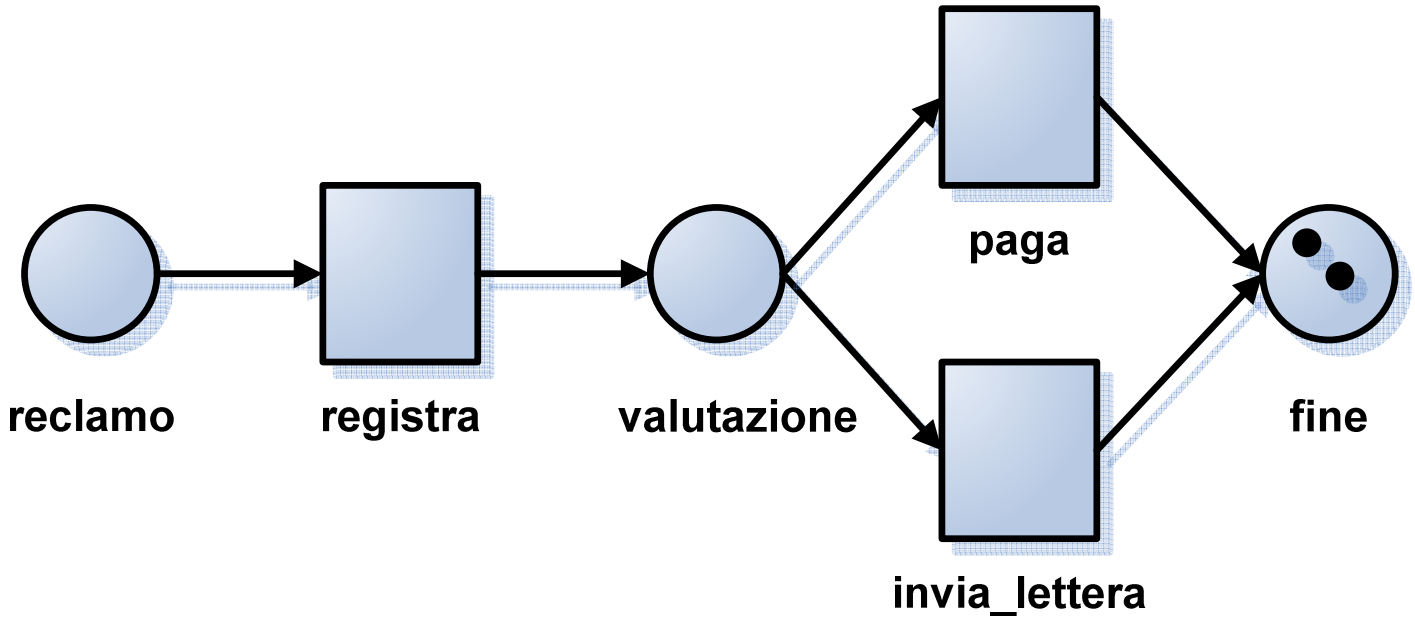
Reti di Petri - Esempio (3)

- Stato: (1, 0, 1)



Reti di Petri - Esempio (4)

- Stato: (0, 0, 2)



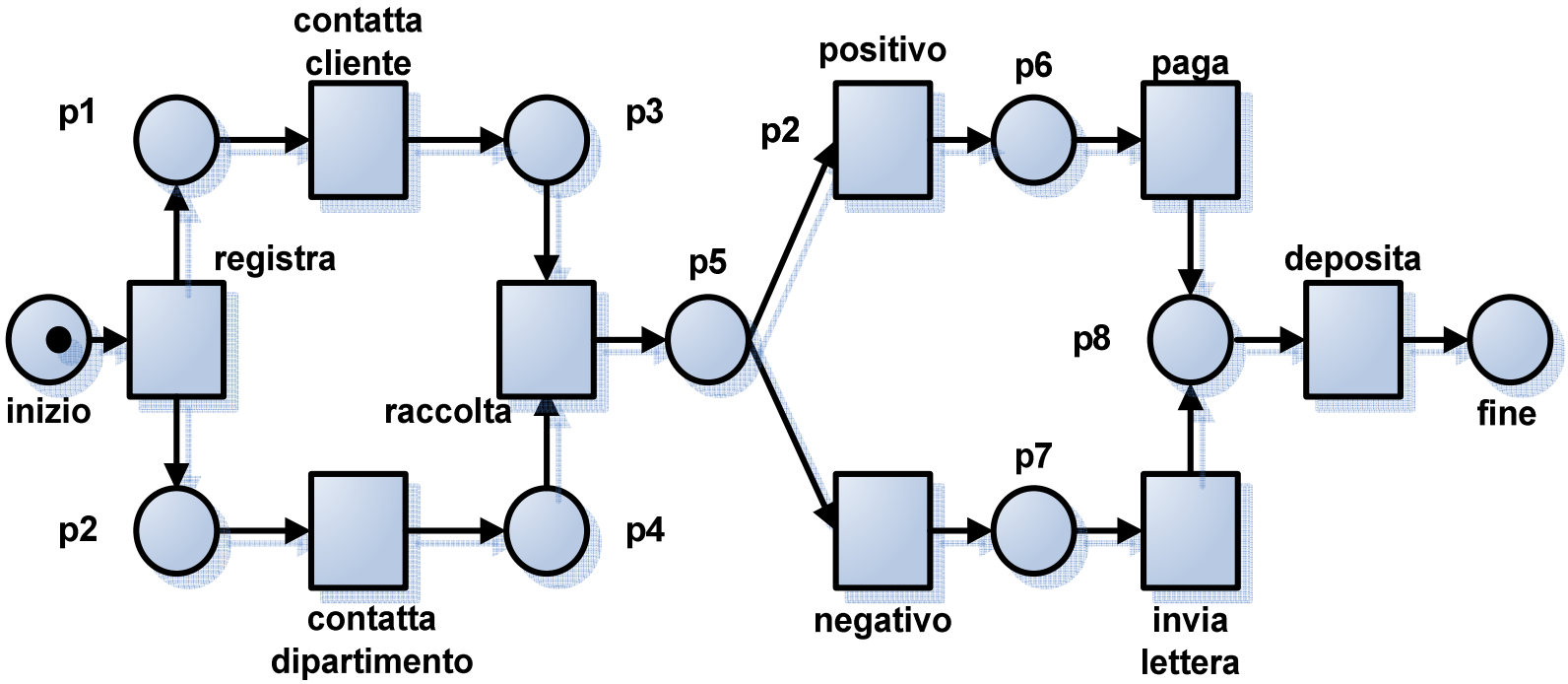
Workflow Nets (WF-Nets)

- Costituiscono una classe di Reti di Petri utilizzata per la descrizione dei ***processi di workflow***.
- Un Processo di Workflow specifica i ***task*** (unità di lavoro atomiche) che devono essere svolti e l'ordine in cui devono essere eseguiti. Ha un unico punto iniziale ed un unico punto finale.

Processo di workflow	**Rete di Petri**
Task	Transizione
Stato del processo	Place
Punto iniziale	Place senza archi entranti
Punto Finale	Place senza archi uscenti

Workflow Nets - Esempio

- WF-net per il processo “gestione reclamo”



Workflow Nets - Routing

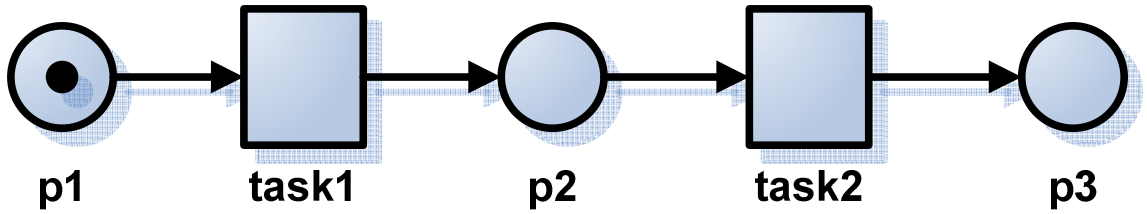
Esistono quattro costrutti di base per definire il cosiddetto *routing* attraverso un processo di workflow:

- ***Routing Sequenziale***
- ***Routing Parallelo***
- ***Routing Selettivo***
- ***Routing Iterativo***

Vediamo come ciascuno di essi viene modellato in una WF net.

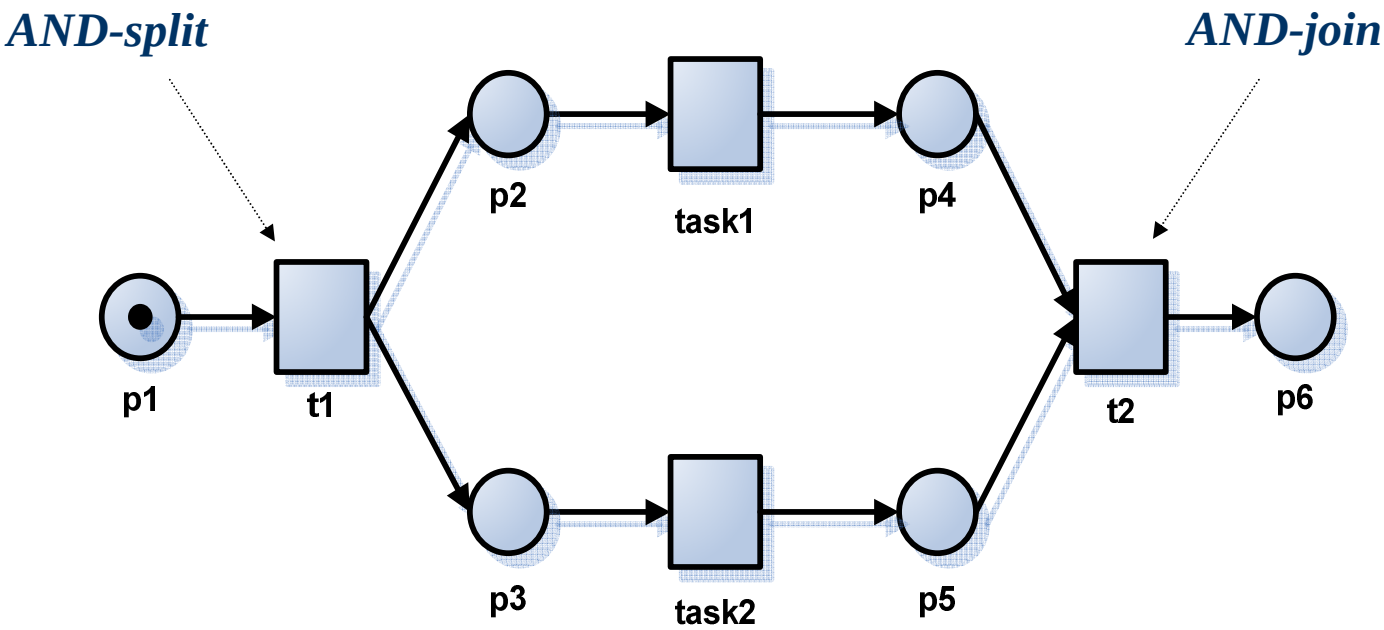
WF-net: Routing Sequenziale

- I task sono eseguiti uno dopo l'altro.



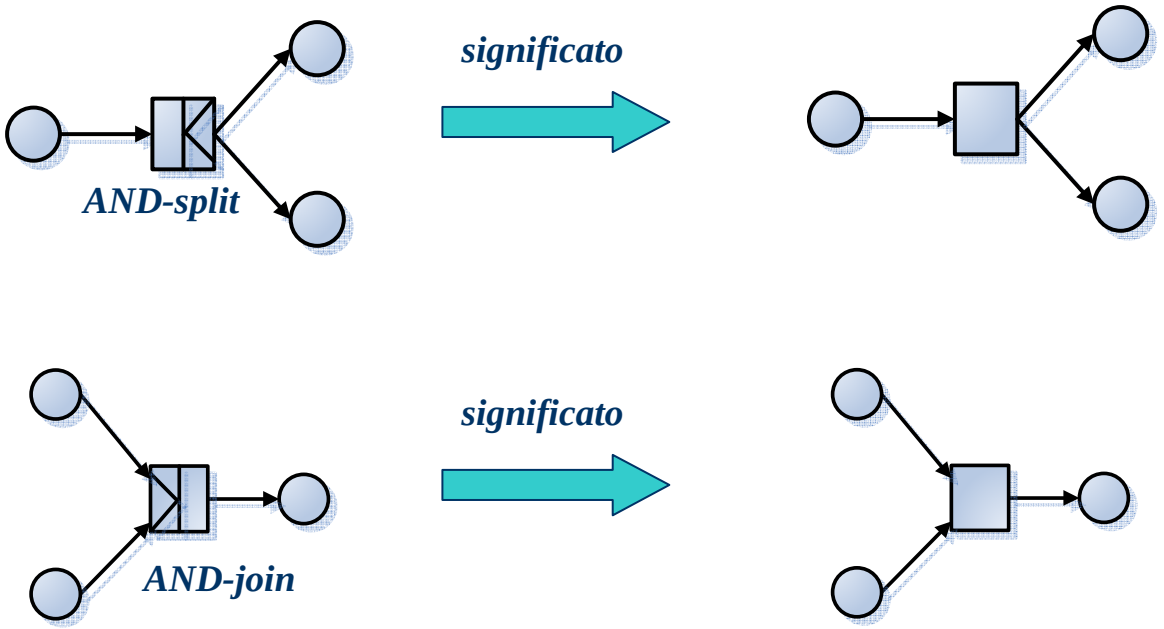
WF-net: Routing Parallelo (1)

- Due o più task possono essere eseguiti in parallelo.



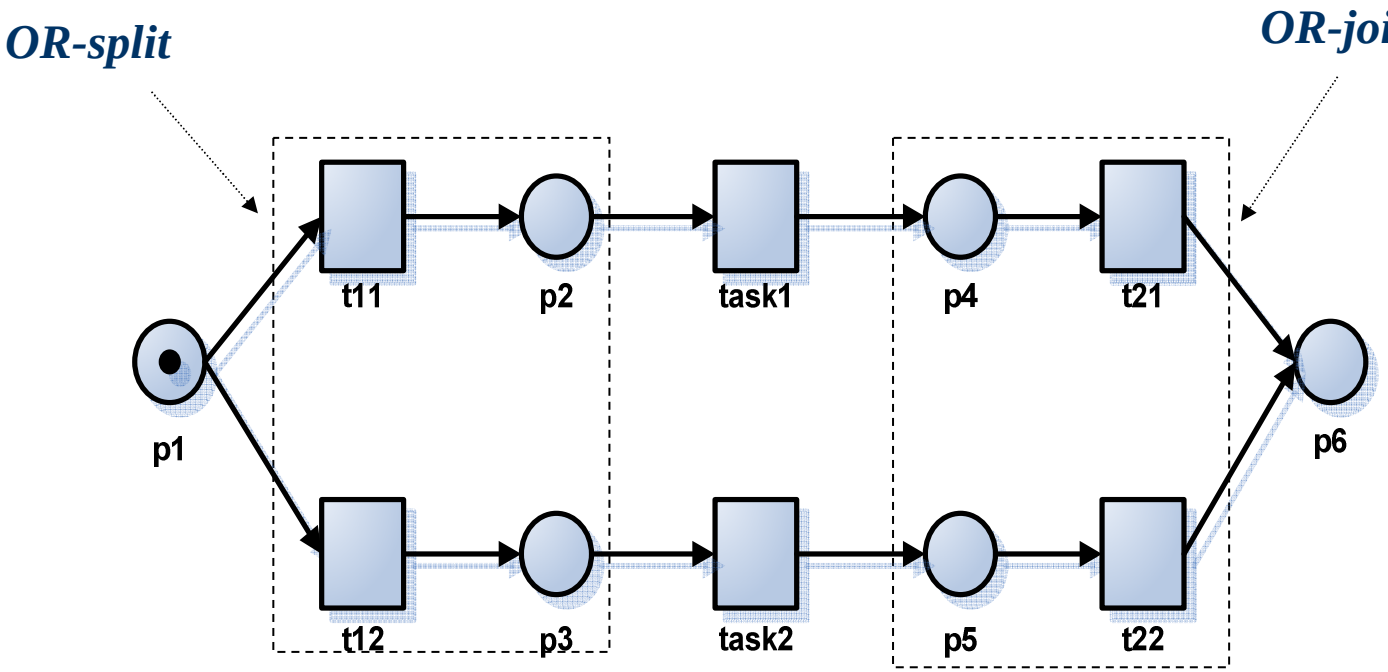
WF-net: Routing Parallelo (2)

- Simboli speciali utilizzati nelle WF-nets:



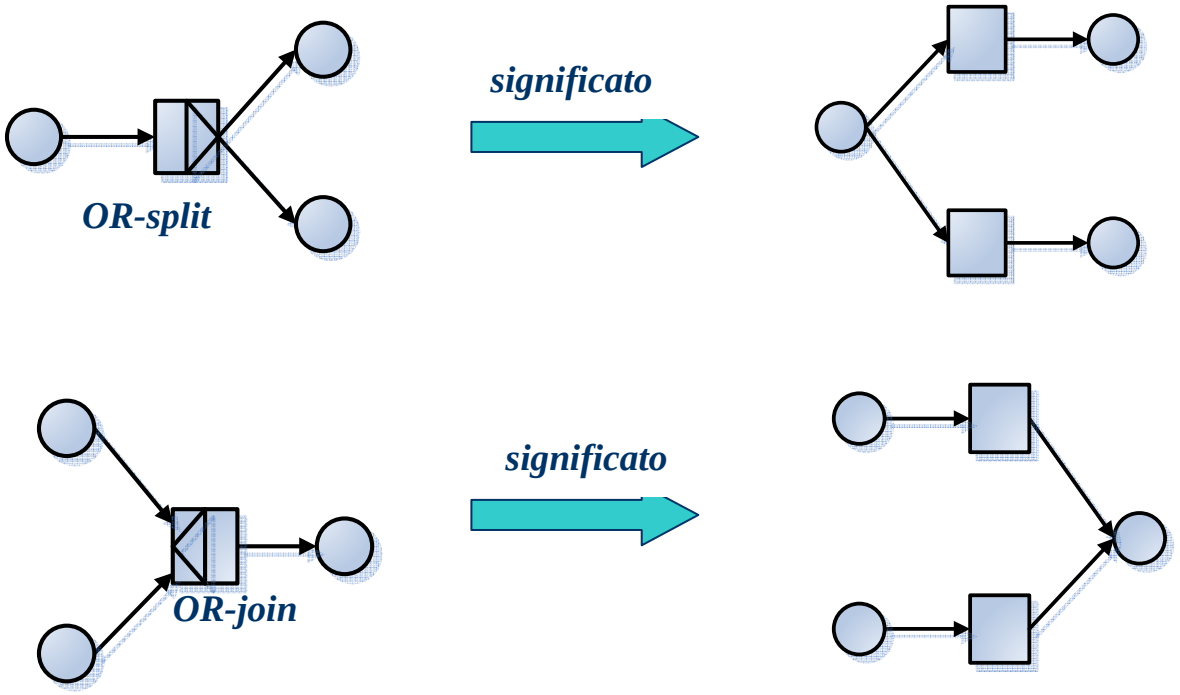
WF-net: Routing Selettivo (1)

- Viene effettuata una scelta tra l'esecuzione di due o più task.



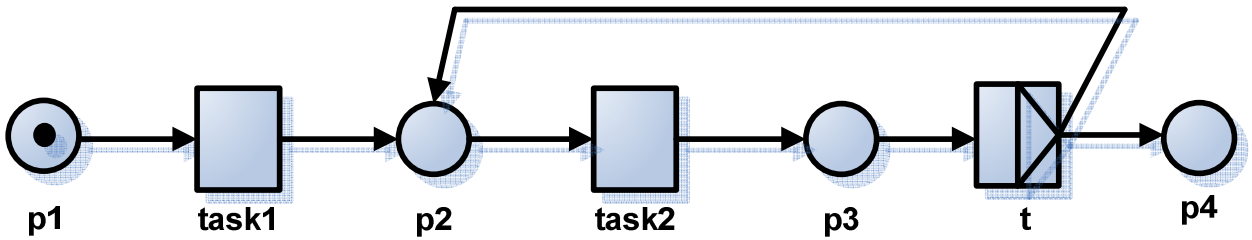
WF-net: Routing Selettivo (2)

- Simboli speciali utilizzati nelle WF-nets:



WF-net: Routing Iterativo

- Ripetizione di uno o più task.



WF-nets: definizione formale

- Una rete di Petri $PN=(P, T, F)$ è una WF-net se e solo se:
  - Esiste un unico place iniziale $inizio \in P$ tale che $\bullet inizio = \emptyset$;
  - Esiste un unico place finale $fine \in P$ tale che $fine\bullet = \emptyset$;
  - Ogni nodo $x \in P \cup T$ si trova in un cammino da *inizio* a *fine*.

WF-nets: Proprietà *Sound* (1)

Una WF-net è *Sound* se e solo se:

1. Per ogni token inserito in *inizio*, uno ed uno solo token prima o poi arriva in *fine*.
2. Quando il token appare in *fine*, tutti gli altri place sono vuoti.
3. Per ogni transizione *t* è possibile partire dallo stato iniziale con un solo token in *inizio* ed arrivare ad uno stato in cui *t* scatta.

WF-nets: Proprietà *Sound* (2)

Sia *i* lo stato iniziale con un solo token nel place *inizio* e sia *f* lo stato finale con un solo token nel place *fine*. Una WF-net è *Sound* se e solo se:

- Per ogni stato *M* raggiungibile dallo stato *i*, esiste una sequenza di scatti che porta dallo stato *M* allo stato *f*:

$$\forall M \quad (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} f)$$

- Lo stato *f* è l'unico stato con un token nel place *fine* raggiungibile dallo stato *i:*

$$\forall M \quad (i \xrightarrow{*} M \quad \wedge \quad M \geq f) \Rightarrow (M = f)$$

- Non ci sono "dead transitions" in (PN, i):

$$\forall t \in T \quad \exists M, M' \quad (i \xrightarrow{*} M \xrightarrow{t} M')$$

WF-nets: Verifica in Spin

- Tradurre una WF-net in un modello Promela.
- Tradurre le proprietà in formule LTL.
- Utilizzare Spin per valutare le formule LTL sul modello Promela.

WF-Nets: Traduzione in Promela (1)

WF-Net	Promela
Place	Variabile intera il cui valore indica il numero di token presenti
Transizione	Processo nel cui corpo vengono specificate le condizioni per far “scattare” la transizione ed il codice per consumare e produrre i token

Per ogni transizione definiamo inoltre una variabile booleana *transizione_eseguita* inizializzata a *false* che verrà posta a *true* quando la transizione scatta.

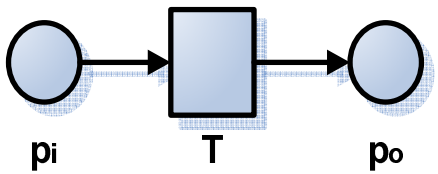
WF-Nets: Traduzione in Promela (2)

```
proctype transizione()
 {
    do :: atomic
 {
    TEST_ABILITAZIONE →
    CONSUMA_TOKEN;PRODUCI_TOKEN;transizione_eseguita=
                                                    true;
 }
   od;  }
```

Il codice relativo a TEST_ABILITAZIONE, CONSUMA_TOKEN e PRODUCI_TOKEN dipenderà dal numero di input place e di output place della transizione e dal tipo di transizione (transizione semplice, AND-split, OR-split, AND-join, OR-join).

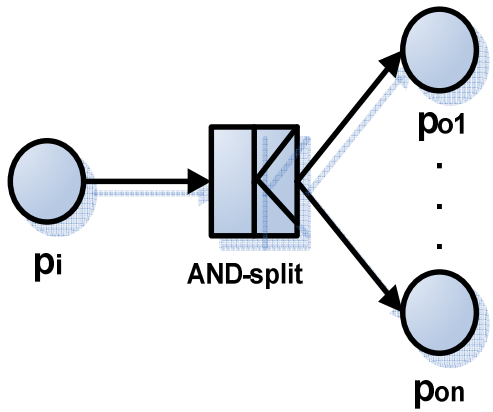
Transizione Semplice

WF-Net	Promela

	proctype T ()
	{
	do :: atomic
	{
	(pi > 0) → pi= pi -1; po= po+1;
	T_eseguita=true;
	}
	od
	}

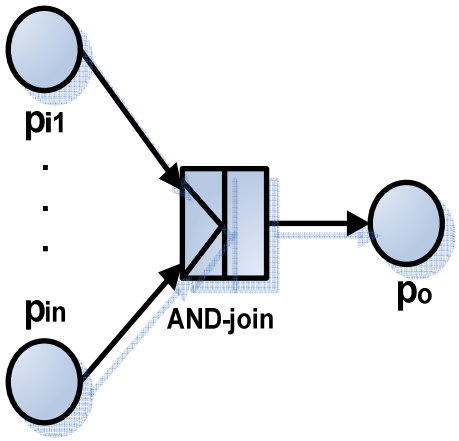
AND-Split

WF-Net	Promela

	proctype AND-split ()
	{
	do :: atomic
	{
	(pi > 0) → pi= pi -1; po1= po1+1; … ;
	pon =pon+1; AND-split_eseguita = true;
	}
	od
	}

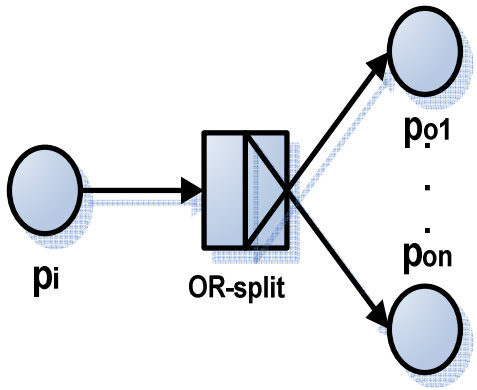
AND-Join

WF-Net	Promela

	proctype AND-join ()
	{
	do :: atomic
	{
	((pi1 > 0) && … && (pin > 0)) →
	pi1= pi1 -1; …; pin= pin -1; po= po +1;
	AND-join_eseguita= true;
	}
	od
	}

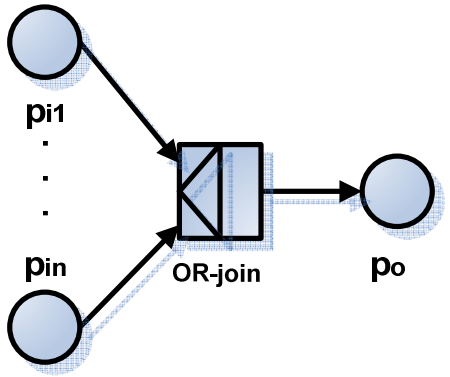
OR-Split

WF-Net	Promela

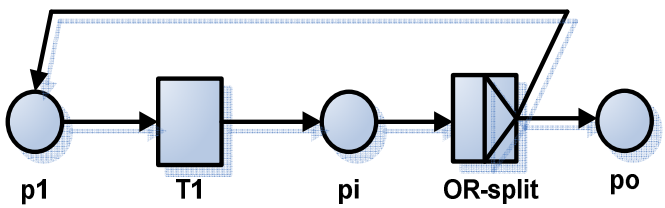
	proctype OR-split () {
	 do :: atomic
	 {
	 (pi > 0) →
	 if
	 :: true→ pi= pi -1; po1=po1+1; OR-split_eseguita =true;
	 :: true → pi= pi -1;po2=po2+1; OR-split_eseguita = true;
	 .
	 .
	 .
	 :: true → pi=pi -1;pon=pon+1; OR-split_eseguita = true;
	 fi
	 }
	 od }

OR-Join

WF-Net	Promela

	proctype OR-join () {
	do :: atomic{
	((pi1 > 0) || (pi2 > 0) || … || (pin > 0)) →
	if
	:: (pi1>0)→pi1=pi1-1;po=po +1;OR-join_eseguita = true;
	
	:: (pi2 > 0)→pi2=pi2-1;po=po +1;OR-join_eseguita =true;
	.
	.
	.
	:: (pin > 0)→pin=pin-1;po=po +1;OR-join_eseguita =true;
	
	fi }
	od }

Loop

WF-Net	Promela
	**proctype OR-split ()** **{** **do :: atomic** **{** **(pi > 0) →** **if** **:: true→ pi= pi -1; po= po+1; execute_OR-split=true;** **:: true → pi= pi -1; p1= p1+1; execute_OR-split = true;** **fi** **}** **od** **}**

Schema di Traduzione

```
# define Place int
bool transizione1_eseguita =false;
bool transizionen_eseguita=false;

Place inizio,p1,p2,…,pn, fine;

proctype transizione1() { }
proctype transizionen() { }

init{atomic{
inizio=1;
p1=0;
pn=0;
fine=0;
run transizione1();
run transizionen(); }}
```

Sound: traduzione in LTL

Una WF-net è *Sound* se e solo se:

1. Per ogni token inserito in *inizio*, uno ed uno solo token prima o poi arriva in *fine*.
2. Quando il token appare in *fine*, tutti gli altri place sono vuoti.
3. Per ogni transizione *t* è possibile partire dallo stato iniziale con un solo token in *inizio* ed arrivare ad uno stato in cui *t* scatta.

Introduciamo i seguenti predicati:

marcato(p): p contiene un token.

abilitata(t): tutti gli input place di *t* contengono un token.

eseguita(t): t è “scattata”.

Sound: traduzione in LTL (2)

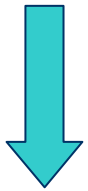
- **Requisiti 1 e 2: VERIFICARE LA VALIDITA' DI:**

F (marcato(fine)Λ¬marcato(inzio) Λ¬marcato(p1)Λ … Λ ¬marcato(pn))

F (fine==1 && inizio==0 && p1==0 && p2==0 && … && pn==0)

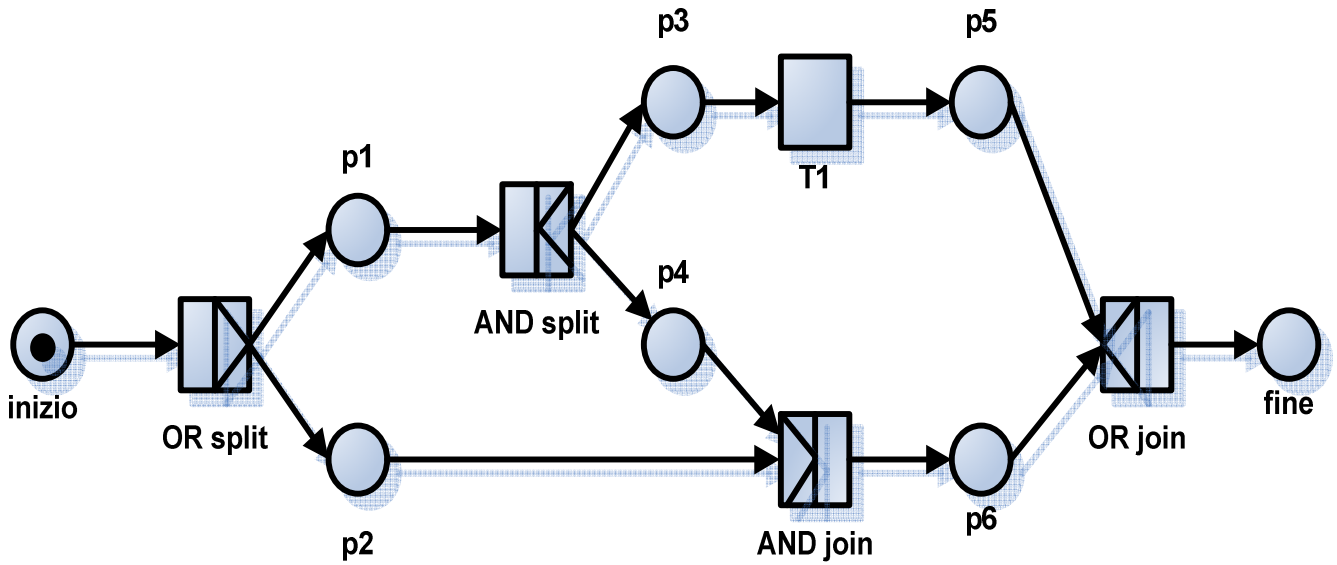
- **Requisito 3: VERIFICARE LA NON VALIDITA' DI:**

G(¬eseguita(t))



G (¬ (t_eseguita = =1))

Esempio


p3
p5
p1
T1
p4
AND split
inizio
OR split
fine
OR join
p2
AND join
p6

Diagrammi delle Attività UML

- Particolarmente adatti alla descrizione dei processi di workflow: supportano l'esecuzione parallela, condizionale ed iterativa.

- La verifica formale dei Diagrammi delle Attività prevede i seguenti passi:

  - Trasformare il Diagramma delle Attività in una WF-net equivalente.
  - Verificare la proprietà *Sound* sulla WF-net equivalente.
  - Stabilire se il Diagramma delle Attività è corretto.

Diagrammi delle Attività UML

Nella nostra trattazione prenderemo in considerazione i seguenti elementi:

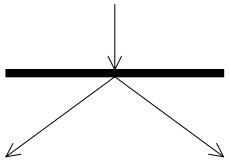
- **Nodo Attività**

- **Nodo Iniziale**

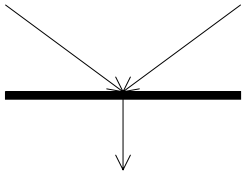
- **Nodo Finale**

Diagrammi delle Attività UML

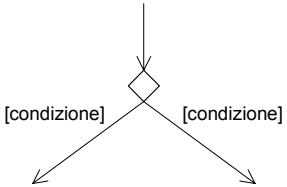
- **Nodo Fork**



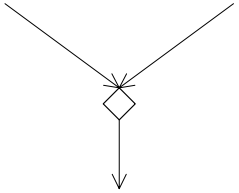
- **Nodo Join**

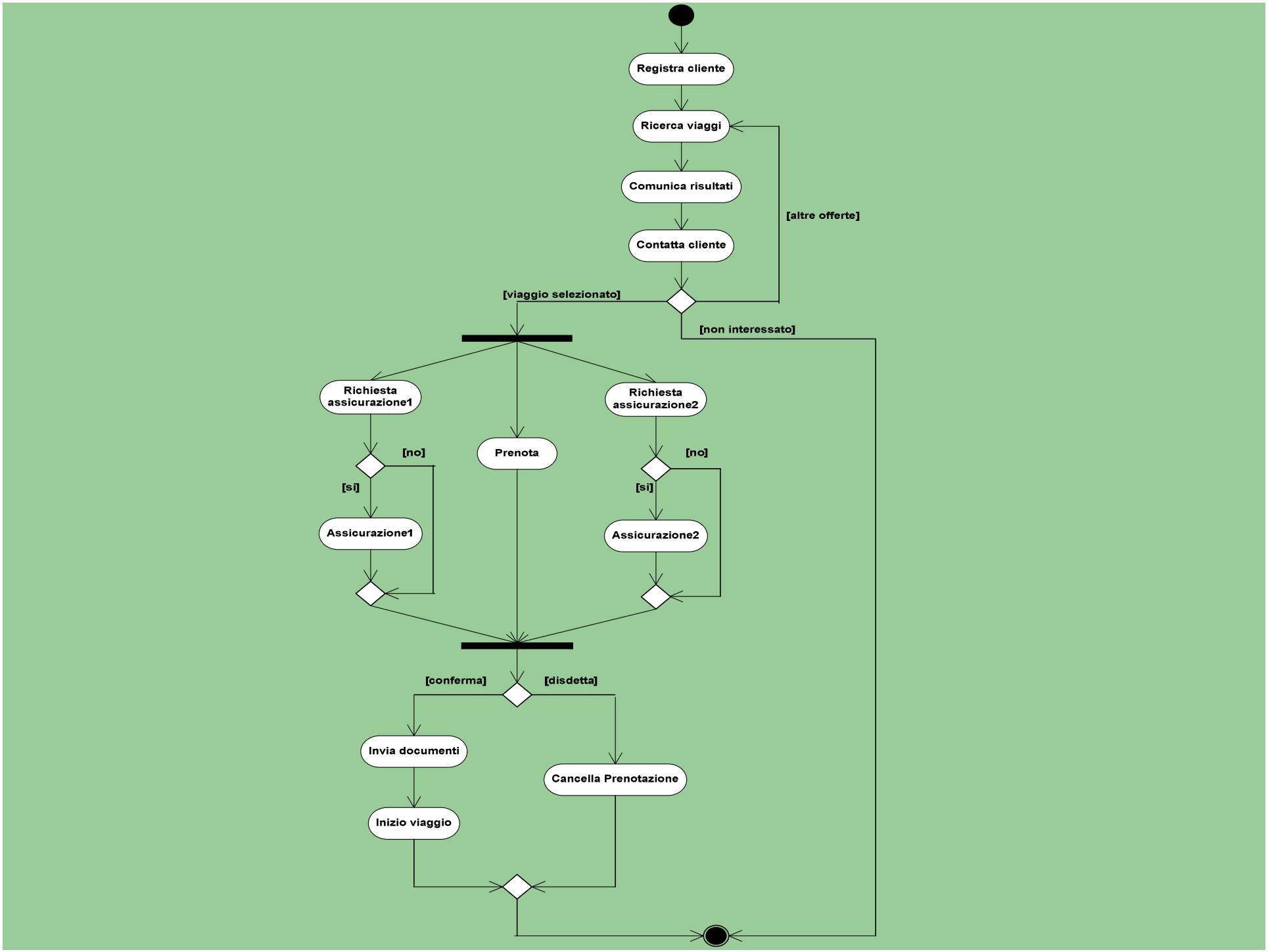


- **Nodo Decisione**



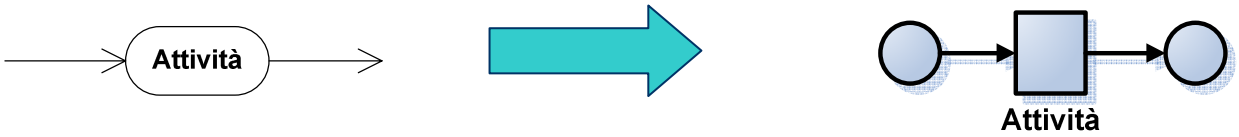
- **Nodo Merge**




Registra cliente
Ricerca viaggi
Comunica risultati
[altre offerte]
Contatta cliente
[viaggio selezionato]
[non interessato]
Richiesta assicurazione1
Prenota
Richiesta assicurazione2
[no]
[si]
[no]
[si]
Assicurazione1
Assicurazione2
[conferma]
[disdetta]
Invia documenti
Cancella Prenotazione
Inizio viaggio

Trasformazione Diagramma delle Attività – WF-net

- **Nodo Attività**

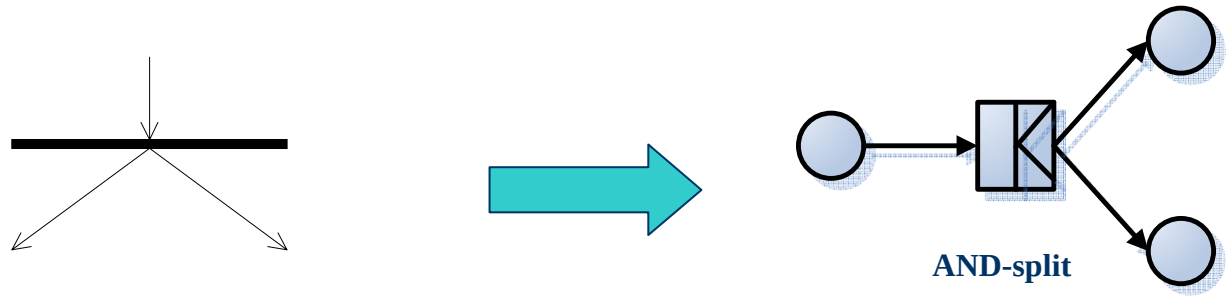


- **Nodo Iniziale**

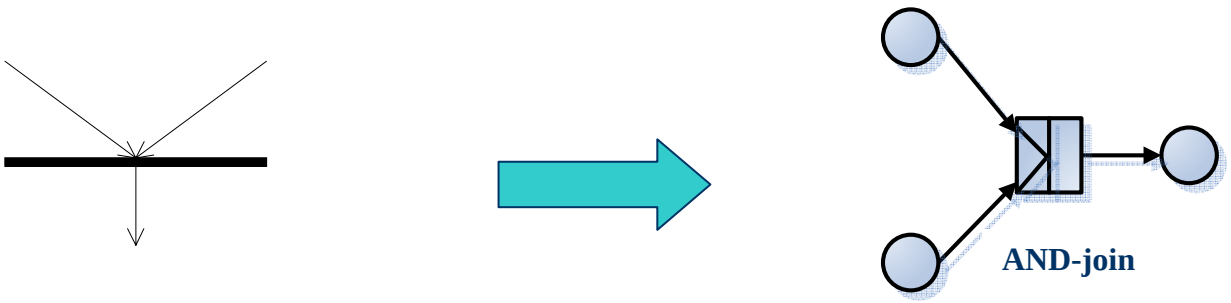
- **Nodo Finale**

Trasformazione Diagramma delle Attività – WF-net

- **Nodo Fork**

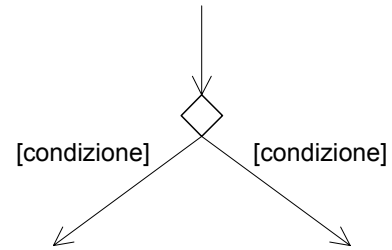


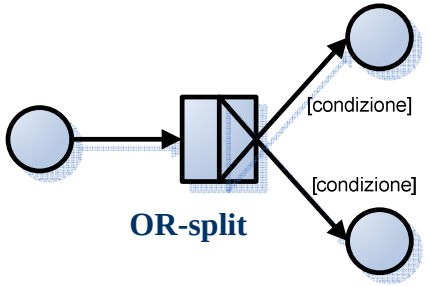
- **Nodo Join**



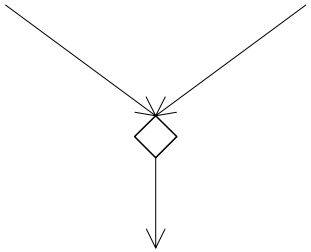
Trasformazione Diagramma delle Attività – WF-net

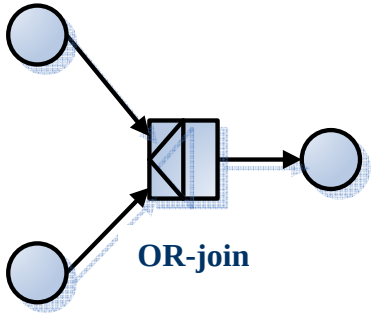
- **Node Decisione**

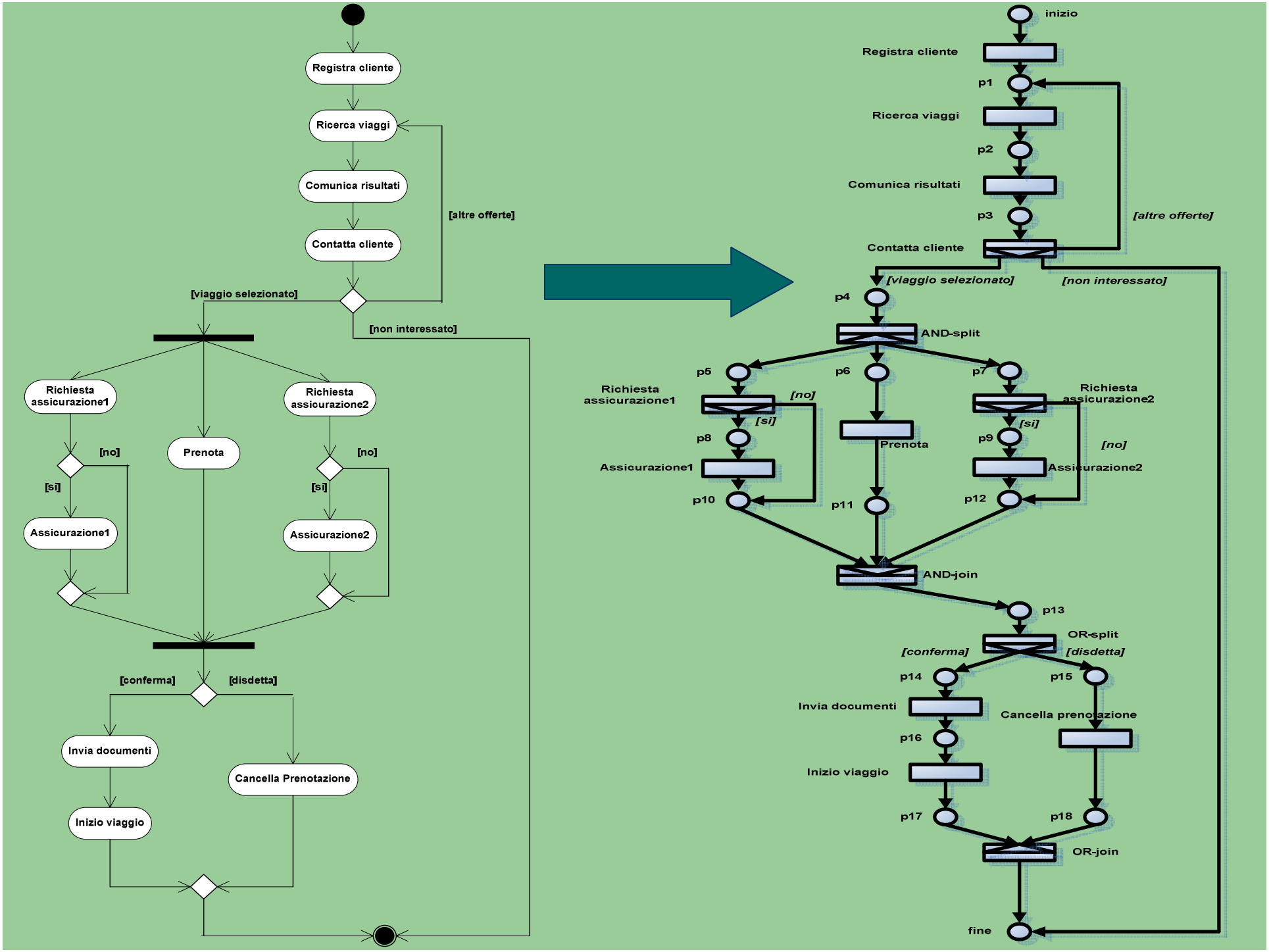




- **Nodo Merge**






Registra cliente
Ricerca viaggi
Comunica risultati
Contatta cliente
[altre offerte]
[viaggio selezionato]
[non interessato]
Richiesta assicurazione1
Prenota
Richiesta assicurazione2
[no]
[si]
[no]
[si]
Assicurazione1
Assicurazione2
[conferma]
[disdetta]
Invia documenti
Cancella Prenotazione
Inizio viaggio
inizio
Registra cliente
p1
Ricerca viaggi
p2
Comunica risultati
p3
[altre offerte]
Contatta cliente
[viaggio selezionato]
[non interessato]
p4
AND-split
p5
p6
p7
Richiesta assicurazione1
[no]
[si]
Prenota
[si]
Richiesta assicurazione2
[no]
p8
p9
Assicurazione1
Assicurazione2
p10
p11
p12
AND-join
p13
OR-split
[conferma]
[disdetta]
p14
p15
Invia documenti
Cancella prenotazione
p16
Inizio viaggio
p17
p18
OR-join
fine

Verifica in Spin

define terminazione (fine==1 && inizio==0 && p1==0 && p2==0 && p3==0 && p4==0 && p5==0 && p6==0 && p7==0 && p8==0 && p9==0 && p10==0 && p11==0 && p12==0 && p13==0 && p14==0 && p15==0 && p16==0 && p17==0 && p18==0)

F terminazione

(Valida)

WF-net Sound → **Diagramma Delle Attività corretto**

Verifica di altre proprietà

Possiamo verificare anche proprietà specifiche del processo modellato dal diagramma. Ad esempio:

(1) Un cliente può partire per un viaggio senza aver stipulato alcuna assicurazione.

(2) Un viaggio prenotato può essere cancellato.

(3) Se un viaggio viene cancellato, i documenti non vengono inviati.

(4) Una volta inviati i documenti, un viaggio non può più essere cancellato.

Verifica di altre proprietà (2)

(1) Un cliente può partire per un viaggio senza aver stipulato alcuna assicurazione.

#define assicurazione1 (ASSICURAZIONE1_eseguita==true)
#define assicurazione2 (ASSICURAZIONE2_eseguita==true)
#define inizioviaggio (INIZIOVIAGGIO_eseguita==true)

G ((!assicurazione1 && !assicurazione2) → (!inizioviaggio))

(Non Valida)

Verifica di altre proprietà (3)

(2) Un viaggio prenotato può essere cancellato.

#define prenota (PRENOTA_eseguita==true)

#define cancellaprenotazione(CANCELLAPRENOTAZIONE_eseguita==true)

G (prenota → !cancellaprenotazione)

(Non Valida)

Verifica di altre proprietà (4)

(3) Se un viaggio viene cancellato, i documenti non vengono inviati.

#define inviadocumenti (INVIADOCUMENTI_eseguita==true)

#define cancellaprenotazione (CANCELLAPRENOTAZIONE_eseguita==true)

G (cancellaprenotazione → !inviadocumenti)

(Valida)

(4) Una volta inviati i documenti, un viaggio non può più essere cancellato.

G (inviadocumenti → !cancellaprenotazione)

(Valida)

Riferimenti

[1] Will van der Aalst, Kees Max van Hee. *Workflow Management: Models, Methods, and Systems*.

[2] Theo C. Ruys. *SPIN Beginners' Tutorial*.

[3] Gerard Holzmann, Theo C. Ruys. *Advanced SPIN Tutorial*.

[4] *http://spinroot.com*