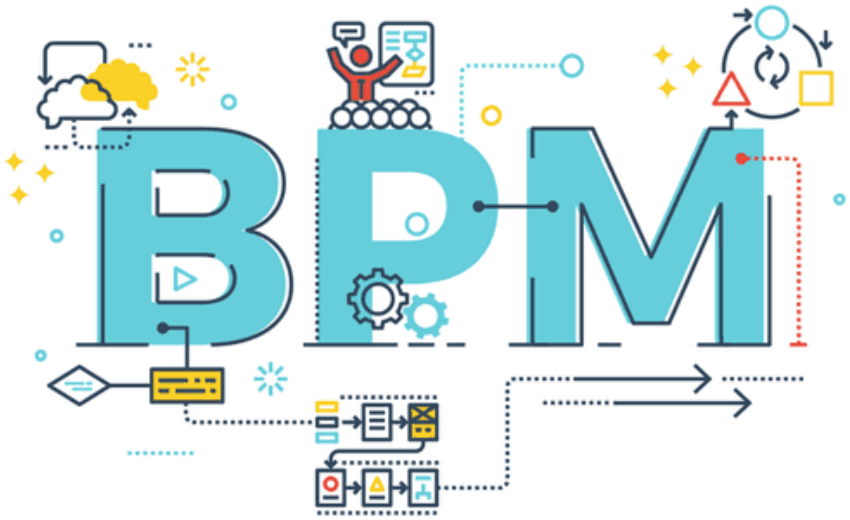


Business Process Modeling

Corso di Laurea Magistrale in Data Science and Business Informatics

Dipartimento di Informatica

Università di Pisa

Calendar management

Studente

Tommaso Cavalieri

[597707]

Anno accademico 2019/2020

Indice

1 Il modello **2**
1.1 Variante 3

2 Petri Nets **6**
2.1 Utente 7
2.2 Applicazione 10
2.3 Worflow system 13

Elenco delle figure

1.1 Collaboration diagram 4
1.2 Collaboration diagram - variante 5

2.1 Da BPMN a Petri nets 6
2.2 Petri net rappresentanti il processo dell'utente 8
2.3 *Semantical analysis* delle reti dell'utente 8
2.4 *Reachability graph* delle Petri net dell'utente 9
2.5 Petri net rappresentanti il processo dell'applicazione 11
2.6 *Semantical analysis* delle reti dell'applicazione 11
2.7 *Reachability graph* delle Petri net dell'applicazione 12
2.8 Petri net dell'intero *worklow system* 14
2.9 *Semantical analysis* del *worflow system* 14

Introduzione

Lo scenario considerato in questo studio è quello di un'applicazione per la gestione di calendari che prevede la modellizzazione di un processo per l'utente e uno per l'applicazione. Nello specifico, l'utente avvia l'applicazione e può decidere se caricare un calendario già esistente o crearne uno nuovo, in seguito può decidere tramite un menu di scelta se crearne uno nuovo appuntamento, cercarne uno già presente nel calendario aperto, oppure chiudere l'applicazione. Dopo un'eventuale ricerca è possibile procedere con la rimozione o modifica dell'appuntamento selezionato previa conferma dell'utente, che è richiesta anche nel caso in cui si scelga di chiudere dell'applicazione. I dati vengono salvati nel calendario in automatico dopo ogni modifica. I processi progettati rispecchiano fedelmente lo scenario appena descritto e sono tra di loro compatibili. È stata poi ripetuta l'analisi sui medesimi processi modificati in modo tale da dare la possibilità all'utente di lavorare su un altro calendario prima di procedere con la chiusura dell'applicazione.

Capitolo 1

Il modello

Per la rappresentazione grafica dei processi descritti nell'Introduzione è stata utilizzata la Business Process Model and Notation (BPMN); tale rappresentazione è stata realizzata tramite l'applicazione Camunda Modeler. Come si può osservare nel *collaboration diagram* in Figura 1.1, sono stati creati due processi, uno per l'utente e l'altro per l'applicazione, entrambi contenuti nelle rispettive *pool*. Il processo comincia quando l'utente avvia l'applicazione, così facendo invia un messaggio al processo dell'applicazione che comincia a sua volta grazie ad un *message start event*. La prima attività del processo dell'app è una *user task* che corrisponde ad una scelta dell'utente, rappresentata tra due *exclusive gateway* nel suo processo: egli può decidere se caricare uno dei calendari che si trovano già all'interno di un apposito database oppure se crearne uno nuovo. Una volta aperto il calendario desiderato, nuovo o pre-esistente che sia, e rappresentato nella notazione utilizzata da un *data object*, si apre un menu che permette all'utente di aggiungere un nuovo evento, cercarne uno già presente sul calendario o chiudere l'applicazione; tale scelta viene rappresentata tramite un *exclusive gateway* nel processo dell'utente. Nel processo dell'applicazione è invece presente, dopo l'attività **Open Menu**, un *event based gateway*, che fa sì che venga attivato solamente il ramo corrispondente alla scelta fatta dall'utente; la comunicazione tra i due processi avviene grazie a dei un *message flow* tra *throw and catch intermediate message events* (in certi casi viene utilizzata direttamente una *send task* per inviare il messaggio). Se l'utente decide di chiudere l'app, il corrispettivo ramo viene percorso nel processo dell'applicazione, la quale, tramite una *user task*, richiede all'utente se desidera o meno confermare tale operazione. A questo punto un *message flow* ci riporta nel processo del-

l'utente che tramite un *exclusive gateway* decide se confermare la chiusura dell'applicazione, che porterebbe alla terminazione di entrambi i processi (dopo aver salvato il calendario aperto nell'apposito database), o se annullarla tornando così al menu precedente tramite un apposito *exclusive gateway*. Si noti che in caso di chiusura dell'app il processo dell'utente termina con un *message end event*, che manda un messaggio al processo dell'applicazione, il quale salva il calendario aperto nel database e termina a sua volta. Nel caso invece in cui l'utente decida di aggiungere un nuovo evento al calendario, esso viene aggiornato e, sempre tramite l'apposito *exclusive gateway*, viene riproposto il menu, cosicché l'utente possa decidere se aggiungere un altro evento, cercarne uno o chiudere l'applicazione. Qualora l'utente decidesse di cercare un evento, ciò viene fatto all'interno del calendario aperto, che nell'attività **Search Event** viene utilizzato come input; una volta selezionato l'evento desiderato, l'utente può decidere se rimuoverlo o modificarlo (scelta rappresentata tra due *exclusive gateway*). In base alla scelta effettuata viene percorso nel processo dell'applicazione il ramo corrispondente che, come nel caso della chiusura dell'app, tramite una *user task* richiede all'utente se desidera confermare l'operazione o meno. Anche in questo caso un *message flow* ci riporta nel processo dell'utente, che tramite un *exclusive gateway* può decidere se confermare o annullare l'azione. In entrambi i casi i processi, tramite gli appositi *exclusive gateway*, tornano al menu iniziale, ma se l'azione viene confermata il calendario viene aggiornato con la modifica o rimozione dell'evento selezionato, altrimenti rimane invariato. Si noti che sia nel caso di chiusura dell'app che in quello di modifica/rimozione di un evento, la task del processo dell'applicazione **Ask Confirmation** è seguita da un *event based gateway*, che permette al processo dell'applicazione di percorrere l'uno o l'altro ramo in base alla scelta dell'utente di confermare o meno l'azione.

1.1 Variante

In questo studio è stata presa in considerazione anche una variante del modello appena descritto, nella quale viene data all'utente la possibilità di lavorare su un altro calendario prima di chiudere l'applicazione. Nella rappresentazione tramite BPMN ciò viene ottenuto aggiungendo un ramo in uscita dall'*exclusive gateway* che segue l'evento **Confirmation asked** nel processo dell'utente e uno in uscita dall'*event based gateway* che segue la task **Ask closing confirmation** nel processo dell'applicazione. Tali rami riconducono

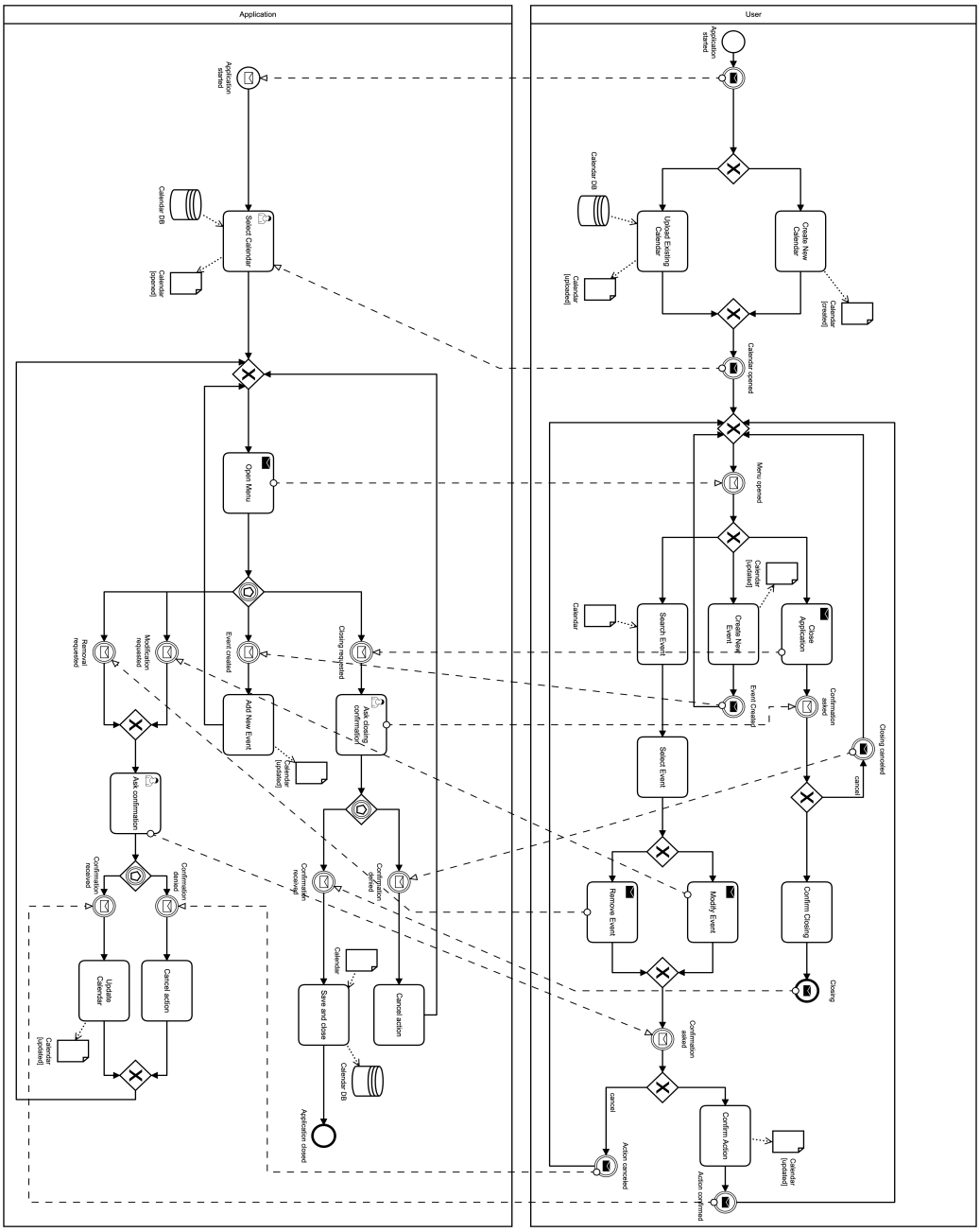


Figura 1.1: Collaboration diagram

a degli *exclusive gateway* appositamente inseriti per ricondurre i processi al momento della scelta iniziale dell'utente, ovvero se creare un nuovo calendario o caricarne uno già esistente. Si noti che, come nel caso di chiusura

dell'app, nel processo dell'applicaione è previsto il salvataggio nel calendario aperto nell'opportuno database prima di far scegliere all'utente il nuovo calendario da aprire o creare. È possibile osservare il *collaboration diagram* della variante appena descritta nella Figura 1.2.

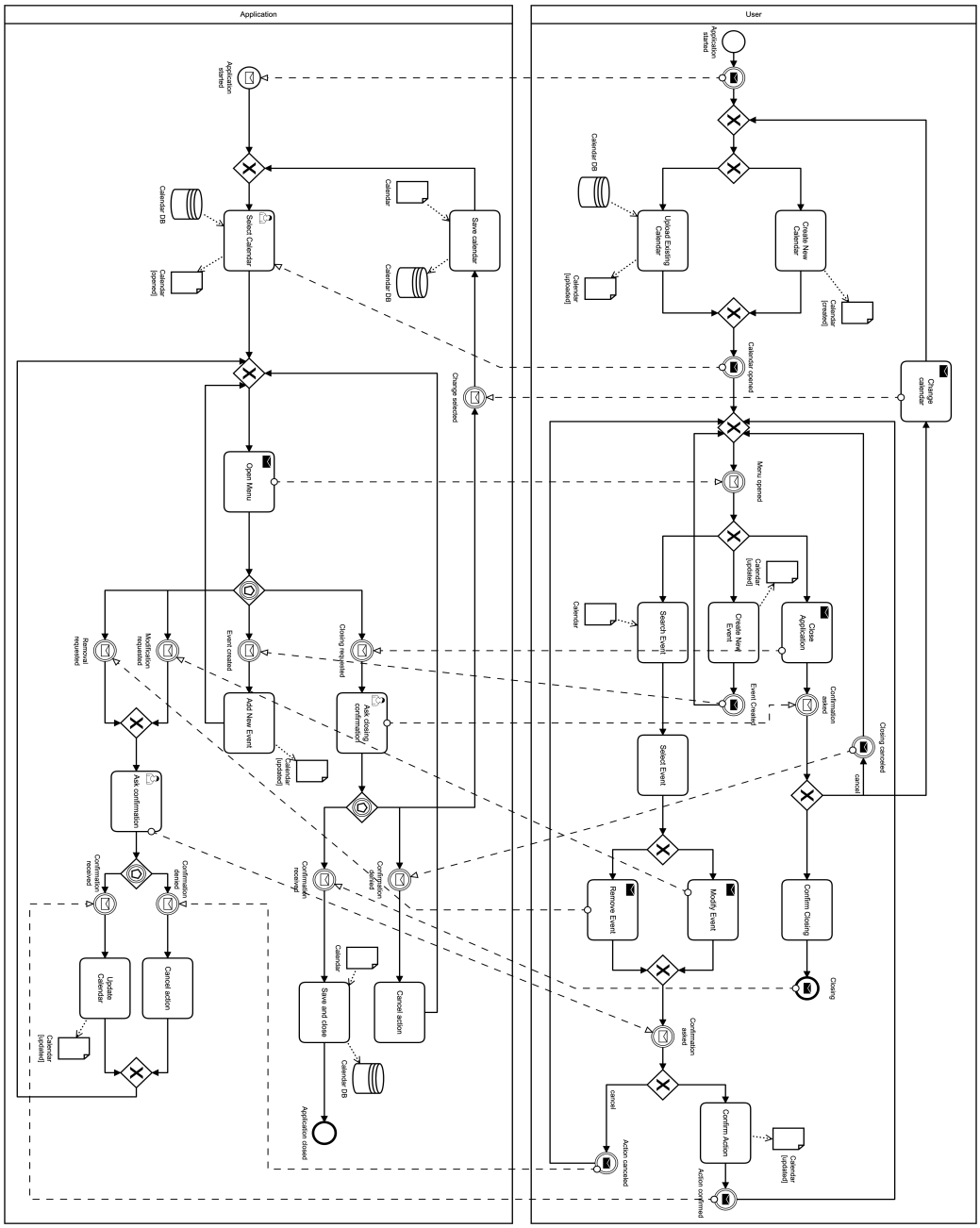


Figura 1.2: Collaboration diagram - variante

Capitolo 2

Petri Nets

Le reti progettate tramite modelli più astratti come la BPMN sono state poi trasformate, tramite tecniche apposite, in Petri nets, per verificarne la correttezza ed analizzarne le caratteristiche. Il metodo più semplice prevede, in linea di massima, la creazione di una piazza per ogni arco, una transizione per ogni evento e una transizione per ogni attività, i gateway sono stati trasformati in **XOR-join** se presentavano multipli archi in entrata e uno solo in uscita ed in **XOR-split** nel caso opposto; ci sono però delle eccezioni, come per esempio per gli *start* ed *end event*: è stato infatti necessario aggiungere, sia per la rete dell'utente che per quella dell'applicazione, una piazza iniziale, priva di *incoming flow* e segnata con un token, ed una piazza finale, priva di *outgoing flow*. Uno schema generale dei metodi di trasformazione da BPMN a Petri net utilizzati è consultabile in Figura 2.1. Il programma utilizzato per costruire le reti è stato WoPeD.

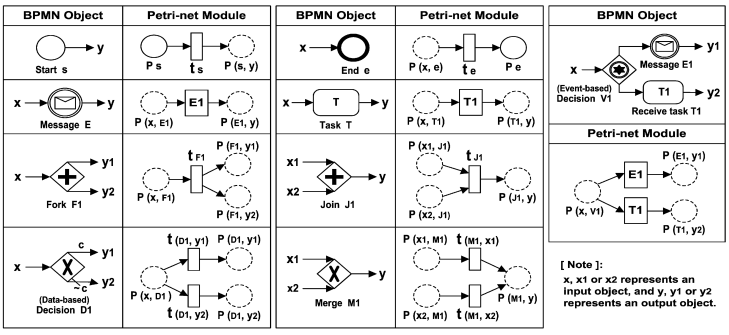


Figura 2.1: Da BPMN a Petri nets

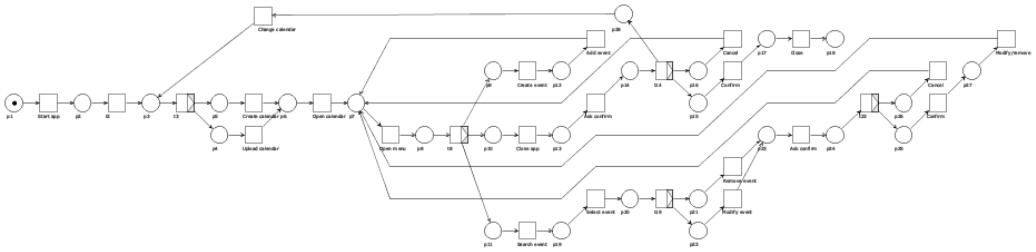
2.1 Utente

La Petri net rappresentate il processo dell'utente è presentata in Figura 2.2 (a) nella sua forma base e nella Figura 2.2 (b) per quanto riguarda la variante; di seguito sono elencate alcune delle loro caratteristiche:

- la rete dell'utente è composta da 27 piazze e 32 transizioni per un totale di 64 archi nel modello base, una piazza, due transizioni e quattro archi vengono aggiunti nella variante;
- le reti, sia nel modello base che nella variante, sono **workflow net** (N) e sono **safe** e **sound**;
- sono entrambe **S-system**, in quanto il pre-set e post-set di ogni transizione contiene esattamente una piazza, questo ci permette anche di trovare facilmente una *S-invariant* per le reti, che sarà dunque della forma **I**=[k k ... k];
- nessuna delle due è un **T-system** nemmeno con l'aggiunta della funzione *reset* (N*), in quanto diverse piazze hanno più di una transizione nel loro post-set o pre-set;
- sono entrambe **free-choice** in quanto per ogni coppia di transizioni i loro pre-set sono identici o disgiunti;
- sono entrambe **well-structured** poiché non presentano né *TP-handles* né *PT-handles*;
- N è **bounded** e **deadlock-free** sia nel modello base che nella variante e N* è **live** in entrambi i modelli;
- sono entrambe **S-coverable**, nello specifico da una *S-component* composta da tutte le piazze delle rispettive reti;
- sono entrambe **connected** (**strongly** se si aggiunge la funzione *reset*).

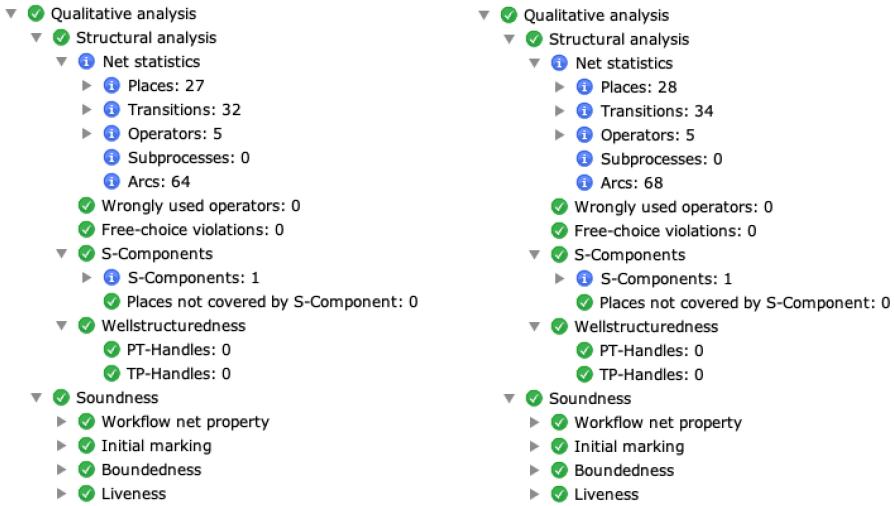
Siccome entrambe le reti sono *bounded*, il loro ***reachability graph*** è finito e coincide con il *coverability graph*, che può essere facilmente ottenuto tramite l'apposita funzione su WoPeD. Nella figura 2.4 vengono presentati i *reachability graph* del modello base e della variante, il primo contenente 27 vertici e 32 archi, nel secondo sono presenti un arco e un vertice in più.

(a) Modello base



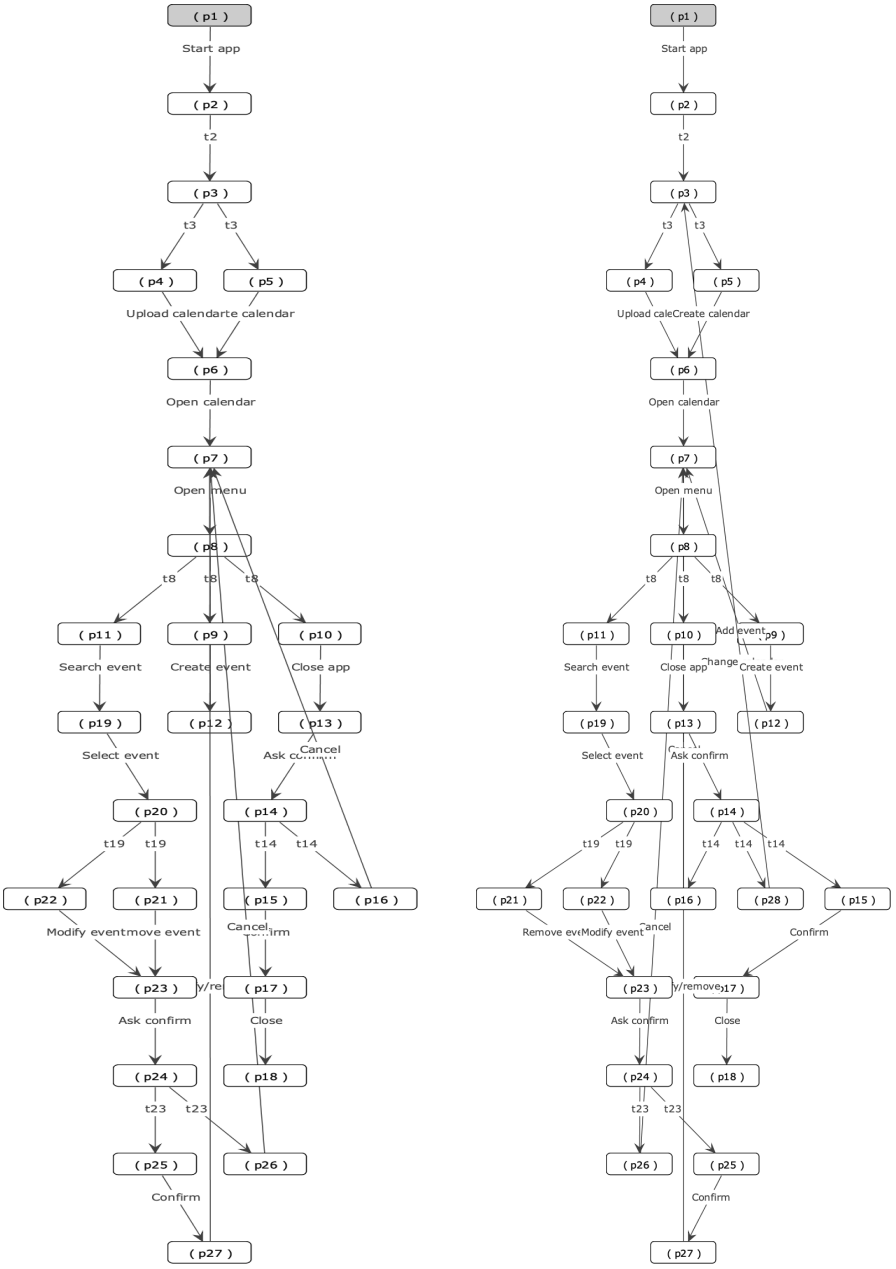
(b) Variante

Figura 2.2: Petri net rappresentanti il processo dell'utente



(a) Modello base (b) Variante

Figura 2.3: *Semantical analysis* delle reti dell'utente



(a) Modello base (b) Variante

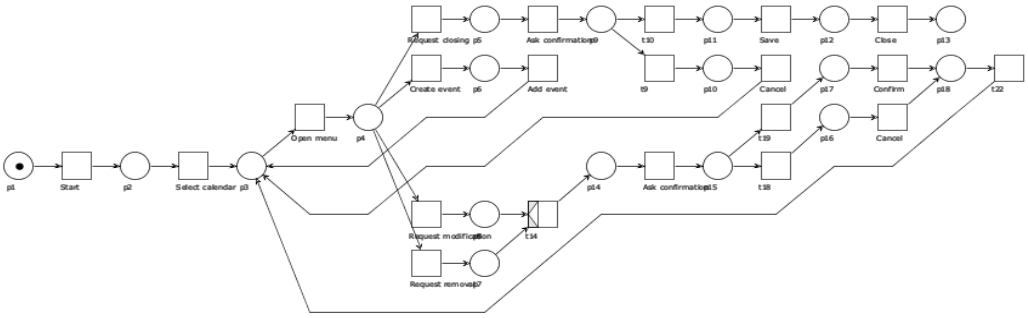
Figura 2.4: *Reachability graph* delle Petri net dell'utente

2.2 Applicazione

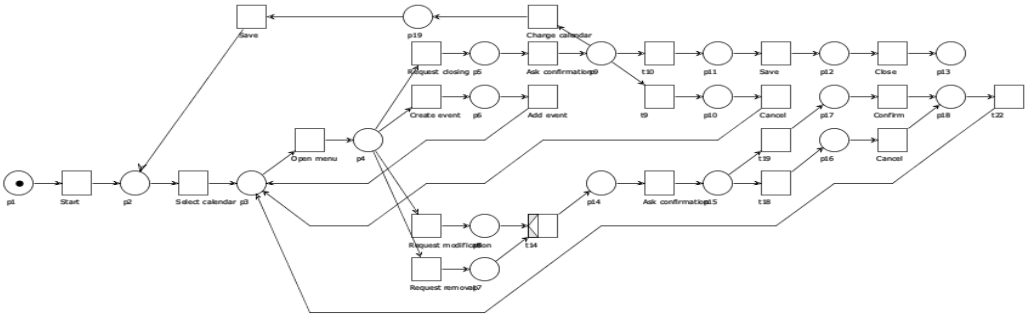
La Petri net rappresentate il processo dell'applicazione è presentata in Figura 2.5 (a) nella sua forma base e nella Figura 2.5 (b) per la variante; di seguito sono elencate alcune delle loro caratteristiche:

- la rete dell'applicazione è composta da 18 piazze e 22 transizioni per un totale di 44 archi nel primo modello, una piazza, due transizioni e quattro archi vengono aggiunti nella variante;
- le reti, sia nel modello base che nella variante, sono **workflow net** (N) e sono **safe** e **sound**;
- sono entrambe **S-system**, in quanto il pre-set e post-set di ogni transizione contiene esattamente una piazza, questo ci permette anche di trovare facilmente una *S-invariant* per le reti, che sarà dunque della forma **I**=[k k ... k];
- nessuna delle due è un **T-system** nemmeno con l'aggiunta della funzione *reset* (N*), in quanto diverse piazze hanno più di una transizione nel loro post-set o pre-set;
- sono entrambe **free-choice** in quanto per ogni coppia di transizioni i loro pre-set sono identici o disgiunti;
- sono entrambe **well-structured** poiché non presentano né *TP-handles* né *PT-handles*;
- N è **bounded** e **deadlock-free** sia nel modello base che nella variante e N* è **live** in entrambi i modelli;
- sono entrambe **S-coverable**, nello specifico da una *S-component* composta da tutte le piazze delle rispettive reti;
- sono entrambe **connected** (**strongly** se si aggiunge la funzione *reset*).

Siccome entrambe le reti sono *bounded*, il loro ***reachability graph*** è finito e coincide con il *coverability graph*. Nella figura 2.7 vengono presentati i *reachability graph* del modello base e della variante, il primo contentente 18 vertici e 21 archi, il secondo 19 vertici e 23 archi.

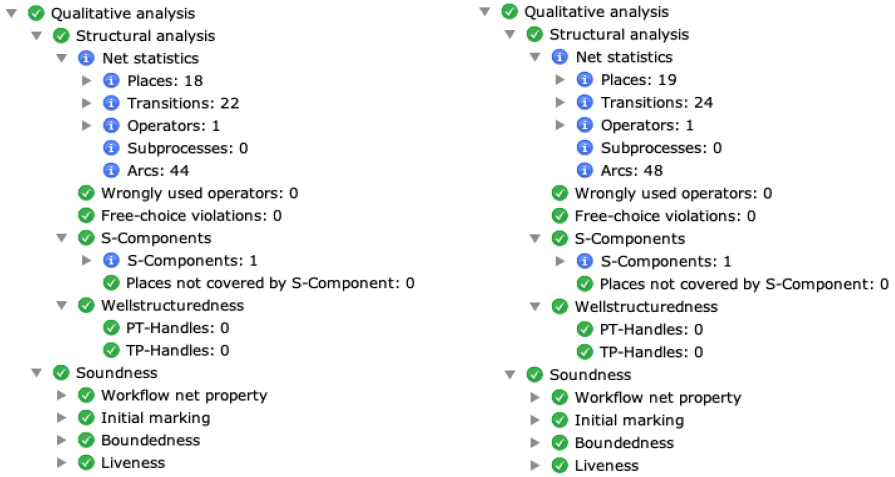


(a) Modello base



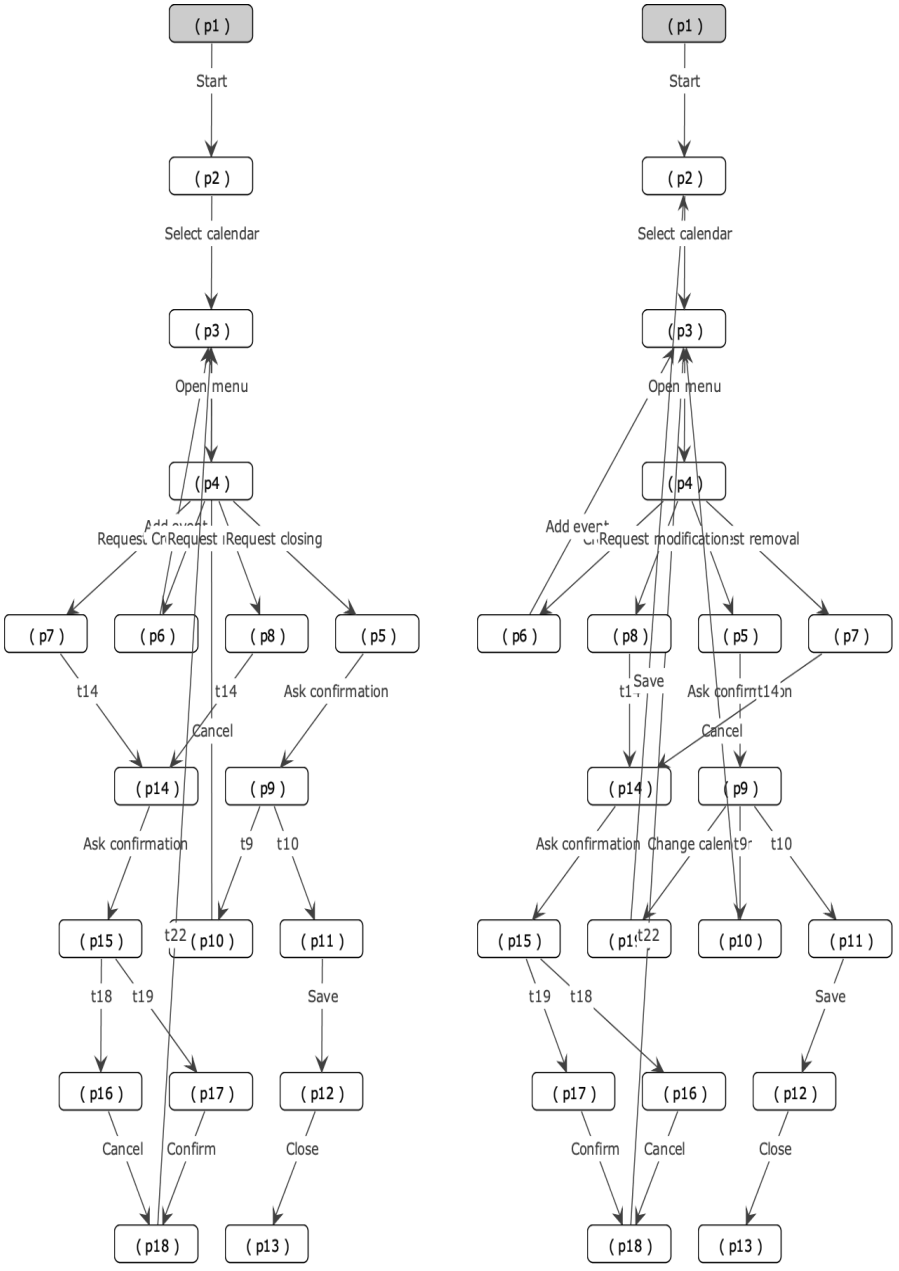
(b) Variante

Figura 2.5: Petri net rappresentanti il processo dell'applicazione



(a) Modello base (b) Variante

Figura 2.6: *Semantical analysis* delle reti dell'applicazione



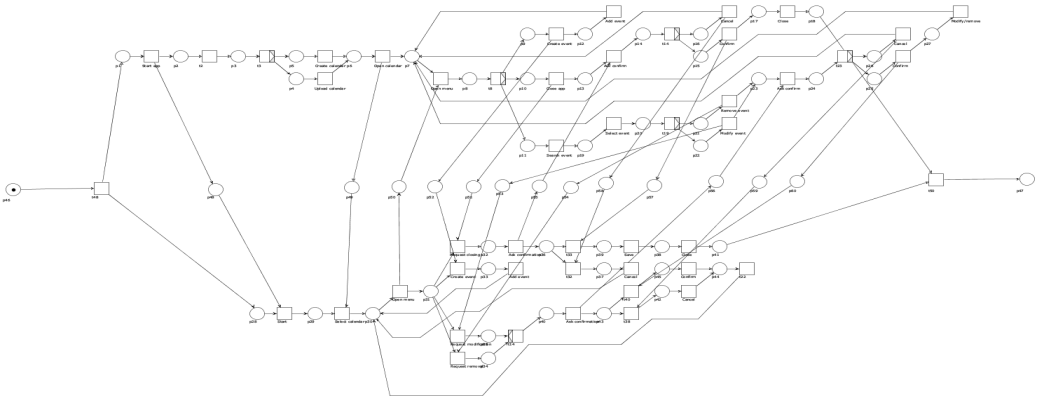
(a) Modello base (b) Variante

Figura 2.7: *Reachability graph* delle Petri net dell'applicazione

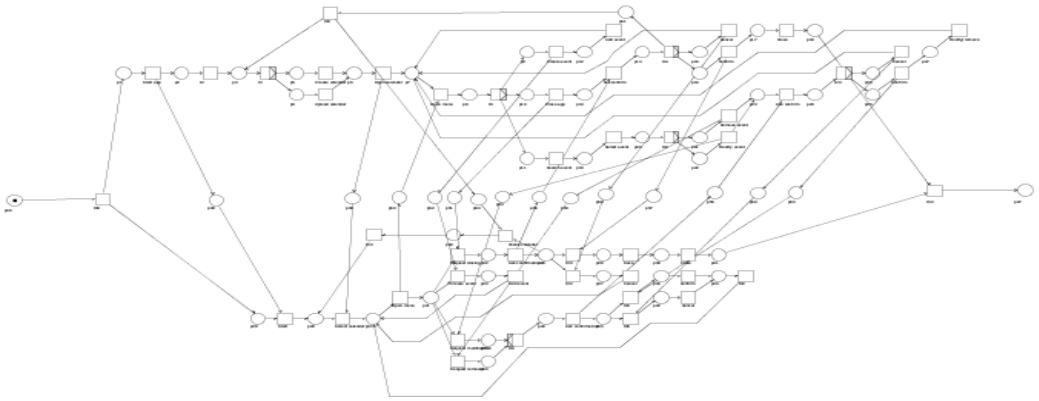
2.3 Worflow system

In questa sezione le reti dei processi dell'utente e dell'applicazione vengono trasformati in *workflow module*: al primo vengono aggiunte 10 piazze e 10 archi in uscita (11 nella variante) e 3 piazze e 3 archi in entrata, mentre è il contrario per il secondo (3 in uscita e 10/11 in entrata). Essendo essi strutturalmente compatibili, si può ottenere un *workflow system* che rappresenta l'interazione tra i due. Le reti in Figura 2.8 sono dunque state ottenute tramite l'aggiunta di 13 piazze d'interfaccia per quanto riguarda il modello base e 14 nella variante, che corrispondono ai *message flow* presenti nelle rappresentazioni BPMN. Sono inoltre state aggiunte una piazza iniziale priva di archi in entrata con una transizione ad essa collegata ed una piazza finale priva di archi in uscita con una transizione ad essa collegata. La transizione iniziale ci porta, tramite un **AND-split**, dalla piazza iniziale alle piazze iniziali dei singoli processi, mentre le piazze finali dei singoli processi sono ricondotte, tramite un **AND-join** rappresentato dalla transizione finale, alla piazza finale. Alcune caratteristiche dei *workflow system* del modello base e della variante sono elencate di seguito:

- la rete del modello base è composta da 60 piazze e 56 transizioni per un totale di 140 archi, mentre ci sono 63 piazze, 60 transizioni e 150 archi in quella della variante (sono compresi anche quelli dell'interfaccia);
- le reti, sia nel modello base che nella variante, sono **workflow net** (N) e sono **safe** e **sound**;
- nessuna delle due è un **S-system** o un **T-system**;
- nessuna delle due è **free-choice** in quanto esistono delle transizioni i cui pre-set non sono né identici né disgiunti;
- nessuna delle due è **well-structured**, infatti la prima presenta 54 *TP-handles* e 74 *PT-handles*, mentre la seconda ne presenta 86 e 86;
- N è **bounded** e **deadlock-free** sia nel modello base che nella variante e N* (N con aggiunta della funzione *reset*) è **live** in entrambi i modelli;
- sono entrambe **S-coverable**, quella del modello base da un set di 71 *S-component* mentre quella della variante da 66 *S-component*;
- sono entrambe **connected** (**strongly** se si aggiunge la funzione *reset*).



(a) Modello base



(b) Variante

Figura 2.8: Petri net dell'intero *worklow system*

- Qualitative analysis
  - Structural analysis
    - Net statistics
      - Places: 60
      - Transitions: 56
      - Operators: 6
      - Subprocesses: 0
      - Arcs: 140
    - Wrongly used operators: 0
    - Free-choice violations: 3
    - S-Components
      - S-Components: 71
      - Places not covered by S-Component: 0
    - Wellstructuredness
      - PT-Handles: 74
      - TP-Handles: 54
  - Soundness
    - Workflow net property
    - Initial marking
    - Boundedness
    - Liveness

(a) Modello base

- Qualitative analysis
  - Structural analysis
    - Net statistics
      - Places: 63
      - Transitions: 60
      - Operators: 6
      - Subprocesses: 0
      - Arcs: 150
    - Wrongly used operators: 0
    - Free-choice violations: 3
    - S-Components
      - S-Components: 66
      - Places not covered by S-Component: 0
    - Wellstructuredness
      - PT-Handles: 86
      - TP-Handles: 86
  - Soundness
    - Workflow net property
    - Initial marking
    - Boundedness
    - Liveness

(b) Variante

Figura 2.9: *Semantical analysis* del *worflow system*