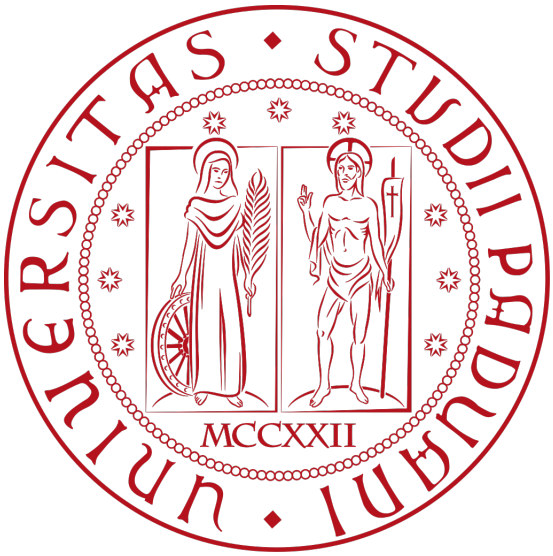
Advanced Topics in Computer Science

Report for Part#1 of Assignment (2018/2019)



Università degli studi di Padova
Laurea Magistrale in Informatica

Ticket Management Petri Net in CPN Tools

Studenti	**Fasolato Francesco** (1177742)
	Zecchin Giacomo (1179034)
Docente	**De Leoni Massimiliano**

April 7, 2019

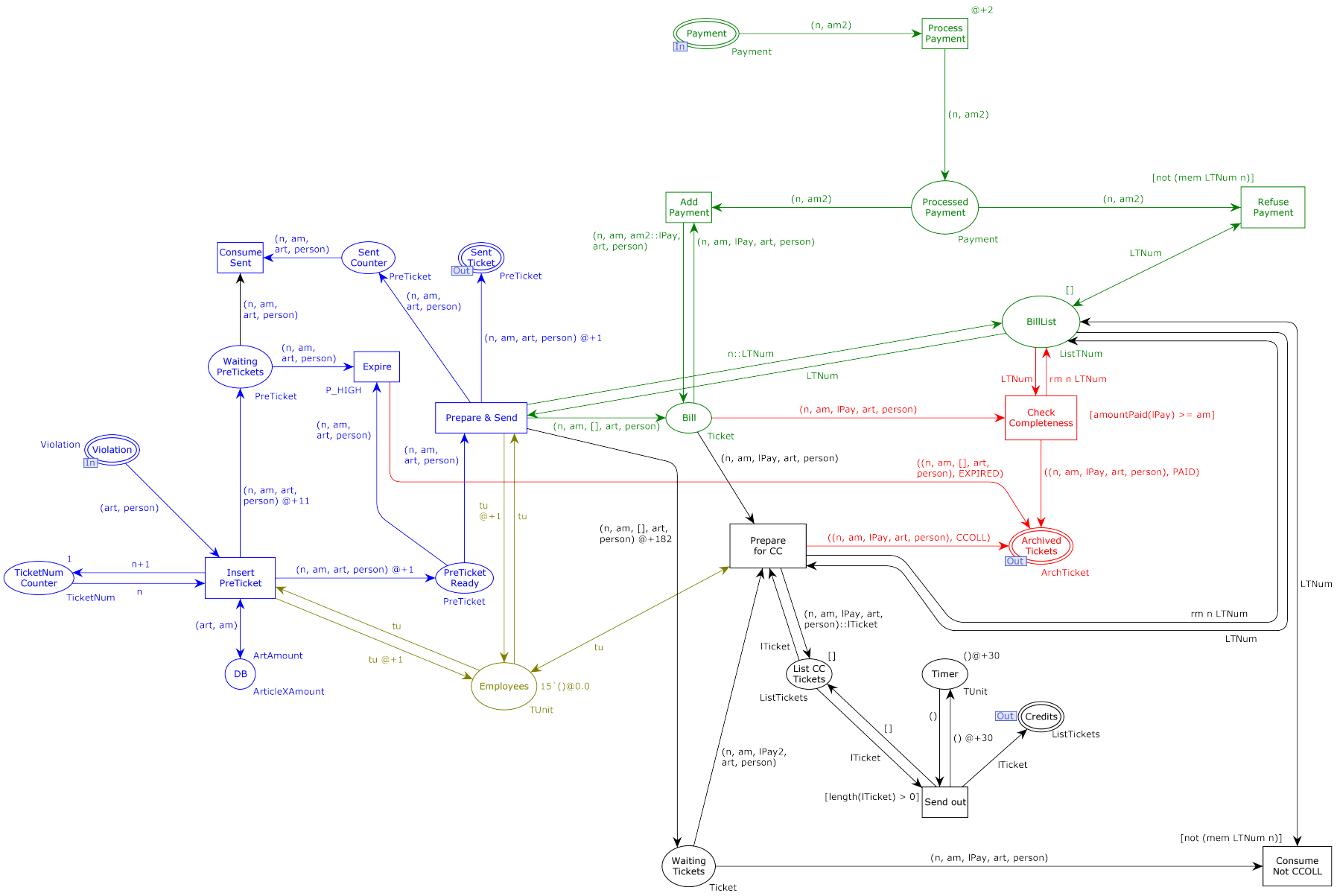


Figura 1: Ticket Management Petri Net (CPN Tools)

1 Creazione, inserimento ed invio dei ticket (subnet BLU)

1.1 Descrizione del processo

All'interno della *subnet* in analisi, `Insert Ticket` rappresenta il primo task che verrà eseguito a seguito dell'arrivo di una violazione dall'esterno del sistema. Questa attività necessita di un **Database** indipendente dove estrarre l'esatta somma da pagare (*am:Amount*) relativa al singolo sinistro commesso (identificato da `art:INT`), oltre ad un **contatore autoincrementale** (inizializzato a 1) che aumenta progressivamente al sopraggiungere di nuove violazioni e che ha lo scopo di identificare univocamente ogni ticket (tramite la variabile `n:numTicket`). Questa attività produce in output dei token provvisori di tipo `PreTicket` che contengono, oltre ad altre informazioni, la somma totale che il trasgressore dovrà pagare.
Successivamente alla creazione, i *PreTicket* sono preparati ed inviati al di fuori della sottorete attuale (tramite place `[Out] Sent Ticket`). Da qui in poi il flusso prosegue con l'archiviazione nel sistema di ticketing analizzata nella quarta sezione. Infine, le transizioni `Prepare & Send` ed `Insert PreTickets` possono effettuare il fire solamente se è disponibile almeno uno dei 15 token nel place `Employees`.

1.2 Scelte progettuali

Il primo problema da risolvere era inserire correttamente il tempo che le transizioni `Insert Ticket` e `Prepare & Send` richiedevano (entrambi 1 day). Abbiamo scelto di inserire il delay necessario nelle etichette delle frecce in uscita (da `Insert Ticket` a `PreTicket Ready` e da `Prepare & Send` a `Send Ticket`), al posto che sulle transizione stesse, altrimenti avrebbemo occupato la risorsa necessaria per fare *fire* per tutta la durata di questo tempo. Così facendo, invece, possiamo eseguire le transizioni e liberare le risorse applicando il delay solamente nell produzione del token in uscita.

Per poter tener traccia del tempo di vita di un token `PreTicket` all'interno della subnet, prima che questo raggiunga `Send Ticket` abbiamo dovuto tenere una **copia** di questi token. La produzione di queste copie impiega in totale 11 giorni (produzione: 1 day, tempo di archiviazione EXPIRED: 10 days). Nel caso in cui siano disponibili sia il token in `PreTicket Ready` sia la sua copia in `Waiting PreTickets`, sarà possibile effettuare *fire* della transizione `Expire`, che archivierà il *Ticket* con etichetta 'EXPIRED'. Visto che, in una situazione come questa, anche la transizione `Prepare & Send` è disponibile per il *fire*, ma deve essere eseguita per forza `Expire` abbiamo attribuito a quest'ultima la priorità 'P_HIGH'. Dato che alcuni token in `Waiting PreTickets` non vengono consumati (perché così configurati vengono consumati solo quelli che scadono), abbiamo dovuto creare un nuovo *place* (`Sent Counter`), dal quale arrivano tutti i token prodotti da `Prepare & Send`. Grazie alla transizione `Consume Sent` possiamo consumare anche le copie dei token prodotti da `Prepare & Send` rimanenti in `Waiting PreTickets`.

2 Pagamenti (subnet VERDE)

2.1 Descrizione del processo

Una volta che un pagamento viene ricevuto quest'ultimo viene anche processato. Da `Prepare & Send` viene prodotto un token di tipo *Ticket* nel place `Bill`. Inoltre salviamo un token di tipo *TicketNum* che aggiungiamo alla lista di tipo *ListTNum* all'interno di `BillList`. La transizione `Refuse Payment` che permette di consumare un pagamento all'interno di `Processed Payment`, può effettuare il *fire* se e solo se il token relativo al pagamento in questione si riferisce ad un ticket già archiviato (ovvero il suo *TicketNum* non è presente all'interno di `BillList`). Nel caso in cui il *Ticket* per il quale è stato ricevuto il pagamento non sia stato archiviato, è possibile procedere al pagamento aggiornando la lista dei pagamenti del rispettivo *Ticket* all'interno di `Bill`. Il place `Bill` contiente tutti i token dei *Ticket* ricevuti da `Prepare & Send` ed ancora attivi.

2.2 Scelte progettuali

Per questa sezione abbiamo creato due place distinti (`Bill` e `BillList`). Bill è il place principale che contiene i token dei *Ticket*, mentre BillList è un place ausiliario che mantiene solo una lista dei TicketNum corrispondenti ai *Ticket* 'attivi' nella subnet. `Refuse Payment`, per poter effettuare il fire ha bisogno che il *TicketNum* del *Payment* non sia presente all'interno della lista di `BillList`. Il controllo viene effettuato grazie alla funzione `[not(mem LTNum n)]` (dove LTNum identifica la lista di *TicketNum*).

3 Credit Collection (subnet NERA)

3.1 Descrizione del processo

Quando un token relativo ad un *Ticket* appena aperto viene prodotto in `Bill`, una copia di questo viene anche aggiunta, con relativo ritardo, a `Waiting Tickets`. La presenza di un token con lo stesso *TicketNum* in `Bill` e `Waiting Tickets` permette il fire di `Prepare for CC`, richiedendo anche la presenza di almeno un impiegato disponibile, della lista *lTicket* presente in `List CC Tickets` e di quella di *TicketNum* presente in `BillList`. `Prepare for CC` consuma i token presenti in `Bill` e `Waiting Tickets` e:

- restituisce la lista *lTicket* con l'aggiunta del token *Ticket* proveniente da `Bill` a `List CC Tickets`;
- restituisce la lista *LTNum* a `BillList`, dopo aver tolto il *TicketNum* relativo al ticket consumato;
- restituisce il token 'tu' ad `Employees`.

Nel caso in cui un *Ticket* venga archiviato per una ragione diversa da CCOLL, il suo *TicketNum* verrà rimosso da `BillList` e, quando verrà prodotto in `Waiting Tickets` dopo il ritardo, potrà causare il fire di `Consume Not CCOLL` (data la guardia `[not(mem LTNum n)]`), consumando i token che altrimenti rimarrebbero in `Waiting Tickets` alla fine della run.

Dopo un certo intervallo di tempo e solo se la lista *lTicket* non è vuota (guardia `[length(lTicket) > 0]`), `Send Out` può effettuare il fire. Questa transizione consuma tutti i *Ticket* presenti nella lista *lTicket* e produce un token corrispondente a questa lista nel place `Credits` (restituendo la lista vuota a `List CC Tickets`).

3.2 Scelte progettuali

L'aggiunta del tempo massimo per saldare un *Ticket* (180 giorni) è stata aggiunta alla freccia in uscita da `Prepare & Send` verso `Waiting Tickets` poiché se fosse stata aggiunta alla transizione `Prepare & Send`, questa avrebbe tenuto occupate le risorse necessarie per il suo fire per 180 giorni. In realtà il tempo è stato fissato a 182 giorni poiché un pagamento viene preso in carico solo dopo essere stato processato (e quindi 2 giorni dopo essere stato ricevuto). Se non avessimo questi 2 giorni di delay, avremmo escluso i saldi effettuati dopo 179 e 180 giorni dall'apertura del *Ticket*, violando le specifiche del sistema.

Alla transizione `Send Out` è stato collegato un `Timer` affinchè questa effettui il proprio fire ogni 30 giorni. Il place `Timer` contiente oggetti di tipo *TUnit* creati appositamente da noi per aggiungere l'attributo 'timed' a *UNIT*. Il primo token *TUnit* presente in `Timer` viene inizializzato con un ritardo di 30 giorni e viene rimesso lì dopo il fire di `Send Out` con un delay di 30 giorni.

4 Archiviazione dei ticket (subnet ROSSA)

4.1 Descrizione e scelte progettuali

Se un ticket presente in `Bill` (e quindi non ancora archiviato) presenta una somma di pagamenti effettuati maggiore o uguale all'ammontare totale della multa, allora la transizione `Check Completeness` può effettuare il fire (guardia *[amountPaid(lPay) >= am]*), richiedendo anche la lista LTNum a `BillList`.
Abbiamo inserito nel controllo anche l'operatore $>$ oltre ad $=$ poiché i pagamenti in arrivo sono casuali quindi la loro somma potrebbe eccedere quella richiesta dalla multa.
Da `Check Completeness`, `Expire` e `Prepare for CC` vengono prodotti in `Archive Tickets` dei token *ArchTicket* con *ArchiveReason* rispettivamente 'PAID', 'EXPIRED' e 'CCOLL'.

5 Simulazione del sistema con CPN Tools

Di seguito vengono riportati i risultati della simulazione dell'**intero sistema** effettuata tramite *CPN Tools*. I parametri della simulazione sono quelli standard indicati dalla consegna: **30 subruns** dove, ognuna di queste, corrisponde ad uno scenario di **5 giorni** in cui vengono commesse **40 violazioni al giorno**.

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Credit_Collection_Tickets	7	700	100.000000	50	150
Expired_Tickets	33	5975	181.060606	50	500
Paid_Tickets	160	27500	171.875000	50	500

Simulation steps executed: 3839
Model time: 483.0

Generated: Fri Apr 5 15:19:12 2019

Figura 2 Report d'esempio di una subrun - *sim_ 30*

Statistics							
Name	**Avrg**	**90% Half Length**	**95% Half Length**	**99% Half Length**	**StD**	**Min**	**Max**
Credit_Collection_Tickets							
count_iid	11.000000	1.020701	1.228566	1.655711	3.290530	3	17
max_iid	428.333333	45.297512	54.522314	73.478483	146.029835	100	500
min_iid	55.833333	6.337255	7.627832	10.279856	20.430004	50	150
sum_iid	1725.833333	237.265403	285.584314	384.875486	764.894723	675	4375
avrg_iid	161.921001	19.286442	23.214110	31.285128	62.175510	82.142857	383.333333
Expired_Tickets							
count_iid	34.200000	1.703680	2.050634	2.763592	5.492314	25	46
max_iid	500.000000	0.000000	0.000000	0.000000	0.000000	500	500
min_iid	50.000000	0.000000	0.000000	0.000000	0.000000	50	50
sum_iid	5457.500000	329.369947	396.445876	534.281092	1061.820777	3200	7950
avrg_iid	160.218191	7.378741	8.881416	11.969282	23.787539	114.285714	211.666667
Paid_Tickets							
count_iid	154.666667	1.529064	1.840456	2.480341	4.929386	144	162
max_iid	500.000000	0.000000	0.000000	0.000000	0.000000	500	500
min_iid	50.000000	0.000000	0.000000	0.000000	0.000000	50	50
sum_iid	24835.833333	644.981414	776.331366	1046.244129	2079.287048	21200	29425
avrg_iid	160.630527	4.092090	4.925441	6.637905	13.192055	133.333333	197.483221

Figura 3: Performance Simulation - Report completo