Università degli Studi di Firenze

Facoltà di Ingegneria
Corso di Laurea Magistrale in
Ingegneria Informatica

Elementi di Software Dependability

Modellazione di un algoritmo di Railways Interlocking con Simulink Stateflow

Mariano Di Claudio, Simone Ercoli e Dario Di Fina

17 maggio 2011

Indice

Elenco delle figure

Introduzione

Lo scopo di questo elaborato è quello di estendere le funzionalità sviluppate da un elaborato precedente, su di un'implementazione teorica di un algoritmo di interlocking, al fine di simulare (all'interno di una stazione fittizia) la richiesta di un itinerario, il passaggio di un ipotetico treno da un itinerario riservato e la cancellazione di un itinerario prenotato.
La macchina a stati finiti realizzata dal precedente elaborato simulava soltanto la richiesta di un itinerario da parte di un operatore esterno, gestendo la comunicazione tra scambi e circuiti di binario costituenti la stazione e rispettando le norme di sicurezza.

Nel primo capitolo presenteremo una breve introduzione sul problema dell'interlocking.
Nel secondo faremo una breve introduzione allo strumento utilizzato per la progettazione del modello, ovvero *Stateflow*, un tool di *Matlab*.
Nel terzo capitolo presenteremo il modello che è stato realizzato dal precedente elaborato, in maniera tale da poter fare il confronto con le modifiche che abbiamo apportato al progetto prima di poter implementare la nostra realizzazione.
Nel quarto capitolo presenteremo il modello finale che abbiamo realizzato.
Infine presenteremo le conclusioni.

Capitolo 1

Breve introduzione sui RIS

RIS è l'acronimo di *Railway Interlocking System*. Nell'ambito della segnalazione ferroviaria, un interlocking è l'apparato di segnalazione che previene movimenti in conflitto tra loro, al fine di garantire la sicurezza della circolazione.
La definizione ufficiale di interlocking in ambito ferroviario è la seguente:

> *"An arrangement of signals and signal appliances so interconnected that their movements must succeed each other in proper sequence."*

L'interlocking si basa sul concetto di itinerario e comunicazione tramite segnali; nasce con lo scopo di garantire la sicurezza nelle operazioni di *scelta/occupazione* di itinerari all'interno di una stazione ferroviaria (ambiente safety critical).

Per questo motivo, fra le regole fondamentali alla base del RIS, si trovano le seguenti:

- nessun segnale può permettere a più treni di muoversi se i loro movimenti danno origine ad una situazione di conflitto;
- segnali di switch come quelli che comandano ad esempio scambi e semafori, devono essere settati nella posizione corretta prima che un segnale permetta al treno di occupare il percorso richiesto;
- una volta che al treno è stato dato il segnale di occupare il percorso, tutti gli switch devono essere bloccati in posizione finché il treno non è completamente passato; potrebbe anche accadere che il permesso di procedere sia stato ritirato. In tal caso, deve essere trascorso un tempo sufficientemente lungo per assicurare che il treno coinvolto in quel segnale abbia avuto l'opportunità di fermarsi prima di ricevere il segnale stesso.

Dal punto di vista fisico, l'interlocking può essere realizzato in modo *meccanico*, *elettronico* (logica a relay) o *elettro-meccanico*.
I vari sviluppi che sono stati apportati in questo ambito hanno portato ad una standardizzazione dei sistemi di interlocking e delle loro specifiche contrattuali.

Per rispondere a queste esigenze è nato il *progetto Eurointerlocking* ad opera dei principali operatori ferroviari europei.

In esso ha preso campo l'uso di *statecharts*, e cioé l'utilizzo di linguaggi UML e dialettica di tipo Statemate; da qui la scelta di impiegare appunto strumenti di tipo chart per la risoluzione dei problemi di Railway Interlocking.

1.1 Approcci per l'implementazione del sistema di interlocking

Esistono, in tale ambito, svariati possibili approcci: tra i più significativi vi è senz'altro quello **funzionale** (sviluppato da ISTI). Esso consiste nell'utilizzo di un unico database in cui vengono trascritte tutte le regole (la condition table). Tale approccio tuttavia presenta come criticità il fatto che, qualora ci sia la necessità di modificare per qualche ragione il layout del circuito ferroviario rispetto alla condizione di progettazione, si rende necessaria una riprogrammazione totale del software di modellazione.
Per ovviare a tale problema si può scegliere una soluzione di tipo *geografica*, che si basa sulla caratterizzazione geografica del layoutappunto. Questo consente quindi che nel caso si dovesse andare a modificare il layout stesso, la speranza è che tale cambiamento non vada a coinvolgere tutto il circuito, ma solo qualcuno degli elementi che vi fanno parte, e, nella fattispecie, quelli vicini tra loro geograficamente.
In questo modo i componenti interessati (siano essi circuiti di binario o scambi) potranno essere agevolmente composti e interconnessi tra loro.

1.2 Il Model Driven Development

Il **Model Driven Software Development** è un'alternativa alla programmazione tradizionale, che non prevede la scrittura di un codice sorgente da compilare o eseguire, ma lo sviluppo di un modello del sistema che si vuole studiare. La modellazione viene per questo motivo effettuata attraverso tool visuali, dai quali sarà possibile generare il codice in un linguaggio di programmazione tradizionale in base a cosa bisogna realizzare. Quindi ciò che risulta importante sono non più il tradizionale codice, ma bensì i modelli.
Il vantaggio evidente che ne deriva è la maggior facilità nell'esprimere le specifiche e nel progetto stesso del software. Risulta altresí di fondamentale importanza la possibilità di generare il codice a partire dal modello stesso; a questo si affianca la capacità da parte dei moderni strumenti di eseguire una verifica automatica dei modelli, il che rende il model checking più semplice e rapido. Quando si esegue questo tipo di modellazione lo scopo ultimo del lavoro è quello di verificare che il modello soddisfi determinate specifiche, e ció viene fatto a livello formale tramite, appunto, il model checking.

Affinché un modello sia ben progettato, e quindi questo approccio risulti veramente efficace, occorre rispettare cinque regole in fase di progettazione:

Astrazione - Secondo il principio del rasoio di Occam, occorre, almeno in prima analisi, effettuare un'astrazione del sistema che si va a modellare trascurando dettagli ininfluenti per il particolare punto di vista. Successivamente, quando il modello riprodurrà le caratteristiche essenziali del sistema, si potrà andare a perfezionarlo introducendo dettagli sempre più accurati;

Comprensibilità - L'uso di modelli dovrebbe permettere una comprensione più intuitiva rispetto al listato di codice, e ció è permesso solo se il modello è chiaro e ben leggibile;

Accuratezza - Il modello deve fornire una rappresentazione consistente con le funzionalità del sistema che esso intende rappresentare;

Predittività - Deve essere possibile predire le proprietà del sistema, durante la simulazione o in fase di analisi formale;

Economicità - Deve essere manifestamente più conveniente analizzare il modello piuttosto che il sistema fisico reale.

Nel presente elaborato abbiamo utilizzato *Stateflow*, usato anche nel precedente elabarato per realizzazione del modello della stazione.

Capitolo 2

Elementi di base di Simulink Stateflow

2.1 Fondamenti teorici

Stateflow é uno strumento per la modellazione di *macchine a stati di Harel*. È integrato con Simulink, e insieme ad esso permette la realizzazione di modelli ad elevato livello di dettaglio.

Mentre Simulink è maggiormente impiegato nella progettazione di sistemi di controllo per impianti a tempo continuo e discreto, Stateflow è lo strumento adatto per modellare sistemi event-driven. Infatti, si presta bene ad analizzare sistemi reattivi, ovvero sistemi che modificano il proprio stato in base all'accadere di particolari eventi.

Come filosofia di base, Stateflow presenta l'approccio che fu dato da Harel alla fine degli anni '80; tale approccio parte dalle macchine a stati finiti e ne sviluppa le potenzialità grazie ad aspetti quali, ad esempio, la gerarchia (presenza di stati con priorità diverse) ed il parallelismo, ampiamente usati nel precedente elaborato.

2.2 Elementi costitutivi di Stateflow

Essendo Stateflow nato per modellare macchine a stati finiti, gli elementi fondamentali messi a disposizione dal tool saranno stati e transizioni tra stati.
Gli stati del modello sono elementi reattivi, cioé si passa da uno stato ad un altro della macchina al verificarsi di certi eventi, cioé il verificarsi di particolari condizioni.

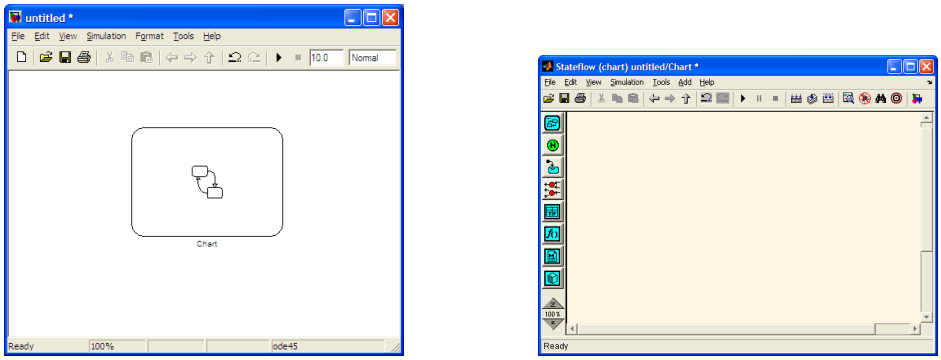

Figura 2.1: Schermata iniziale di Stateflow. A sinistra è mostrato come appare un chart di Stateflow all'apertura; all'interno di esso si troveranno gli elementi mostrati nell'immagine di destra.

2.2.1 Stati

Ogni stato costituisce una modalità operativa di una macchina.
Per esempio, in una macchinetta per l'erogazione di bevande sono stati l'erogazione di una bevanda, o lo stato di *idle* in cui la macchina è in attesa che arrivi un utente a inserire la monetina.

Uno stato puó essere attivo o non attivo; uno stato non attivo diventa attivo all'accadere di un evento ad esso associato. Come possiamo vedere dalla figura 2.2, uno stato attivo si distingue da uno stato non attivo dalla colorazione, infatti gli stati attivi hanno i bordi evidenziati di colore blu.

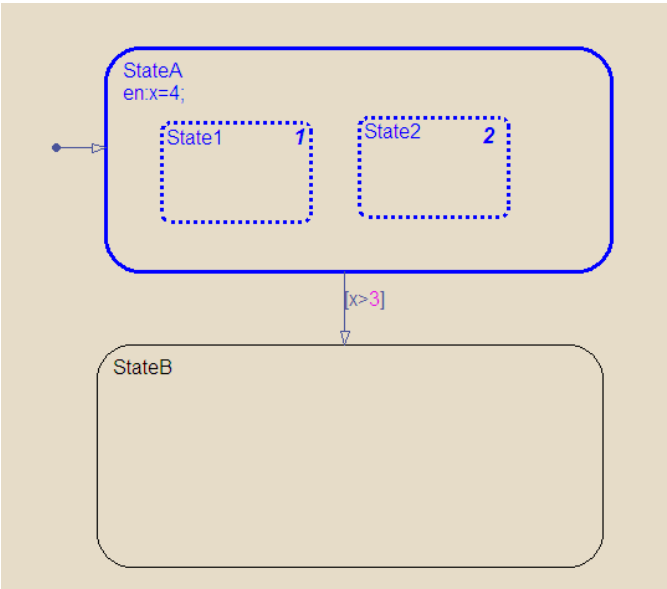

Figura 2.2: Stati attivi e non attivi

Ovviamente è possibile inserire degli stati dentro altri stati per creare una

gerarchia; chiaramente ogni stato ha un processo padre (in figura 2.2 lo stato A é padre degli stati 1 e 2); se il diagramma fosse composto da un solo stato, il padre sarebbe il diagramma stesso.

Ad ogni stato viene assegnato un nome, che lo distingue dagli stati allo stesso livello gerarchico. Due o più stati possono essere tra loro:

- Stati paralleli (AND). In questo caso durante la simulazione vengono attivati nello stesso istante (sono nell'esempio fatto gli stati 1 e 2) e sono graficamente riconoscibili dai bordi tratteggiati;
- Stati esclusivi (OR). In questo caso vengono eseguiti in modo sequenziale; poiché hanno la stessa priorità, occorre sempre specificare quale deve essere eseguito per primo, tramite una default transition. Sempre nell'esempio di figura 2.2 sono stati esclusivi lo stato A e lo stato B. In questo caso, la default transition specifica che durante la simulazione lo stato A diventa attivo per primo.

2.2.2 Transizioni

Le transazioni sono degli oggetti grafici che collegano altri oggetti (non necessariamente due stati). Possono connettere tra loro:

- Due stati
- Uno stato e una junction
- Due junction
- Una junction e un oggetto box
- Uno stato e un oggetto box

E' importante notare come le transizioni siano direzionate. Inoltre, vi sono dei particolari tipi di transizione, le *default transition*, che si usano quando si vuole specificare qual é lo stato iniziale tra una serie di stati di uguale priorità.

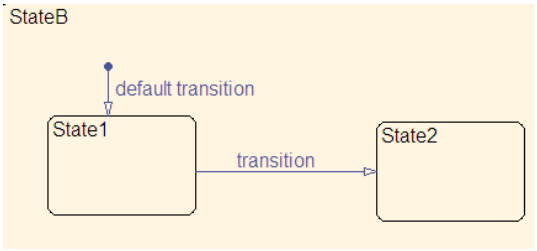

Figura 2.3: Transizione tra due stati e default transition

2.2.3 Condizioni

Le condizioni sono espressioni booleane associate alle transizioni. La transizione diventa superabile quando la condizione é verificata. La sintassi per esprimere una condizione é:

$$[x == 0]$$

che sta ad indicare che la transizione diventa superabile quando la variabile x assume valore pari a zero.

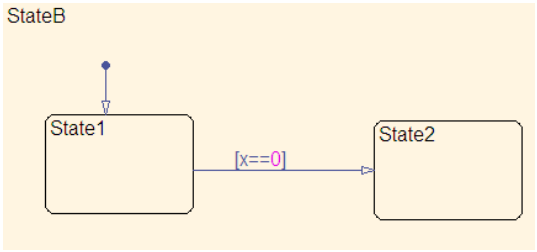

Figura 2.4: Condizione booleana che determina il passaggio da uno stato all'altro

2.2.4 Azioni

Le azioni possono essere [1]:

- specificate nell'etichetta di una transizione. In questo ambito si hanno due ulteriori possibilità:
  1. azioni legate alla condizione. Esse graficamente si esprimono come:

     $$\{x = 1\}$$

     dove viene banalmente messo a 1 il valore della variabile x;
  2. azioni legate alla transizione, che si indicano in modo diverso rispetto alle prime:

     $$\backslash x = 1$$

- azioni eseguite all'interno di uno stato. In questo caso si distinguono i seguenti tipi di azione :
  1. **entry** (**en**). Sono le azioni eseguite appena lo stato diventa attivo;
  2. **during** (**du**). Vengono eseguite per tutto il tempo che lo stato resta attivo;
  3. **exit** (**ex**). Vengono eseguite quando lo stato diventa inattivo, ovvero si esce da quello stato;

[1] le parole in grassetto rappresentano delle keyword di Stateflow

4. "**on event**" L'azione viene compiuta quando si verifica l'evento "event" specificato.

Quanto appena detto viene illustrato in figura 2.5.

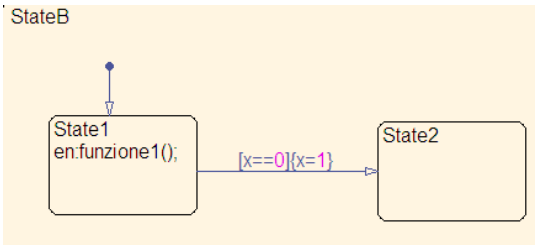

Figura 2.5: Azioni eseguite all'interno di uno stato e durante una transizione

2.2.5 Dati

I dati sono oggetti non grafici, che costituiscono le variabili impiegate nel modello.

Essi sono accessibili dal *Model Explorer* (figura 2.6), in cui risulta particolarmente importante settare il tipo di dato e la visibilità.

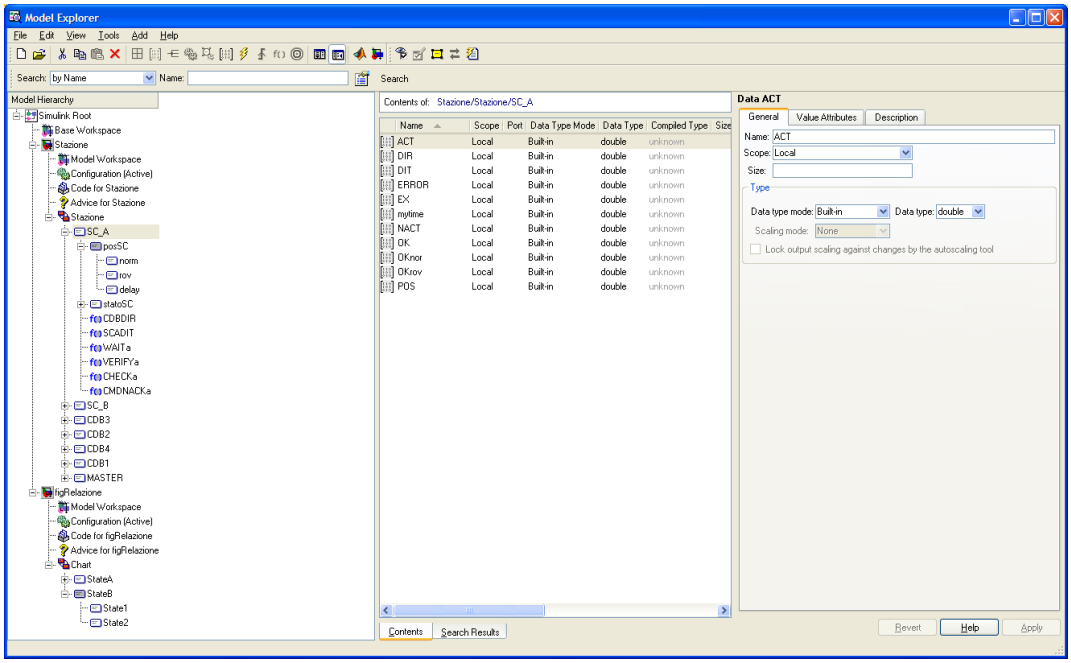

Figura 2.6: Model Explorer in cui sono definiti i parametri del chart

Puó essere interessante notare che, quando si ha a che fare con vettori e matrici (come succede nel modello sviluppato in questo elaborato), la notazione

adottata da Stateflow é la seguente:

$$A[0][0] = A_{11}$$

Ovvero, nell'esempio fatto, si considera l'elemento nella prima riga e nella prima colonna della matrice A, in quanto la posizione all'interno di un vettore viene contata a partire dallo zero.

2.2.6 Eventi

Analogamente ai dati, anche gli eventi sono oggetti non grafici, che hanno una funzione simile agli interruttori, guidando l'evoluzione del chart.

Anche gli eventi vanno definiti nel *Model Explorer*, e puó esserne impostata la visibilità. Va detto che è possibile il broadcasting di eventi tra elementi dello stesso chart tramite il comando **send**; ció avrebbe senz'altro agevolato la comunicazione tra i vari circuiti di binario, ma nel modello sviluppato si è preferito adottare una diversa soluzione, soprattutto per evitare possibili problemi in fase di model checking.

2.2.7 Junction

Le *junction* sono elementi grafici che rappresentano punti di decisione del sistema, utilizzati in generale per implementare i principali costrutti, come ad esempio: *for*, *switch-case* e *if-then-else*.

Le transizioni connesse alle junction sono chiamate *segmenti* e hanno le stesse caratteristiche di tutte le altre transizioni.

2.2.8 Funzioni grafiche (flowchart)

Le *funzioni grafiche* sono dei diagrammi di flusso in cui compaiono junction e transizioni interconnesse.

Possono avere degli argomenti in ingresso e restituire valori in uscita; la sintassi corretta é:

$$z = f(x, y)$$

dove z costituisce il valore di ritorno, e x e y gli argomenti passati in ingresso alla funzione.

Tutto ció che è stato detto appare chiaro dalla figura 2.7, in cui sono riportate sia la chiamata alla funzione grafica, che la funzione grafica stessa e il codice Matlab equivalente corrispondente a tale funzione.

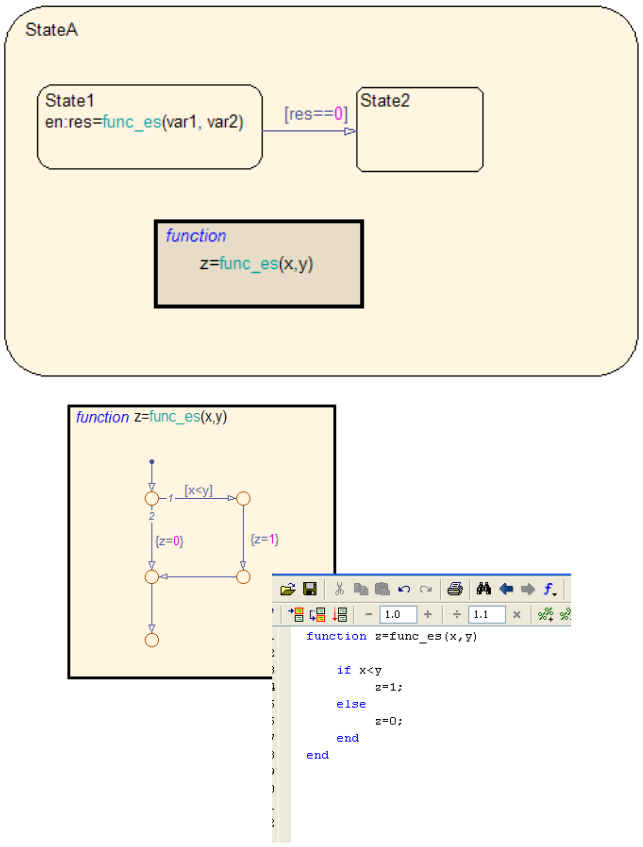

Figura 2.7: Esempio di utilizzo di una flowchart

Capitolo 3

Precendente elaborato

3.1 Specifiche

Rispetto al precedente elaborato abbiamo aggiunto delle funzionalità in più che ci permettessero di poter soddisfare gli obiettivi inizialmente proposti. Per poter inserire le nuove funzionalità è stato necessario rispettare le stesse specifiche del precedente elaborato.

Per quanto detto sull'approccio geografico, si suppone che ogni nodo sia connesso ad almeno due nodi vicini e conservi una tabella di itinerari a cui appartiene (servirà per sapere con chi il nodo deve comunicare in base all'itinerario scelto).

In prima approssimazione i nodi potranno essere solamente di due tipi:

- circuiti di binario. Gli stati corrispondenti saranno: libero, occupato o riservato su route x;
- scambi. Essi potranno trovarsi in stato: normale, rovescio, comandato a normale, comandato a rovescio in parallelo a libero, occupato, riservato su route x.

Si definisce inoltre la topologia nel seguente modo:

- un cdb è adiacente al più a due cdb/scambi, detti adiacente sinistro, adiacente destro (in caso di assenza, è un binario morto);
- uno scambio è adiacente a tre cdb/scambi;
- uno scambio sinistro ha un adiacente destro, un adiacente normale sinistro, un adiacente rovescio sinistro;
- uno scambio destro ha un adiacente sinistro, un adiacente normale destro, un adiacente rovescio destro.

Per tali ragioni appare evidente come il grafo di adiacenza sia aciclico.

Per quanto riguarda gli itinerari si definiscono le regole:

- un itinerario è costituito da un cammino diretto formato da enti di binario adiacenti;
- il primo e l'ultimo nodo di ogni itinerario sono cdb;
- il primo nodo definisce il punto di richiesta itinerario;
- il primo nodo deve essere occupato alla richiesta;
- la richiesta verifica la libertà degli enti successivi;
- la richiesta procede solo in direzione destra o in direzione sinistra.

Nella creazione del modello si seguiranno i seguenti passi:

1. definire macchina a stati per ciascun ente;
2. definire il protocollo di comunicazione tra due enti adiacenti;
3. definire le proprietà di safety desiderate.

L'algoritmo funziona così: in una prima fase si ha solo richiesta e occupazione.

Dopo una prima fase di configurazione, parte l'algoritmo vero e proprio, in cui è possibile richiedere un itinerario comunicandolo al nodo punto di inizio. Al termine della prenotazione di un itinerario (e durante questa fase, vista la natura distribuita e quindi concorrente dell'interlocking), si possono anche richiedere altri itinerari.

In termini di eventi, si ha che quando un itinerario viene richiesto, il circuito di binario iniziale (gli scambi possono essere solo intermedi), se è libero passa a riservato, e invia la richiesta al suo adiacente appartenente all'itinerario. L'ultimo circuito di binario, se è libero, si mette in riservato e invia un segnale di *aknowledge* ripercorrendo il tragitto a ritroso, facendo sì che tutti i nodi interessati si mettano a riservato.

Quando si è giunti di nuovo al primo cdb, esso invia al proprio adiacente un *commit*, come ulteriore verifica, che viene inoltrata fino all'ultimo cdb. Se questi è d'accordo, invia indietro un segnale di *agree*. Quando l'*agree* torna al primo circuito di binario, è il segnale che tutti i controlli sono andati a buon fine, l'itinerario richiesto è stato riservato e il treno può partire.

La figura 3.1 schematizza quanto appena detto.

Va detto che se qualche controllo non andasse a buon fine (per esempio, un nodo riceve la richiesta di itinerario mentre è riservato per un altro) viene mandato un messaggio di *nack* (non-acknowledge) e la richiesta per l'itinerario viene annullata.

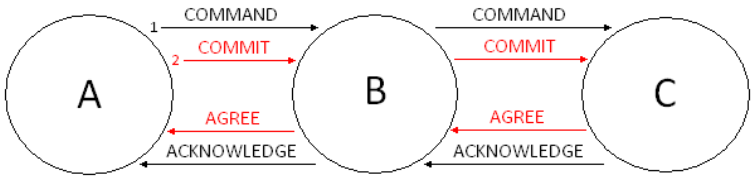

Figura 3.1: Diagramma delle comunicazioni tra i nodi

L'assunzione di fondo è che la comunicazione tra nodi proceda all'andata da sinistra verso destra, al ritorno da destra verso sinistra [1].

Sono presenti tre stati; libero, riservato e occupato. Le transizioni tra gli stati sono scandite dagli eventi secondo la seguente notazione:

- $CMDIT$. E' la richiesta dell'itinerario IT che viene passata dal primo cdb all'ultimo. Se tutti i controlli vanno a buon fine (nessun nodo era già stato riservato o occupato per un altro itinerario) allora l'ultimo cdb manderà indietro un segnale di
- ACK. Esso costituisce la conferma che tutti i nodi interessati dall'itinerario richiesto si sono messi a riservato. Quando l'ACK raggiunge il primo elemento dell'itinerario, esso invia al proprio adiacente un
- $COMMIT$. E' un secondo segnale di verifica che viene trasmesso tra i nodi dall'inizio alla fine dell'itinerario. Quando arriva all'ultimo cdb, se esso "è d'accordo" invia indietro un
- $AGREE$.Quando questo comando arriva al primo cdb, esso avrà la conferma che tutto l'itinerario richiesto è stato effettivamente prenotato[2]. Il treno può partire.

Va precisato che esistono altri due possibili eventi: uno è il $DISAGREE$, corrispondente all'eventualità che il controllo in fase di check sia andato male per qualche nodo. In questo caso, viene segnalato tramite la variabile $ERROR$.

L'altro è l'evento $SENSORE$, che modella la presenza fisica di un treno sul cdb; infatti, essa potrebbe essersi verificata volutamente, ma anche in seguito ad eventi imprevisti, e di questo viene tenuto conto nel modello.

Infine, sono presenti le variabili ULTIMO e PRIMO. Se ne comprende l'utilità se si considera il caso di un cdb che costituisce proprio l'ultimo nodo dell'itineario; quando gli viene passato il segnale $CMDIT$, esso non deve trasmetterlo alla

[1] Se ciò non dovesse risultare intuitivo qualora l' itinerario richiesto fosse per esempio il $2-3$, basta pensare di ribaltare il punto di vista, mettendosi come un osservatore che dal cdb3 guarda il binario 4. Apparirà subito evidente come la convenzione adottata risulti corretta in qualunque senso si percorra il circuito.

[2] altrimenti viene attivata la variabile $ERROR$, con la quale il comando di itinerario viene resettato, e insieme ad esso lo stato del sistema, che può tornare libero.

sua destra, ma, essendo l'ultimo, deve rispondere indietro con un ACK.

Le differenze riguardano il fatto che uno scambio può essere comandato a normale o a rovescio, mentre tra lo stato di $CHECKING$ e quello di OK è stato aggiunto uno stato di $MOVING$ che dovrebbe modellare il tempo impiegato dallo scambio per portarsi da una posizione all'altra.

3.2 Modello adottato

Il modello adottato fa riferimento ad una implementazione ottimizzata nel precedente elaborato e in sintesi si compone di un chart denominato *Stazione*, all'interno del quale sono stati inseriti secondo una logica distribuita tutti i nodi. All'interno del chart, l'ente *Master* é quello che deve essere attivato per primo e che regola il controllo dell'esecuzione. I meccanismi di comunicazione tra i nodi (CDB e SCAMBI) si basano sull'assegnazione di valori a variabili create per ogni azione di ogni nodo.

Un secondo aspetto importante é la possibilitá di richiesta di due itinerari, come illustrato chiaramente all'interno del *Master*.

3.2.1 Master

Si faccia riferimento alla figura 3.2.

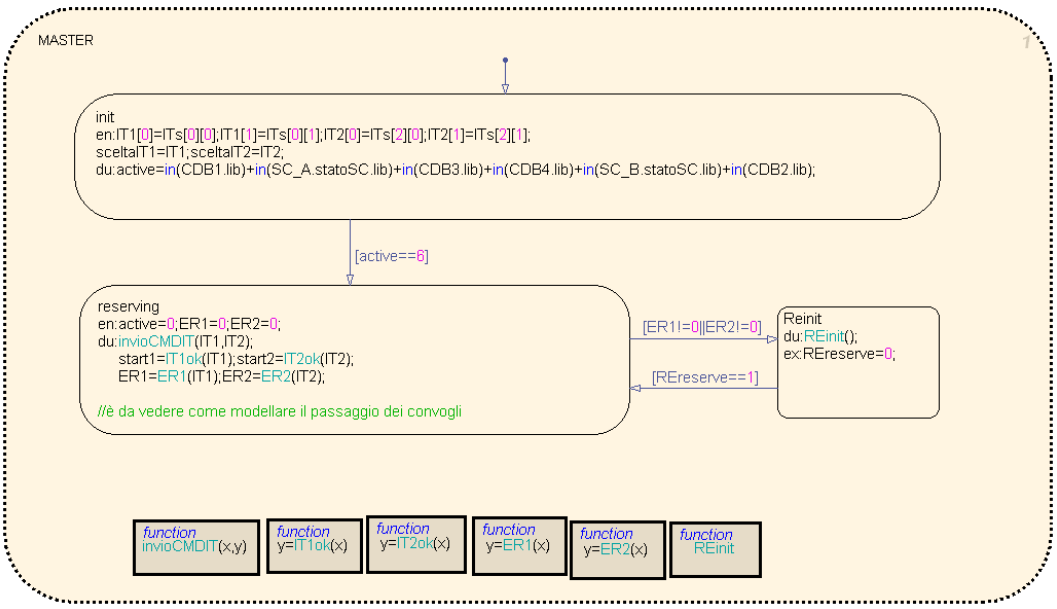

Figura 3.2: Dettaglio del *Master*

Le variabili $IT1$ e $IT2$, entrambe vettori di dimensioni 1×2, rappresentano i due itinerari scelti all'interno della matrice passata in ingresso al chart. Quan-

do tutti i nodi sono nello stato di libero, si attiva la transizione che porta in *reserving*; all'interno di esso la funzione $invioCMDIT$ provvede, tramite confronti sugli elementi dei vettori passati in ingresso, all'attivazione dei cdb iniziali interessati ai due itinerari richiesti.

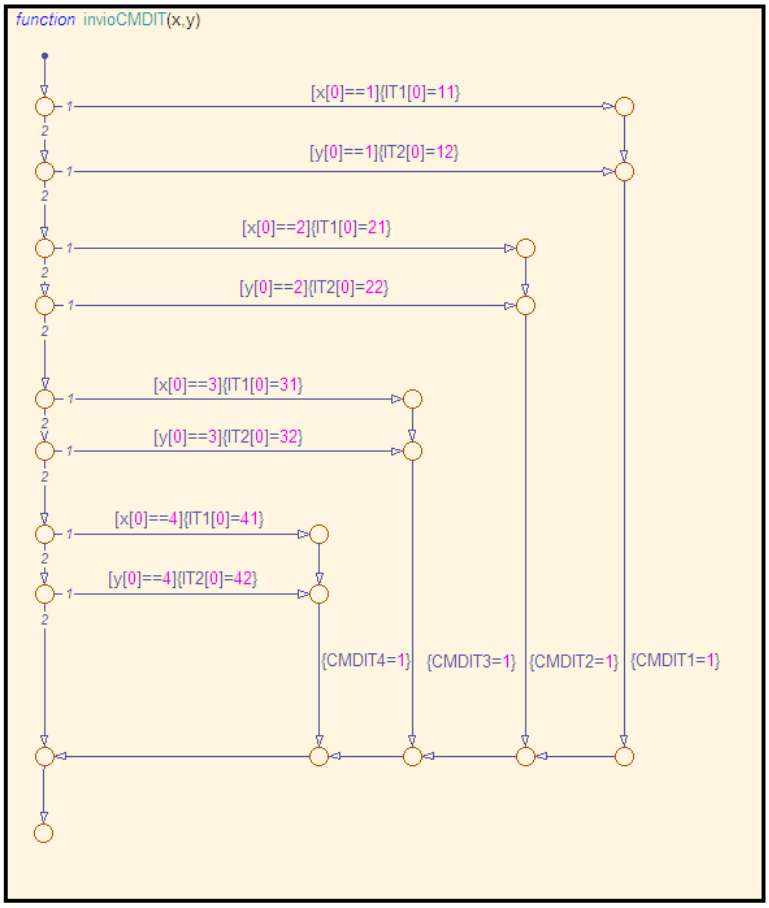

Figura 3.3: Master. Flowchart "invioCMDIT"

Tale attivazione avviene tramite la variabile globale

$$CMDITx$$

in cui x è l'indice del cdb di inizio itinerario; la variabile viene inizializzata a zero ed è impostata a 1 quando si vuole "inviare" il comando di prenotazione itinerario al nodo x. In tale modo, si riesce a simulare la comunicazione tra nodi senza ricorrere al **send** di eventi.

In figura 3.3 si possono notare, associate alle condizioni, delle azioni che assegnano (in base a una precisa logica) determinati valori alle variabili di itinerario; ciò è necessario prima di tutto per non ripercorrere, a ogni ciclo della simulazione, lo stesso ramo che "invia" il $CMDITx$.

Inoltre, servono a riconoscere in fase di errore e re-inizializzazione quel preciso itinerario la cui prenotazione non è andata a buon fine. [3]

In via del tutto analoga al $CMDITx$ vi sarà una variabile per ogni evento che si intende inviare.

$IT1ok$ e $IT2OK$ servono a sapere se tutti i controlli sono andati a buon fine: se il cdb iniziale dell'itinerario è passato a ok, significa infatti che ha ricevuto prima un $ACKNOWLEDGE$ e poi un $AGREE$ da tutti i partecipanti; quindi le variabile $start1$ e $start2$ danno in qualche modo la conferma definitiva che gli itinerari richiesti sono stati riservati. [4]

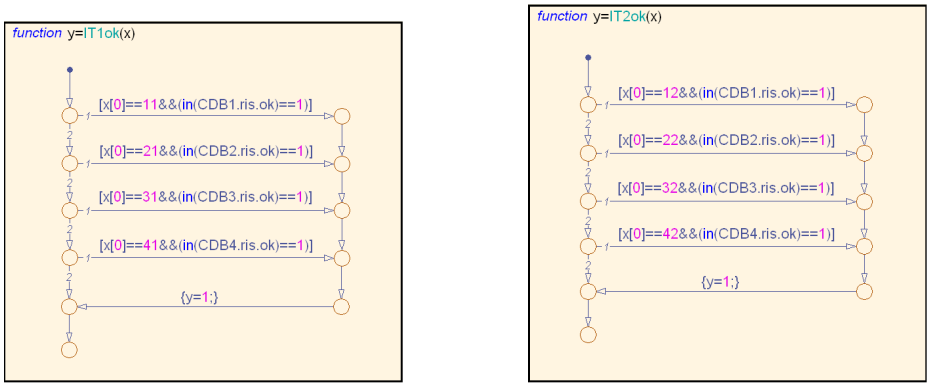

Figura 3.4: Master. Flowchart "IT1ok" e "IT2ok"

Le funzioni $ER1$ e $ER2$ svolgono un compito analogo, anche se restituiscono una variabile che segnala la presenza di un errore.

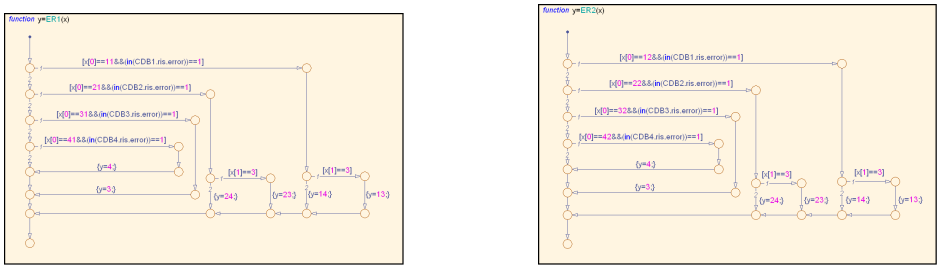

Figura 3.5: Master. Flowchart "ER1" e "ER2"

Si consideri ad esempio $ER1$.

Le diverse condizioni presenti rappresentano le situazioni in cui, a causa di un conflitto tra due itinerari, un nodo va nello stato di errore e di conseguenza

[3] Quanto appena detto viene illustrato dettagliatamente con le funzioni grafiche $ER1$, $ER2$ e $REinit$.

[4] Queste variabili non saranno realmente utilizzate perché le fasi relative al passaggio del convoglio, ovvero le transizioni di ogni nodo riservato verso lo stato occupato non sono state modellate.

anche il primo dell'itinerario; al verificarsi di una di queste condizioni si procede con l'assegnare certi valori alla variabile di uscita.

Il fatto che essa per esempio assuma il valore 13 significa che si è verificato un errore relativo al primo itinerario richiesto (perché ci si sta riferendo a $ER\mathbf{1}$), dove il primo nodo dell'itinerario è il cdb1, e l'ultimo il cdb3; con questa notazione, non solo si potrà eseguire un ulteriore controllo sul comando che ha generato errore, ma soprattutto si potrà scegliere un itinerario alternativo nella funzione *REinit*.

Per fissare le idee, qualora dovesse essere dato errore appunto sull'itinerario $1-3$, esso viene modificato in $1-4$, in modo da tentare se tale nuovo itinerario (alternativo al primo) potrebbe invece essere disponibile.

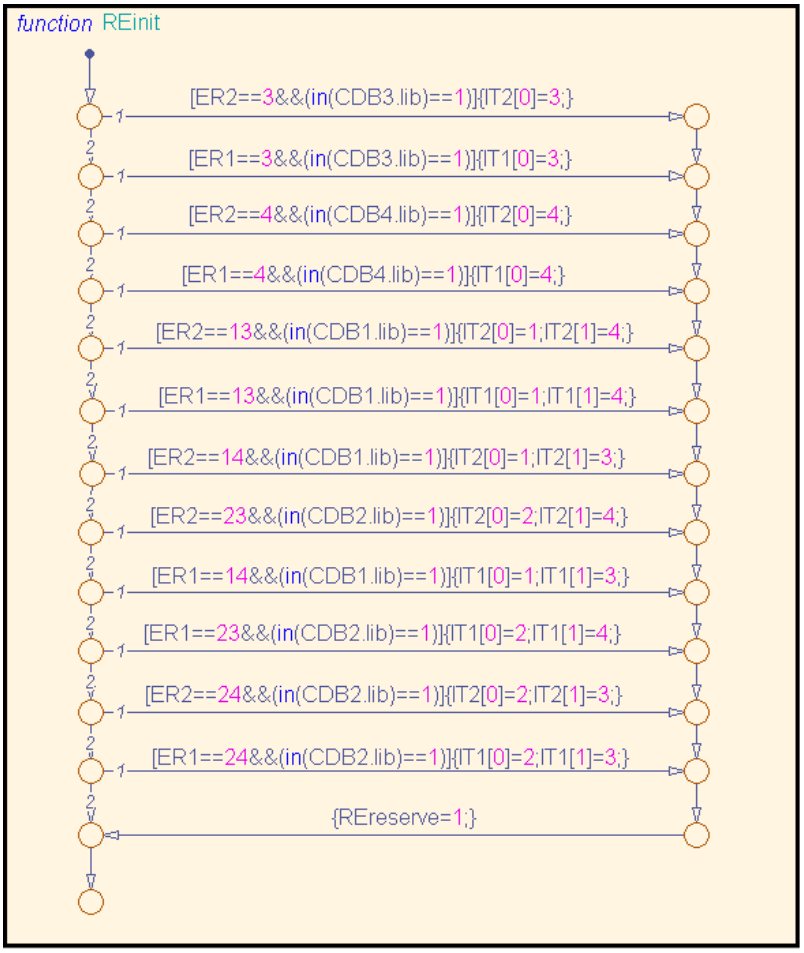

Figura 3.6: Master. Flowchart "REinit"

3.2.2 Circuito di binario 3

Si faccia riferimento alla figura 3.7[5].

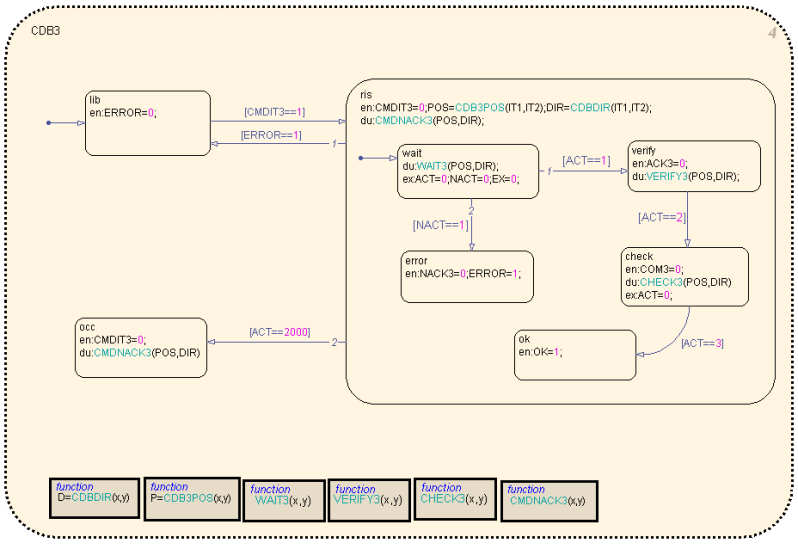

Figura 3.7: Circuito di binario 3

Si ritrovano gli stati già menzionati: libero, riservato e occupato. Le transizioni tra gli stati sono già state discusse ampiamente nei paragrafi precedenti; sarà sufficiente notare come il passaggio per esempio tra gli stati interni a *ris* avviene con variabili locali al nodo (ACT e $NACT$), cioè le transizioni scattano quando tali variabili vengono messe a un valore diverso da zero.

In modo analogo a quanto visto per il $Master$ si andrà ad analizzare in dettaglio le funzioni grafiche, all'interno delle quali si trovano le scelte implementative peculiari di questo modello rispetto a quello teorico di partenza.

In figura 3.8 si trovano le prime due.

A tali funzioni vengono passati in ingresso i due itinerari richiesti; $CDB3POS$ stabilisce con un controllo la posizione del cdb3 (primo elemento dell'itinerario, intermedio o ultimo)[6].

$CDBDIR$ stabilisce invece il senso di percorrenza, rappresentato dalla variabile d. Se $d = 1$, significa che il treno si sta muovendo da sinistra verso destra ed è zero altrimenti.

La funzione $WAIT3$ è caratterizzata da differenti casi, che si distinguono per: la posizione all'interno dell'itinerario (nell'immagine la variabile x), la ricezione di un ACK o un $NACK$. Al verificarsi di una di queste condizioni viene effet-

[5]E' stato volutamente scelto il cdb3 a titolo di esempio perchè le funzioni sono definite rispetto ad un adiacente destro ed uno sinistro, vantaggio "offerto" dalla posizione intermedia e più significativo rispetto al caso dei cdb1 e cdb2.

[6]La notazione adottata è $pos = 2$ se il nodo è il primo, $pos = 1$ se è intermedio e $pos = 0$ quando è l'ultimo dell'itinerario.

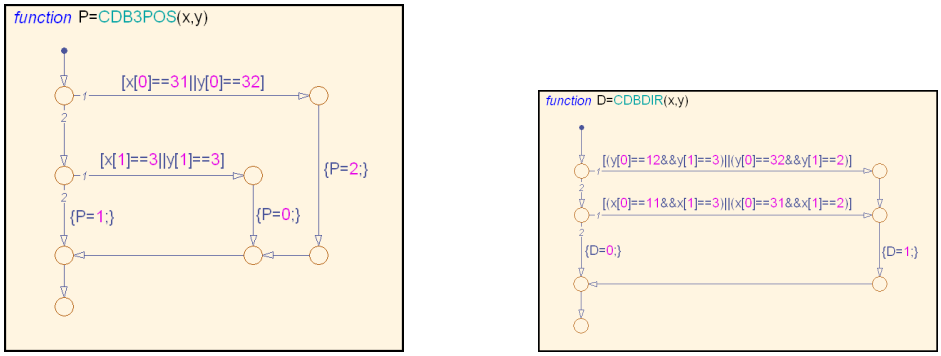

Figura 3.8: cdb3. Flowchart "CDB3POS" e "CDBDIR"

tuato un controllo sulla variabile direzione (y), per definire a quale nodo inviare l'informazione giusta.

Ad esempio, se arriva la richiesta di prenotazione dell'itinerario 31 ($x = 2$ e $y = 0$), la prima condizione che si verifica è quella che porta all'azione

$$\{CMDITa = 1;\}.$$

La seconda che si dovrà verificare, se tutto va bene, sarà

$$[ACK3 == 1]$$

che porta a eseguire la transizione verso lo stato $VERIFY$ ($ACT = 1$) perché la fase successiva sarà l'invio del commit.

Si notano anche alcune azioni sui rami di uscita per ogni condizione, in questo caso con le variabili EX, ACT e $NACT$; la prima è necessaria solo per non ripercorrere la stessa condizione più di una volta durante la simulazione, mentre le altre due hanno il compito di far eseguire l'appropriata transizione nello stato ris una volta che si attraversa quel preciso ramo.

In $VERIFY3$ il cdb può ricevere il segnale di COM(COMMIT) dal suo adiacente, il che autorizza la transizione allo stato successivo ($ACT = 2$), oppure se è il primo e non può ricevere il segnale da nessuno, provvede a mandarlo allo scambio A.

Per la funzione $CHECK3$ si sfrutta ancora la stessa strategia, stavolta con il segnale di AG (agree). [7]

Il passaggio nello stato *ok* avviene se tutti i controlli sono stati affermativi; l' informazione sarà passata al $Master$, che gestirà il tutto. Se invece qualche cosa

[7] Il fatto che, alla ricezione di un segnale di commit o agree nelle rispettive funzioni, non venga eseguito alcun invio di ulteriori segnali, significa che è stata presa in considerazione l'assenza di situazioni con cdb intermedi (solo gli scambi possono esserlo); in futuro, quando verranno aggiunti nuovi itinerari e anche questo cdb potrà diventare intermedio, sarà necessario aggiungere altre opportune condizioni a queste funzioni, similmente a come è stato fatto per le corrispondenti degli scambi.

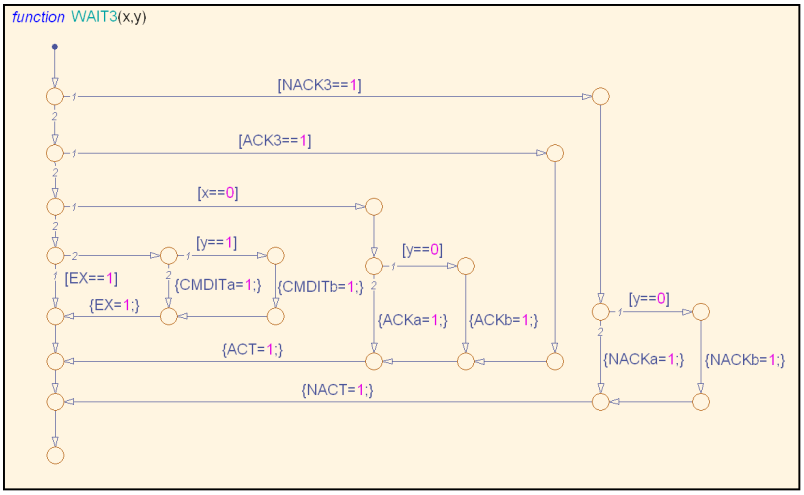

Figura 3.9: cdb3. Flowchart "WAIT3"

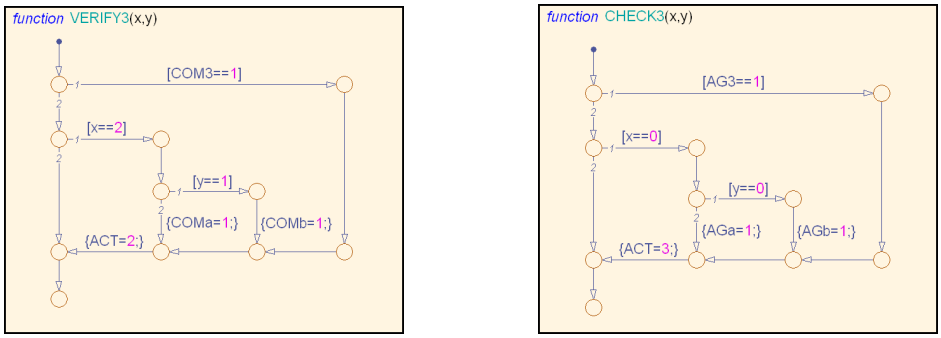

Figura 3.10: cdb3. Flowchart "VERIFY3" e "CHECK3"

è andato storto, si passa nello stato di *error*; anche di questo il *Master* terrà conto, mentre il singolo cdb rimetterà il proprio stato a *libero*. [8]

Infine, si trova la funzione $CMDNACK3$.

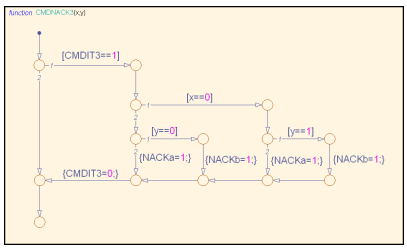

Figura 3.11: cdb3. Flowchart "CMDNACK3"

[8] Si fa notare che nessuna azione imposta la variabile *ACT* affinché sia abilitata la transizione da riservato a occupato; questo perché non è stata modellata nel presente elaborato la presenza fisica di un treno sul binario.

Banalmente, essa modella il fatto che può giungere una richiesta di itinerario quando il cdb è già riservato o occupato; in tal caso deve essere inviato un $NACK$ al proprio adiacente. [9]

3.2.3 Scambio *A*

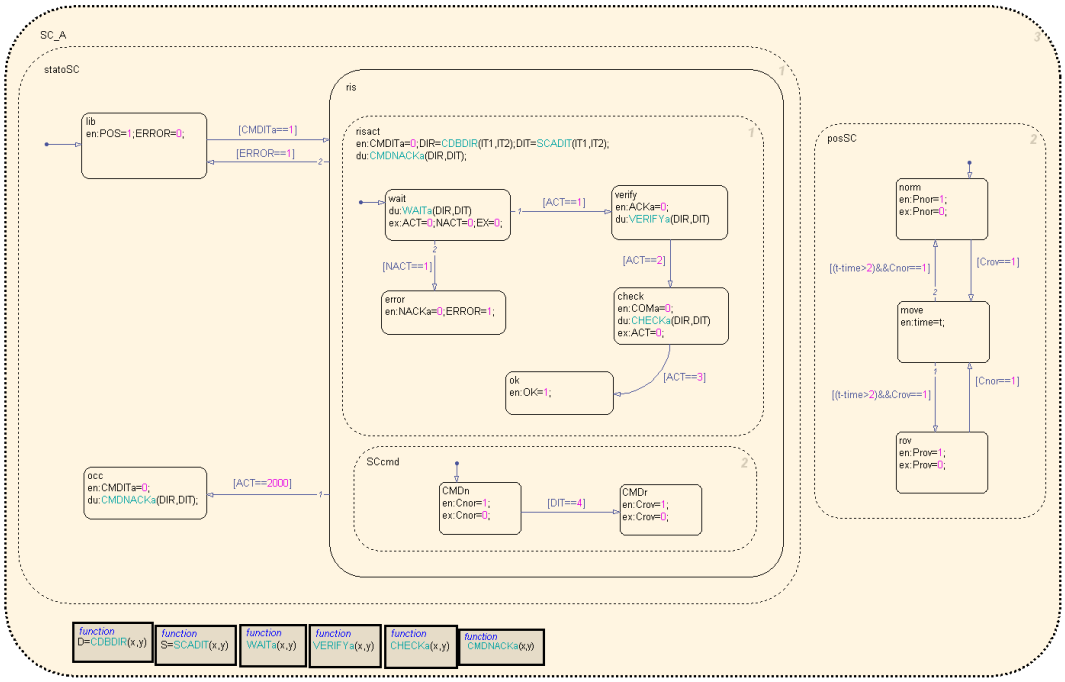

Figura 3.12: Scambio A

In figura 3.12 viene riportato lo schema principale dello scambio. Ancora una volta, si ritrova nella struttura principale quanto già visto nel modello teorico, e, in parte, nel primo cdb.

Le cose da notare a livello generale sono che lo *statoSC*, che racchiude gli stati libero, riservato e occupato, è in parallelo con *posSC*, che definisce la posizione dello scambio.

Essa può essere, come già detto, normale o rovescio; in questo caso è stata aggiunta alla transizione che passa da una configurazione all'altra un controllo relativo alla variabile *time*. In pratica, si simula un ritardo nel cambiamento di posizione dello scambio, dovuto alla meccanica dello scambio stesso. **t** è infatti una keyword di Stateflow che rappresenta il tempo di esecuzione della simulazio-

[9] Siccome questo è un comando che deve procedere verso sinistra nell'itinerario che sta cercando di prenotare il cdb riservato, è stato necessario effettuare gli appropriati controlli sulla posizione e direzione del cdb.

Ad esempio: se il cdb3 è riservato sull'itinerario $1-3$ ($POS=0$ e $DIR=1$), è automatico pensare che una seconda richiesta di prenotazione itinerario possa arrivare solamente dallo scambio B, quindi il $NACK$ andrà verso di esso.

ne.

Nel caso di uno scambio, non esiste una funzione che stabilisce la posizione; essa è chiaramente sempre intermedia. Invece, si ritrova $CDBDIR$, che imposta la variabile d a 1 se si procede da sinistra verso destra e a zero se viceversa.

La funzione $SCADIT$ restituisce la variabile DIT, che indica con che cdb comunicherà lo scambio in base all'itinerario scelto (per lo scambio A, potrebbero essere il cdb1, il cdb3 o il cdb4).

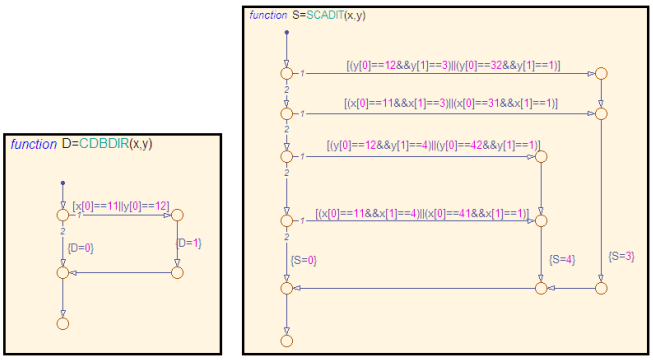

Figura 3.13: scambio A. Flowchart "CDBDIRa e SCADIT"

All'interno della funzione $WAITa$ si capisce subito la ragione di aver definito le due varibili DIR e DIT.

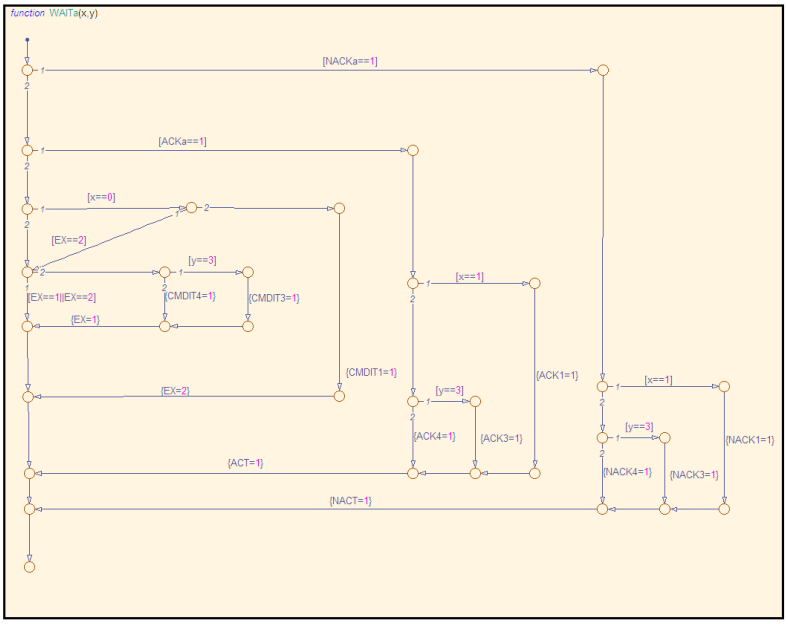

Figura 3.14: scambio A. Flowchart "WAITa"

Mentre il concetto di fondo resta lo stesso della funzione $WAIT3$, devono essere effettuati ulteriori controlli perché devono essere mandati segnali diversi (o aggiornate variabili diverse) a seconda dell'itinerario scelto e quindi del cdb che costituisce l'adiacente dello scambio.

Analogo discorso si può fare con la funzione $VERIFYa$; poiché uno scambio può essere solo intermedio, esso dovrà attendere che sia attivata da uno dei suoi adiacenti la variabile $COMa$. Quando ciò viene fatto, sempre a seconda dell'itinerario scelto, esso passa il $COMMIT$ al proprio adiacente sinistro, e si porta nello stato di $CHECK$.

In esso, si chiama la funzione $CHECKa$, che svolge lo stesso compito di $VERIFYa$, ma con il segnale di AG (agree) e l'aggiunta della condizione $[Pnor == 1||Prov == 1]$; questa è necessaria per identificare il completamento del movimento fisico dello scambio (normale o rovescio), quindi dà la possibilità di terminare la prenotazione (stato OK) e far muovere il treno.

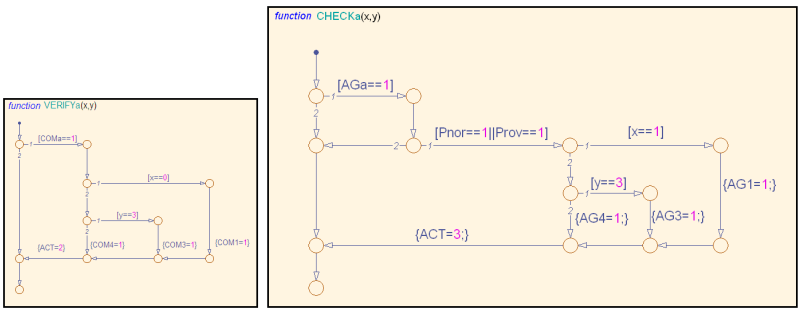

Figura 3.15: scambio A. Flowchart "VERIFYa" e "CHECKa"

Infine, la funzione $CMDNACKa$ provvede ancora a inviare un segnale di $NACK$ qualora dovesse essere fatta una richiesta di itinerario mentre lo scambio risulta già occupato o riservato.

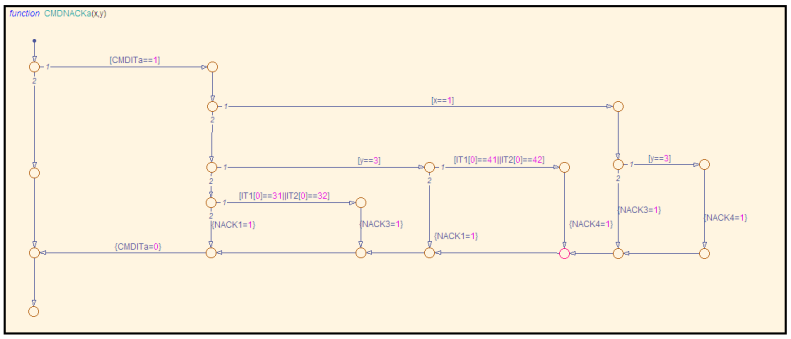

Figura 3.16: scambio A. Flowchart "CMDNACKa"

In appendice saranno riportati per completezza anche gli altri circuiti di binario e lo scambio B; nel presente paragrafo non saranno discussi dal momento

che sarebbero una banale ripetizione di quanto già spiegato per il terzo cdb e lo scambio A.

Ciò che cambia infatti sono soltanto le adiacenze, e quindi le variabili che vengono attivate per determinare l'accadere di eventi, mentre i concetti restano gli stessi.

3.2.4 Model Explorer

Sempre per completezza, vengono fornite di seguito alcune immagini che illustrano come si presenta il Model Explorer del chart realizzato, una volta che sono state definite tutte le variabili e ne sono stati impostati il tipo e la visibilità.

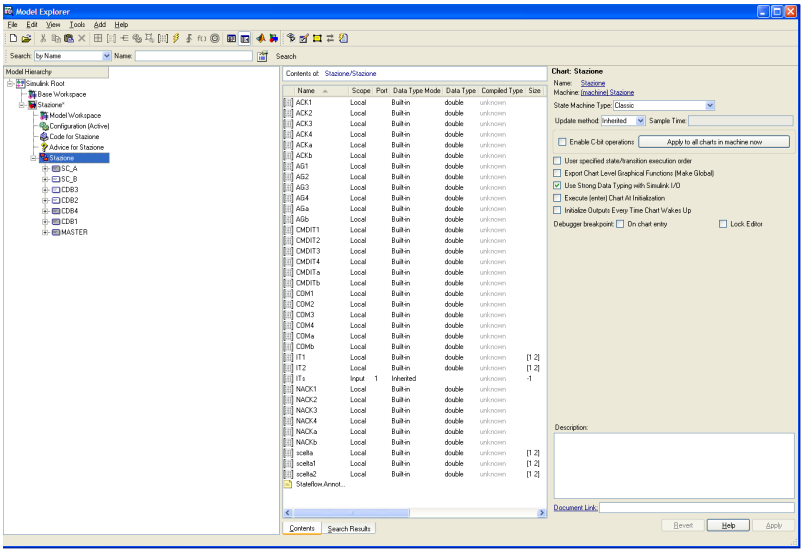

Figura 3.17: Model Explorer dell'intero chart

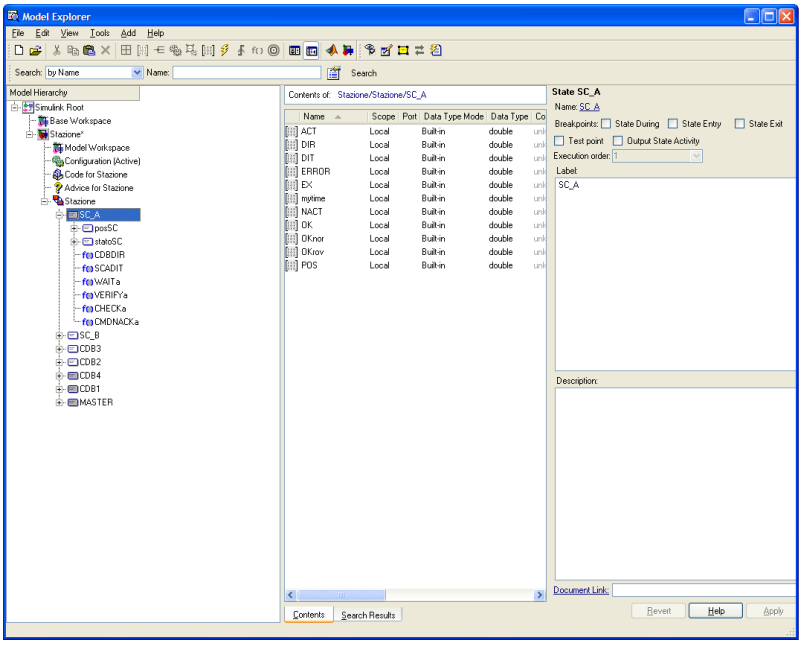

Figura 3.18: Model Explorer relativo allo scambio *A*

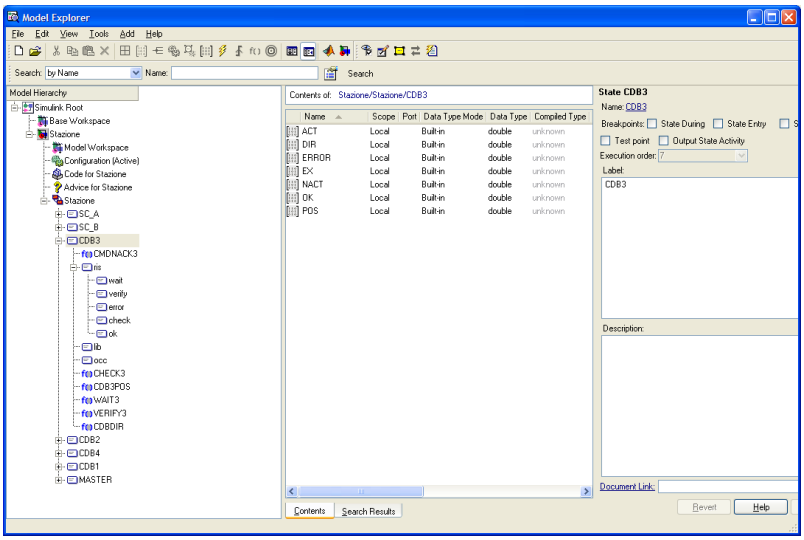

Figura 3.19: Model Explorer relativo al cdb3

Capitolo 4

Implementazione del nostro modello

Per quanto riguarda l'implementazione eseguita per questo elaborato, il lavoro complessivo é stato incentrato nel realizzare, partendo dal modello giá esistente illustrato nei capitoli precedenti, i seguenti punti :

- simulazione passaggio treno da itinerario precedentemente prenotato
- cancellazione di un itinerario prenotato

Oltre a questi due punti chiave, il modello gestisce anche situazioni di conflitto nella prenotazione e nel passaggio dei treni.

4.1 Passaggio di un treno

La simulazione del passaggio treno avviene immediatamente dopo la prenotazione di un itinerario. In ogni *CDB* é quindi stato aggiunto un "simulatore di tempo" rappresentato da uno stato chiamato *simul temp* con lo scopo di rappresentare nel nostro modello il passaggio del treno. Non é stata aggiunta nessuna simulazione di tempo negli scambi poiché si é supposto che il passaggio su questi ultimi avvenga in maniera istantanea. In aggiunta allo stato *simul temp* é presente anche lo stato *occ* il quale ha lo scopo di rappresentare l'occupazione fisica del binario. Una volta che l'ultimo binario dell'itinerario prenotato é passato ad occupato, in cascata si vanno a liberare i binari e gli scambi iniziali coinvolti nel percorso.

In definitiva i due stati aggiunti nei vari *CDB* sono quelli mostrati in figura 4.1 , mentre negli scambi A e B é stato aggiunto solo lo stato di occupato *occ* come mostrato in figura 4.2

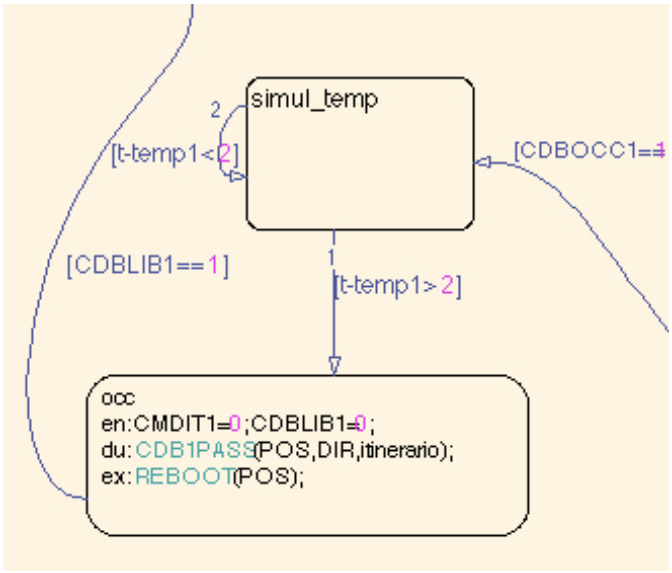

Figura 4.1: Simulazione tempo e stato di occupato

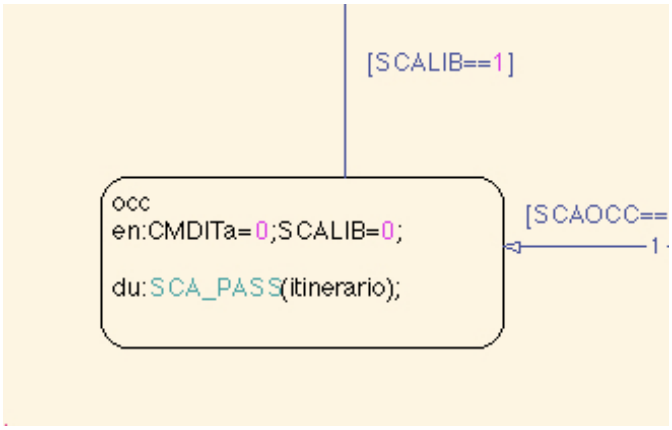

Figura 4.2: Stato di occupato per lo scambio A

Per gestire la liberazione, dopo il passaggio di un treno, degli scambi e dei vari CDB coinvolti in un itinerario é stata usata la funzione *CDB1PASS* (come esempio esplicativo prendiamo solo quella relativa al primo CDB) la quale prende in ingresso la variabile *POS* che sta ad indicare la posizione del CBD corrente nell'itinerario (un CDB puó essere finale, centrale o iniziale), la variabile *DIR* che sta ad indicare la direzione in cui viaggia il treno e la variabile *itinerario* che , come indica il nome, sta a rappresentare l'itinerario.
Tramite queste tre variabili la funzione (figura 4.3) va a liberare (stiamo simulando la liberazione degli scambi e dei CDB dopo il passaggio di un treno), gli scambi e i CDB coinvolti nell'itinerario.

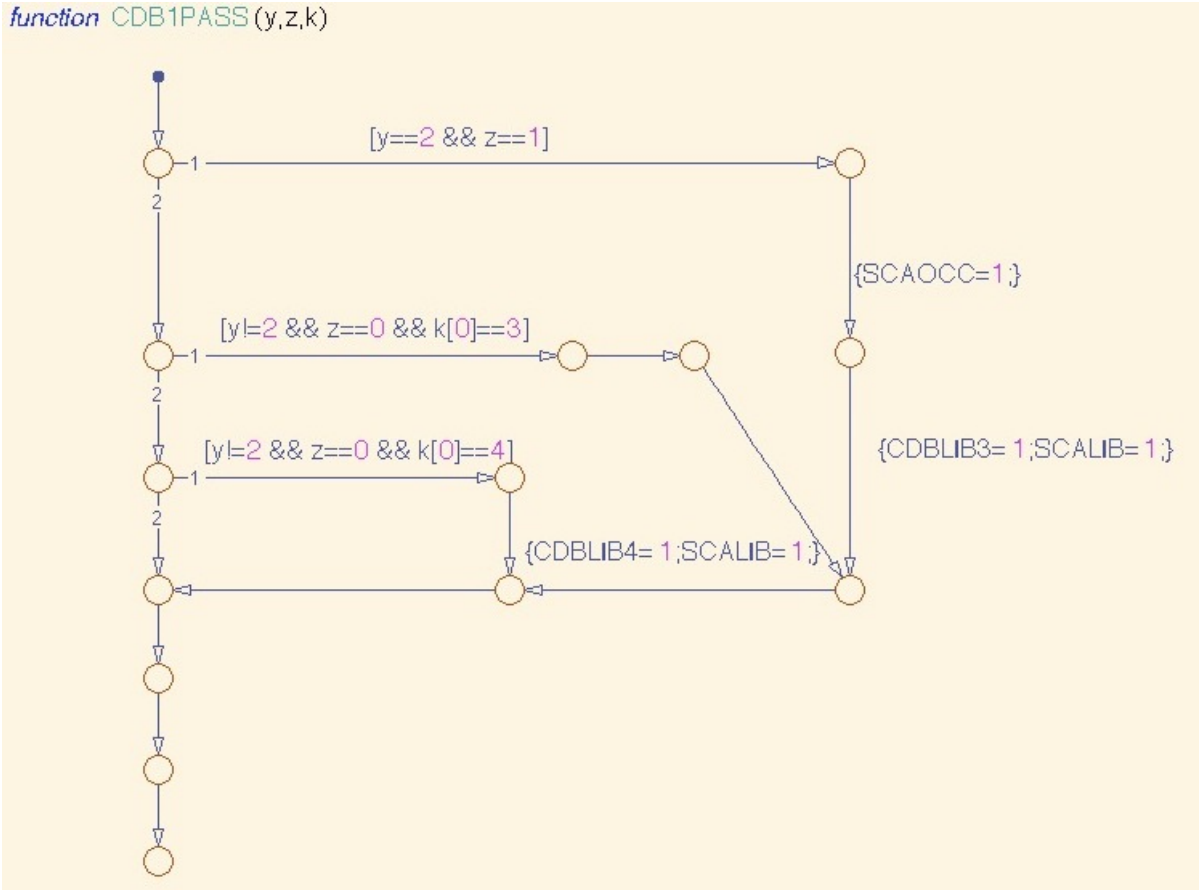

Figura 4.3: Funzione per la liberazione dopo il passaggio

4.2 Cancellazione itinerario

L'altro punto sul quale abbiamo lavorato é stato quello di implementare la cancellazione di un itinerario , procedura non presente nel vecchio elaborato.
La cancellazione riguarda un itinerario precedentemente prenotato, i cui CDB e scambi devono essere rimessi nuovamente a liberi.
Per implementare questa funzionalitá é stato aggiunto a ciascun CDB ed a ciascuno scambio lo stato interno *del* il quale a sua volta si compone di due stati interni *wait* e *ok* come mostrato in figura 4.4 (a titolo di esempio é stato preso solo lo stato relativo al CDB1).

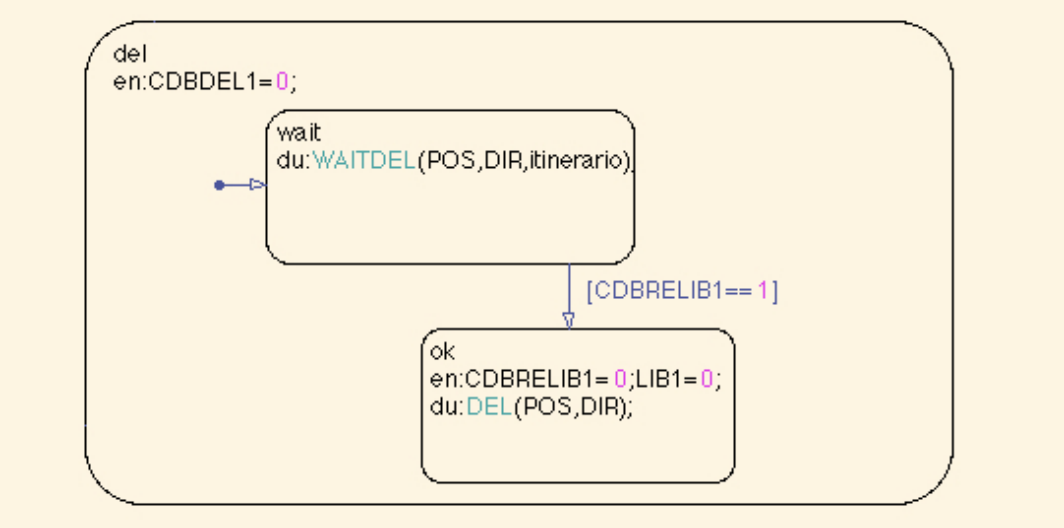

Figura 4.4: Stato per la cancellazione di un itinerario

Tramite la funzione *WAITDEL* che prende in ingresso i parametri *POS*, *DIR* e *itinerario* e lal funzione *DEL* che prende in ingresso i parametri *POS* e *DIR* il sistema é in grado di cancellare un itinerario prenotato precedentemente all'arrivo di un segnare speciale di "cancellamento" andando a liberare nell'ordine i vari CDB e gli cambi coinvolti.
Per completezza vengono ora presentate le due funzioni coinvolte nel cancellamento (sempre relativo al CDB1).

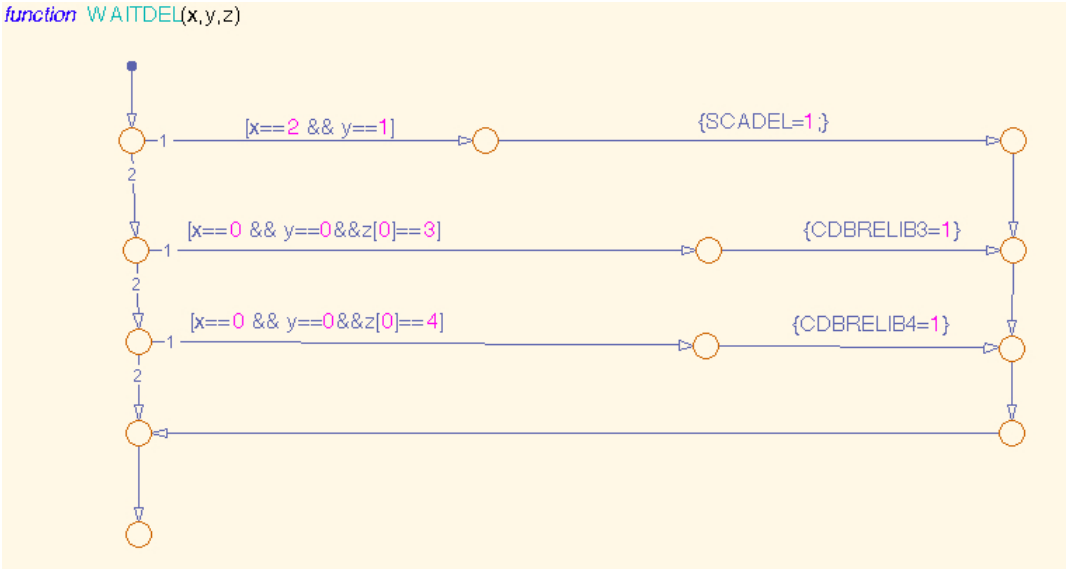

Figura 4.5: funzione WAITDEL

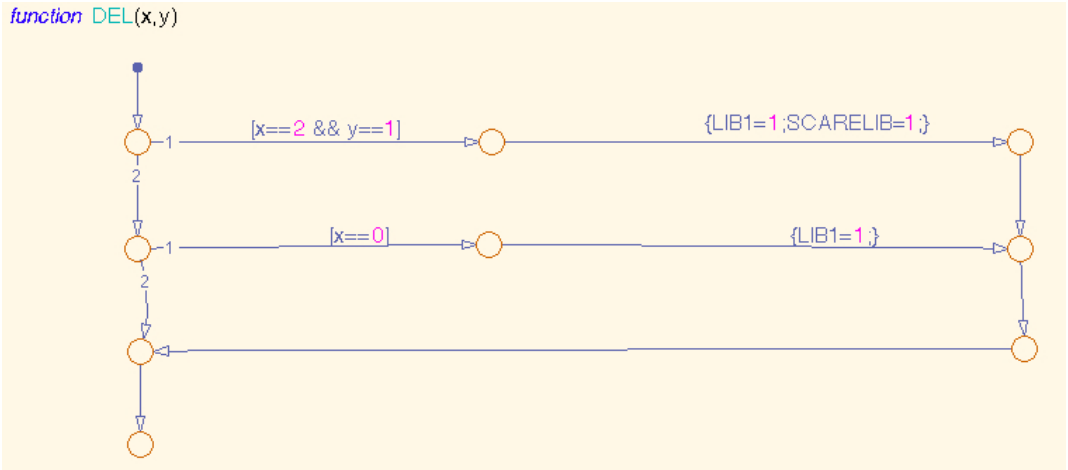

Figura 4.6: funzione DEL

4.3 Modifica del Master

Per far fronte alle nuove funzionalitá introdotte (cancellazione itinerario e passaggio del treno) si é dovuto modificare anche il *Master* (figura 4.7) ,ovvero lo stato che funziona da ”controllore” del sistema e permette di mandare i vari segnali simulando una stazione ferroviaria.

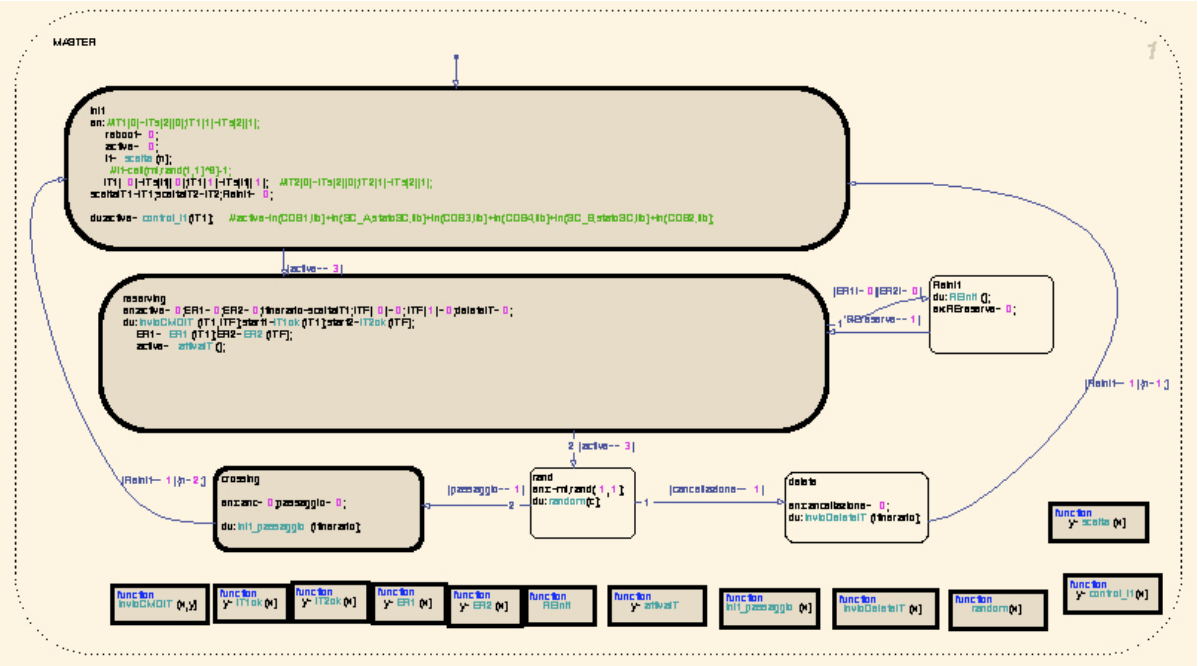

Figura 4.7: Master

4.3.1 Randomizzazione

Per rendere la simulazione piú realistica é stata introdotta una piccola randomizzazione nella scelta del passaggio del treno oppure della cancellazione dell'itinerario. Tutto ció é stato possibile con l'utilizzo di uno stato *rand* (figura 4.8) e di una funzione *random* (figura 4.9)

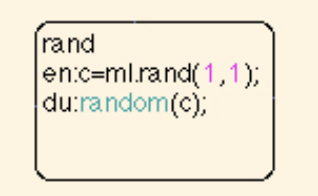

Figura 4.8: Stato per il random

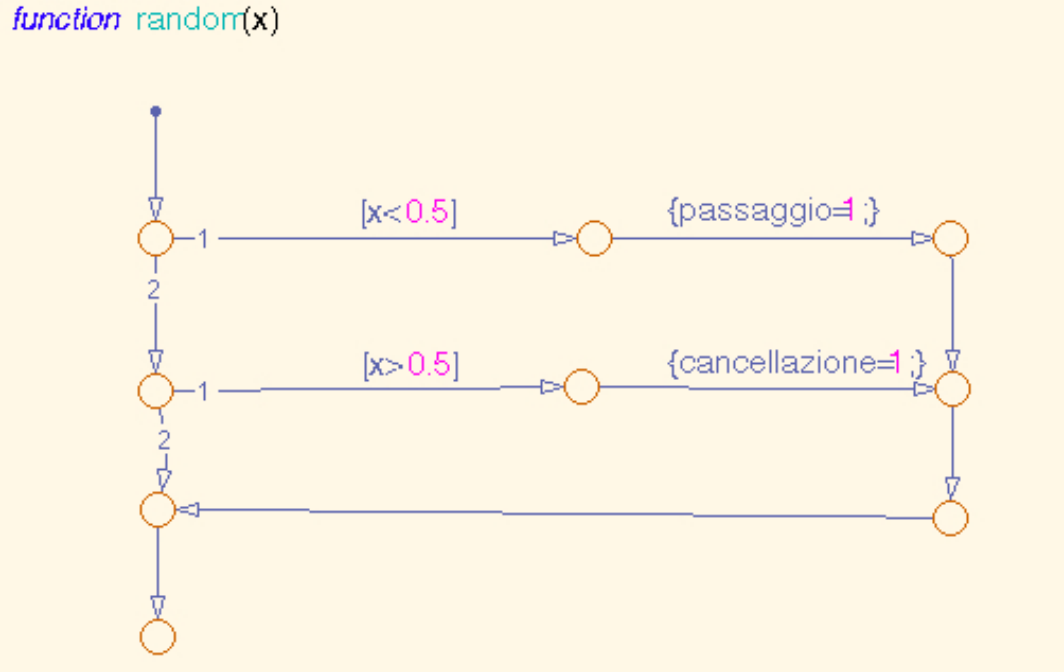

Figura 4.9: Funzione per il random

4.3.2 Passaggio

Per gestire il passaggio di un treno si é creato lo stato di *crossing* (figura 4.10) il quale, tramite la funzione *init passaggio* (figura 4.11), che prende in ingresso l'itinerario, da il via alla procedura di occupazione dei CDB e degli scambi coinvolti nell'itinerario.

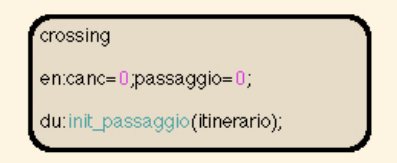

Figura 4.10: Stato per il passaggio

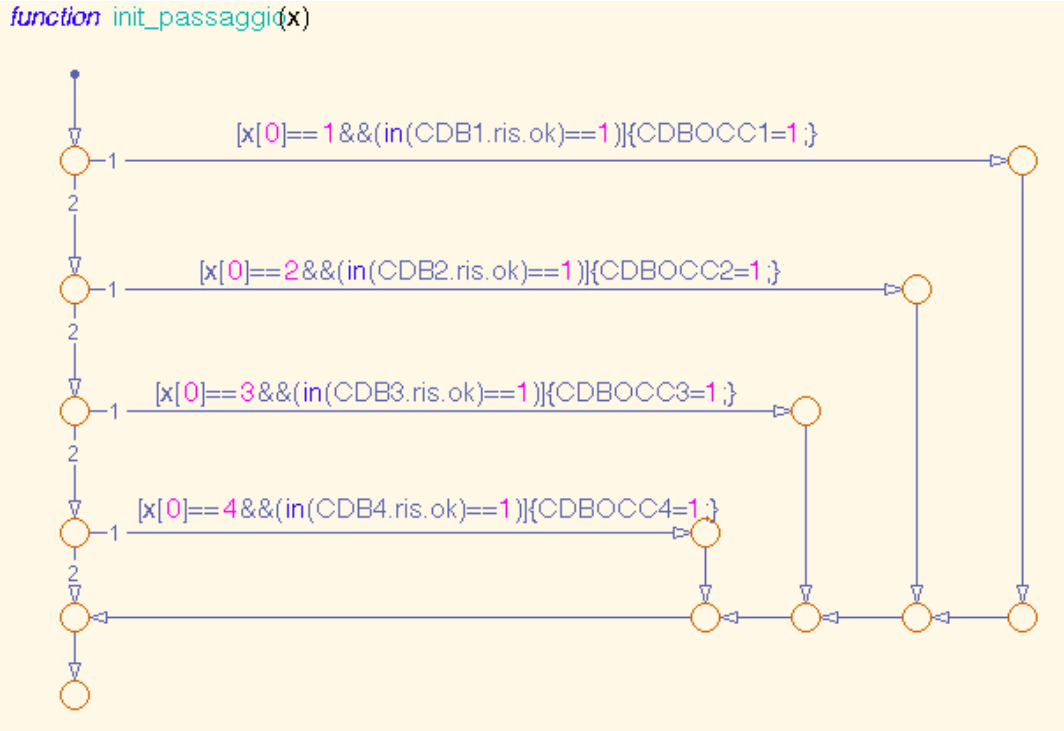

Figura 4.11: Funzione per il passaggio

4.3.3 Cancellazione

Per gestire la cancellazione si é creato lo stato di *cancellazione* (figura 4.12) il quale, tramite la funzione *invioDeleteIT* (figura 4.13), che prende in ingresso l'itinerario, da il via alla procedura di cancellazione dei CDB e degli scambi coinvolti nell'itinerario.

Figura 4.12: Stato per la cancellazione

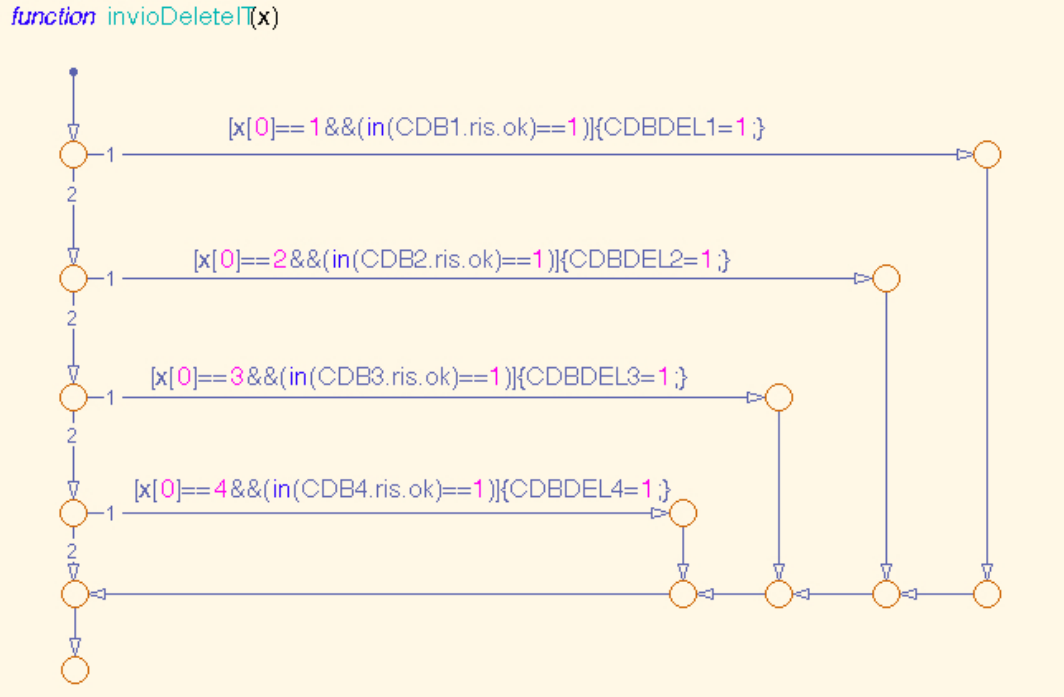

Figura 4.13: Funzione per la cancellazione

Capitolo 5

Simulazione del nostro modello

Dopo aver modellato la macchina a stati dichiarando ogni variabile e implementando le funzionalitá aggiuntive , si é passati alla fase di simulazione, per verificare che effettivamente il modello eseguisse quanto richiesto.

La simulazione consiste nell'osservare l'evoluzione dinamica del modello, con i bordi degli stati attivi che diventano blu. Nel presente capitolo illustreremo il passaggio da uno stato all'altro dei vari nodi sia nel caso dello scenario "*Passaggio del treno*" che in quello di "*Cancellazione Itinerario*".
Vengono quindi riportate alcune "istantanee" dello stato del sistema, che mostrano l'aspetto del modello in fasi diverse dell'esecuzione.

5.1 Prenotazione itinerario

La figura 5.1 mostra che é stata effettuata la prenotazione dell'itinerario 1-4, infatti il CDB1 ,lo scambio A ed il CDB4 risultano essere nello stato *riservato*.

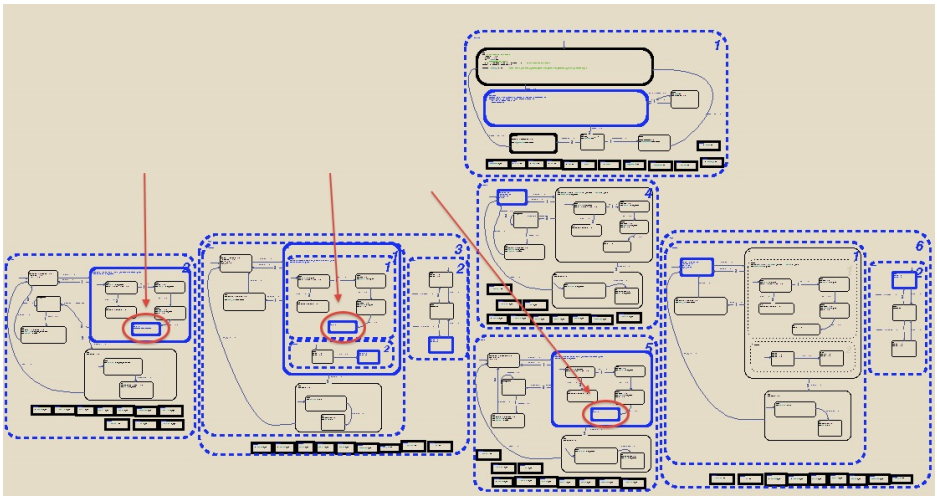

Figura 5.1: Itinerario 1-4 riservato

5.2 Passaggio treno

Nelle figure seguenti, é invece riportata la simulazione del passaggio di un treno. La figura 5.2 mostra come nel master é stato selezionato il ramo che simula l'operazione di passaggio.

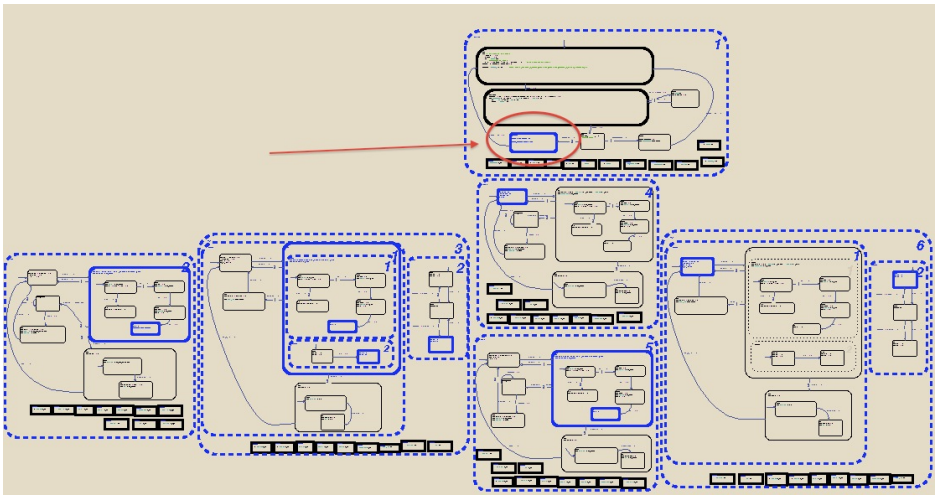

Figura 5.2: Selezione passaggio nel Master

Le figure 5.3 e 5.4 mostrano prima la fase in cui il CDB1 e lo scambio A si trovano nello stato di *occupato* ad indicare la presenza fisica del treno; poi la fase in cui ad essere *occupato* é il CDB4. Si puó notare come quando é occupato un nodo intermedio (non primo) di un itnerario, tutti i nodi precedenti tornano ad essere settati nello stato di *libero* in modo da permettere la prenotazione di un nuovo itinerario.

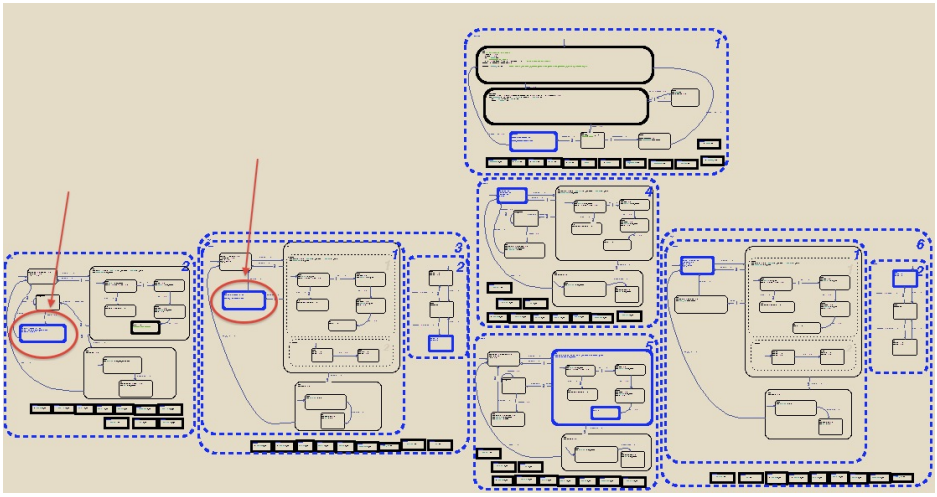

Figura 5.3: Prima fase di passaggio

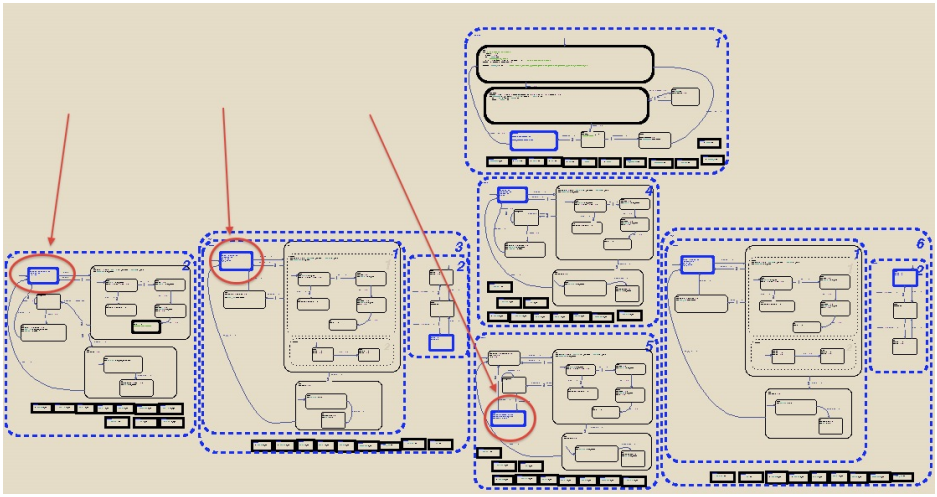

Figura 5.4: Seconda fase di passaggio

5.3 Cancellazione itinerario

Viene adesso riportata la simulazione riguardante la *cancellazione* di un itinerario. In questo contesto l'itinerario é 1-4 .
La figura 5.5 mostra il settaggio nel Master nel caso della cancellazione (viene attivato uno stato apposito che gestisce le operazioni)

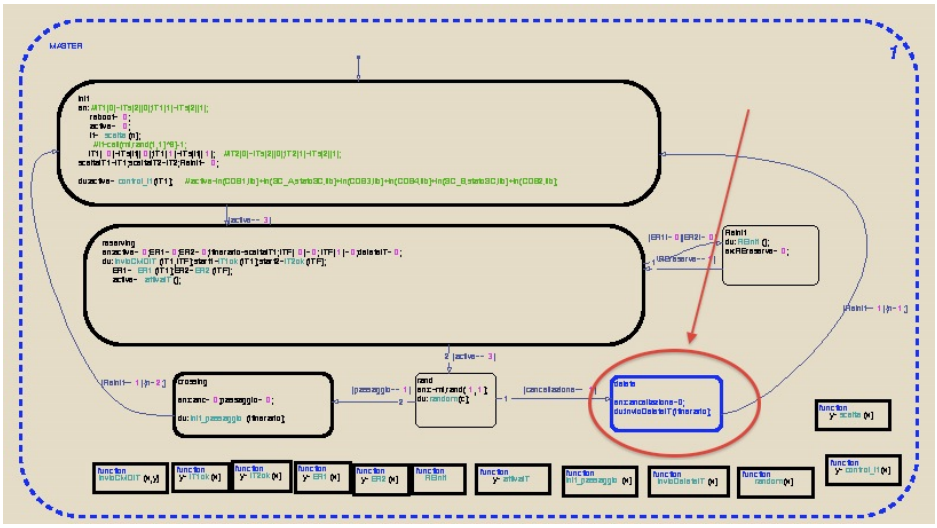

Figura 5.5: Settaggio del Master

Il Master provvede quindi ad azionare un meccanismo molto simile a quello usato per la prenotazione di un itinerario e quindi basato su *COMMIT* ed *AKNOWLEDGE*. La prima fase di questo processo prevede l'attivazione, per ogni CDB e scambio coinvolti, di uno stato apposito. (figura 5.6)

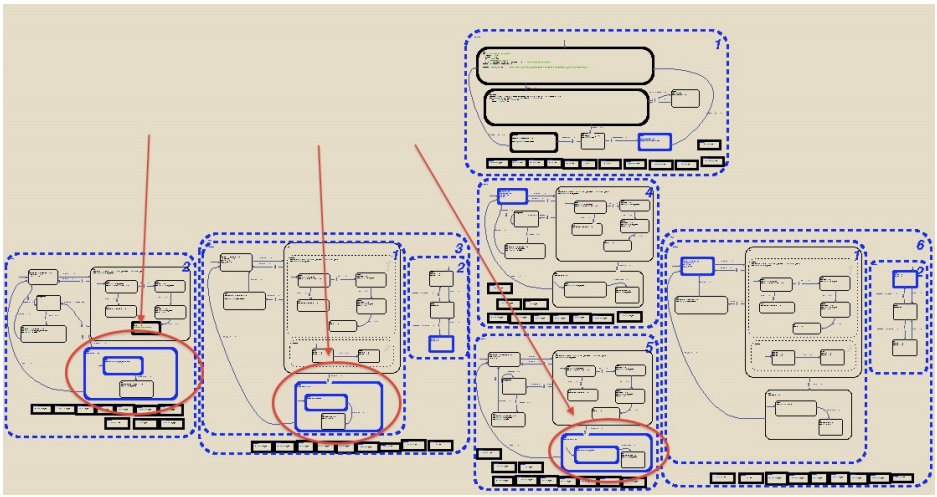

Figura 5.6: Settaggio nodi

Terminata la fase di commit ed aknowledge , inizia la liberazione dei CDB e degli SCAMBI. In questo caso prima si libera il CDB1 che quindi torna a *libero* (figura 5.7) , successivamente viene liberato lo SCAMBIO A (figura 5.8) e per ultimo viene liberato il CDB4 (figura 5.9)

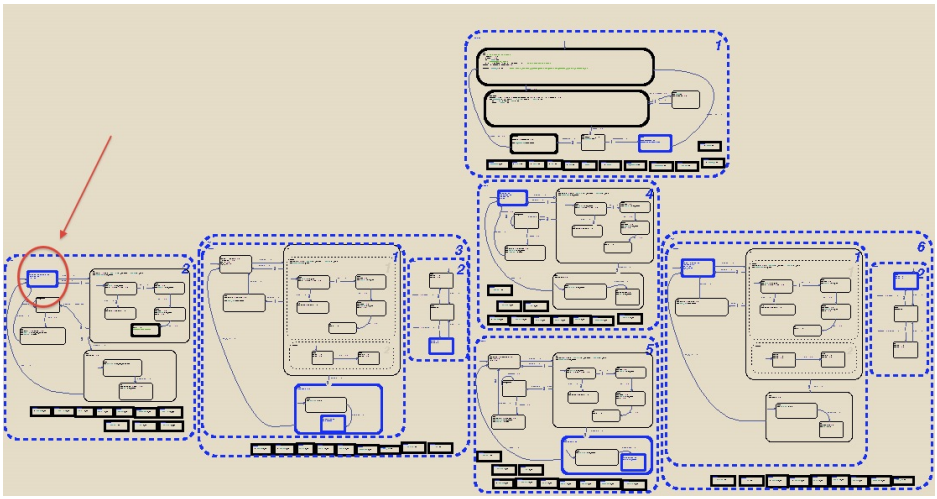

Figura 5.7: Liberazione CDB1

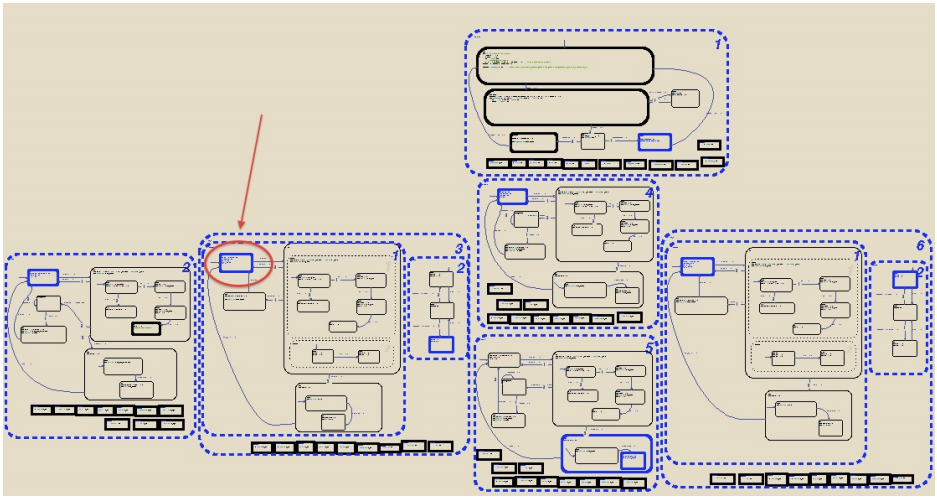

Figura 5.8: Liberazione SCAMBIO A

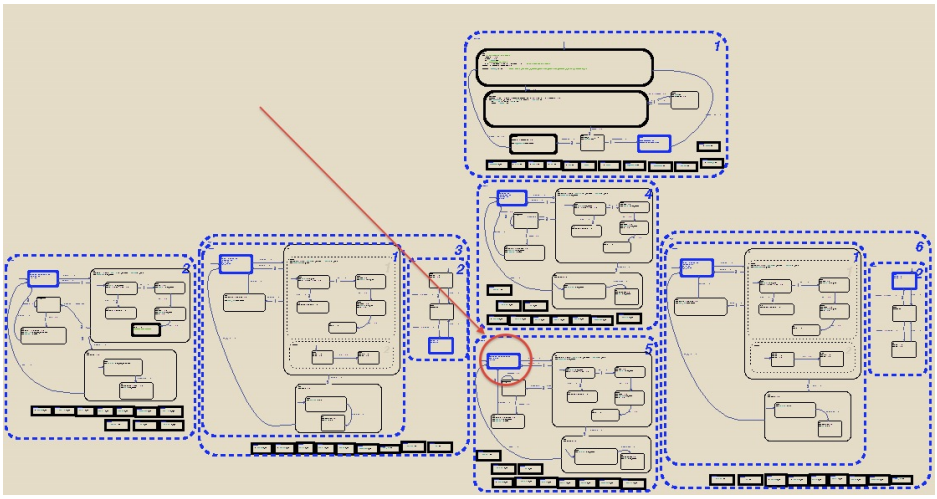

Figura 5.9: Liberazione CDB4

Terminata l'ultima liberazione il sistema é di nuovo pronto per ricevere altre prenotazioni di itinerario.

Conclusioni

L'elaborato risultante alla fine di questa sostanziosa modifica si presenta come un raffinamento del precedente lavoro coprendo diversi punti implementativi lasciati in sospeso.
Tuttavia, rimanendo legato strutturalmente al vecchio elaborato presenta alcuni punti che potrebbero essere migliorati in un lavoro futuro.
Si potrebbe tentare di cambiare la natura del modello al fine di verificare se un approccio diverso potrebbe portare a semplificazioni nella modellazione stessa, introducendo l'impiego di una chart per ogni nodo della stazione e cercando di salvare quanto piú possibile dell'implementazione giá sviluppata tenendo comunque presente che la principale complicazione di tale approccio sarebbe la numerosa presenza di collegamenti tra i chart in Simulink e la conseguente problematica della definizione delle variabili d'ingresso e d'uscita.