# Sommario

# 1- Introduzione

Obbiettivo dell'elaborato è di modellare un sistema di azioni concorrenti svolte da un insieme di robot.
Il modello sarà poi verificato e simulato con apposite tecniche.
Il contesto in cui i robot agiscono è una fattoria divisa in zone diverse.
Tali zone sono raggiungibili mediante l'utilizzo di un ponte. Solo un robot alla volta può passare per il ponte (mutual exclusion, safety). Il sistema è concepito in un ambito distribuito, dove ogni robot conosce tutte le informazioni riguardo se stesso e non vi è nessun elemento di centralizzazione di dati. Ogni robot esegue le azioni basandosi sul proprio clock e risultano asincroni tra di loro.

# 3- MagicDraw e note tecniche principali

MagicDraw è uno strumento di modellazione visiva UML, SysML. Questo consente l'integrazione di vari plug-in (Cameo Simulation toolkit, Alf, ecc..) con funzionalità e logiche integrabili ai linguaggi di modellazione prima citati.
I plug-in principalmente utilizzati sono stati "Cameo Simulation toolkit" e "Alf".
Il primo offre la possibilità di simulare visivamente il flusso di esecuzione e cambi di stato di un diagramma di macchina a stati.
Il secondo permette di implementare un comportamento e dare una semantica alla macchina attraverso l'utilizzo di codice.

I 2 strumenti di modellazione principalmente utilizzati sono il diagramma delle classi e della macchina a stati.
Il primo definisce il dominio e le entità in gioco, mentre il secondo rappresenta l'evoluzione del comportamento nel tempo.
Il diagramma delle classi oltre a poter specificare le classi con i rispettivi attributi da la possibilità di editare qualità più specifiche. Ad esempio, la possibilità di collegare le varie entità mediante associazioni. Ciò si ripercuoterà anche sul comportamento implementabile della macchina a stati. Per esempio, nel progetto in questione si osserva l'associazione di aggregazione tra la classe "RobotContainer" e "Robot". Inoltre, se viene specificato un nome alla associazione e una molteplicità, verrà creato automaticamente il concetto di lista di oggetti (robots).
Tale lista sarà utilizzabile, successivamente, attraverso il linguaggio "Alf ".

In generale, ogni attività, elemento, associazione che segue la logica UML ha un riscontro concreto sul sistema.

La macchina a stati permette di utilizzare elementi quali: lo stato, le transizioni, gli stati finali, le giunzioni ecc...
Nell'immagine si osservano tutti gli elementi utilizzabili.
Ogni elemento possiede proprietà e comportamenti specifici come verrà spiegato in seguito.

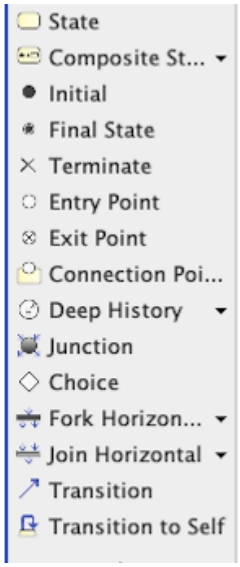


**Fig 1.**

Tutte le entità del sistema, che esse siano diagrammi delle classi o diagrammi di macchine a stati, hanno un rapporto di conoscenza ramificato: ad esempio la macchina a stati conosce la classe relativa da cui prende i dati e i suoi attributi se si trova in un ramo di questa.

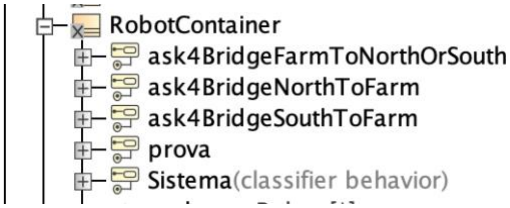


**Fig 2.**

Il comportamento della macchina a stati viene definito attraverso un "Behavior type".
Questi tipi di comportamento sono i seguenti:

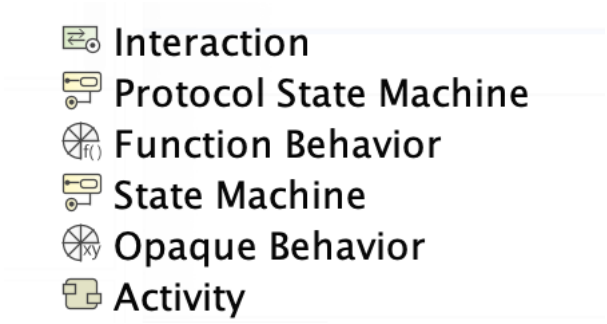



**Fig 3.**

Il comportamento viene specificato su una transizione o su un “entry Point”, un “do activity Point” o un “Exit Point” relativi ad uno stato.
L’” Opaque Behavior” è il comportamento standard che Alf necessita per eseguire. Vi è la possibilità di associare un diagramma di attività o di interazione al comportamento delle transizioni e degli stati. In tal caso vi sarà la possibilità di implementare gli algoritmi Alf all’interno del diagramma di attività, stesso concetto vale per il diagramma di interazione.

Altra caratteristica importante di MagicDraw è la possibilità di esportare il progetto in molti formati differenti, così da essere usati in strumenti di modellazione differenti.


**Fig 4.**

# 5- Modellazione

Obbiettivo del modello è stato di riprodurre, attraverso gli strumenti messi a disposizione da MagicDraw, uno scenario più simile possibile al caso reale di robot comunicanti tra loro.
Come idea principale è stato supposto che ogni robot sia autonomo quanto più possibile e le azioni di comunicazione siano ridotte al minimo.
Inoltre, altre idee guida durante la modellazione sono state:
La mutua esclusione, la "livness", "safety", la "fairness" e clock autonomi di ogni robot, oltre che i rimanenti requisiti.
Meno importanza è stata data invece al requisito per cui il numero di robot non sia predefinito o statico, ma variabile.
Ciò è dovuto al fatto che il modello è stato costruito gradualmente e in una fase finale avrebbe richiesto un cambiamento cospicuo al modello.
Inoltre, alcune funzionalità quali il "Fork" e "Join"(Fig 1.) non sono disponibili per "Cameo Simulation Toolkit", ma solo per MagicDraw.

## 5.1- Diagramma delle classi

Le classi utilizzate nel progetto sono "Robot" e "RobotContainer" (Fig 5.).

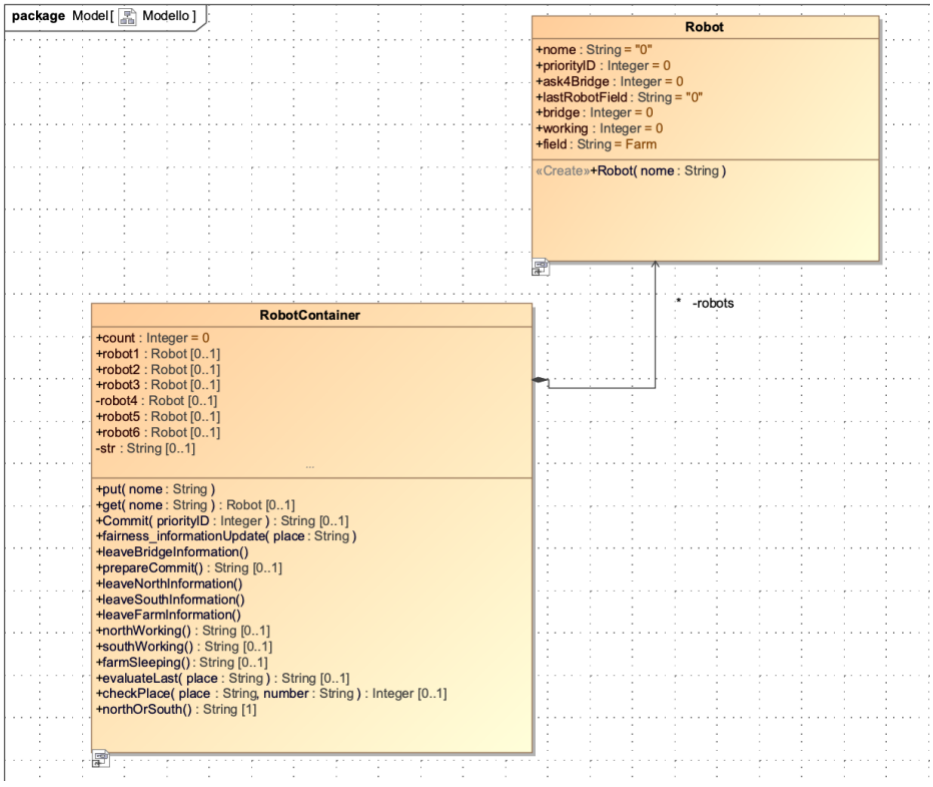



**Fig 5.**Nota l'attributo field di tipo String, inizializzato con valore "Farm" non necessita delle virgolette nel diagramma delle classi.

Come già accennato, le classi sono collegate mediante associazione di aggregazione, ciò è importante per la simulazione.
La simulazione della macchina a stati parte da una classe (in questo caso "RobotContainer") e procede oltre ricostruendo le informazioni della semantica secondo lo standard UML. Questo fa si che la classe da cui parte l'esecuzione debba conoscere tutte le altre entità del dominio attraverso tipi di associazioni diverse. Concetto simile riguarda la visibilità dei metodi e attributi. Ad esempio, affinché la classe RobotContainer possa conoscere gli attributi dei vari robot è necessario impostare quest'ultimi come pubblici.
Per poter istanziare un oggetto è necessario che tale classe disponga del metodo pubblico costruttore, con stereotipo 《Create》.

Ultima nota tecnica va al valore di ritorno di alcuni metodi, la cui molteplicità è [0..1]. Ciò risulta necessario per il corretto funzionamento del codice "Alf".

## *5.2- Macchina a stati*

La macchina a stati è composta da 4 principali stati: "FarmToNorth", "Bridge", "ToFarm" e "FarmToSouth". Ognuno di questi ha incapsulati al suo interno ulteriori stati che vanno a comporre il comporre il comportamento.
Questi 4 macro stati lavorano in maniera tra loro sequenziale. Questa scelta è stata fatta per 2 motivi:
Rendere più facilmente visibile il comportamento dei robot durante l'esecuzione e per ottenere la "Fairness".
La "Fairness", infatti, si concretizza quando due robot diversi non passano mai per due volte consecutive dallo stesso percorso. Le guardie uscenti, insieme ad alcuni algoritmi, gestiranno il corretto funzionamento di tale requisito.

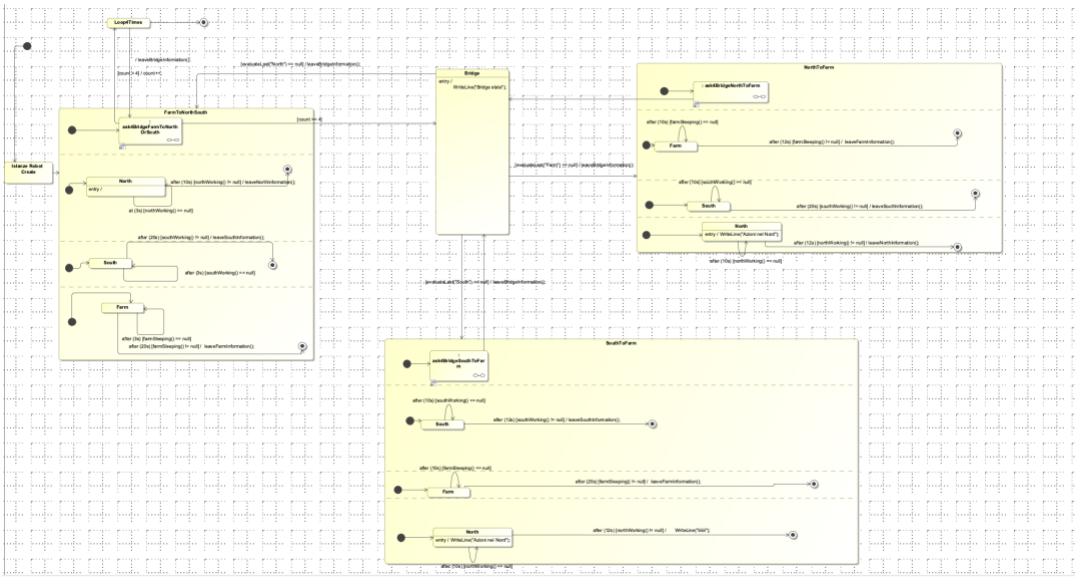


**Fig 6.** Nella figura rispettivamente: “FarmToNorthSouth” a sinistra, “Bridge” al centro, “SouthToFarm” in basso e “NorthToFarm” a destra.

Nella figura 6 si osserva il modello generale della macchina. Quando viene eseguita una simulazione il sistema inizia dallo stato iniziale, proseguendo come specificato dalle transizioni con le rispettive guardie.

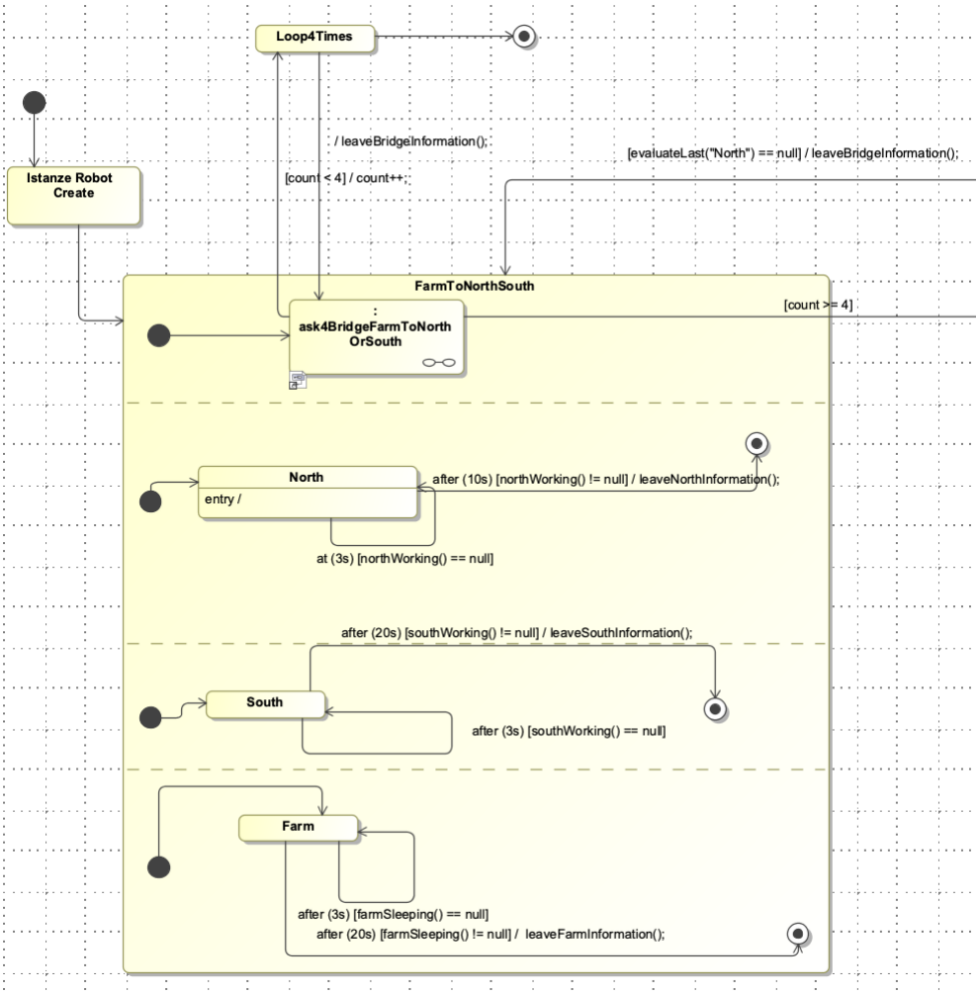



**Fig 7.**

Nella figura 7 è possibile osservare uno zoom dell'immagine precedente.
Partendo dallo stato iniziale più a sinistra di tutti, si giunge nello stato "Istanze Robot Create". Tale stato specifica un tipo di entry che esegue la funzione in figura 8.

put

```
1 robot = this.robots->select e (e.nome == nome)[1];
2 if(robot != null){
3 robot.nome = nome;
4      this.robots->add(robot);
5  WriteLine("messo nome "+nome);
6 }
7 else
8 {
9  robot = new Robot(nome);
10 WriteLine("Nuova istanza "+nome);
11    this.robots->add(robot);
12 }
13
```

**Fig 8.**

Scopo principale di tale funzione è riempire la lista "robots" di singole istanze "Robot". L'importanza di questa azione risiede, inoltre, nel fatto di sfruttare al meglio il plug-in di debug (Cameo Simulation Toolkit) per osservare la variazione degli attributi e degli stati nel tempo.

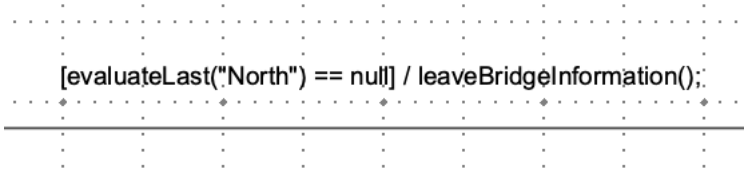
Procedendo oltre nel flusso di esecuzione si giunge allo stato "FarmToNorth".
Come si osserva, tale stato è uno stato ortogonale con 4 regioni al suo interno. La disposizione degli stati, delle transizioni e degli oggetti risulta fondamentale per ottenere uno specifico comportamento (concorrente).
Ad esempio, il "final state" posto all'interno della regione ortogonale ha un comportamento differente rispetto al caso in cui questo sia posto fuori.
Nello specifico, se i tre flussi ("North", "South", "Farm") hanno concluso la rispettiva traccia, giungendo al final state, ma il flusso "ask4BridgeFarmToNorthOrSouth" non ha ancora completato la propria esecuzione, il sistema non procede allo stato "Bridge". Se invece, uno qualsiasi dei 3 "final state" in figura 7 fosse stato posto al di fuori dello stato ortogonale, allora sarebbe bastata la conclusione di un'unica traccia tra "North", "South" e "Farm" perché il sistema procedesse allo stato "Bridge".

La scelta della disposizione mostrata in figura è stata fatta con l'idea di simulare in maniera più reale possibile lo scenario del progetto e, come già detto, garantire la "Fairness". Nello specifico, le azioni di lavoro e richiesta del ponte devono essere svolte di continuo e parallelamente dai vari robot. L'azione di attraversare il ponte è l'unica che, invece, deve essere compiuta da un robot per volta.

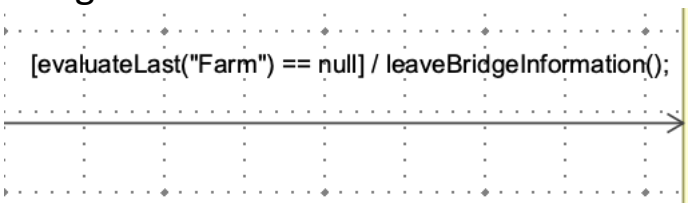
Per tale motivo il sistema della macchina a stati ha una struttura dove in ogni istante vi è la richiesta di ask4Bridge e parallelamente il lavoro (Nord, sud) o il riposo (Fattoria).
Gli unici istanti di tempo in cui i robot smettono di lavorare, riposare o richiedere l'accesso al ponte è l'arco temporale che va dall'uscita di un robot qualsiasi da uno dei 3 macro stati ortogonali alla conclusione di una transizione uscente dallo stato "Bridge". Tuttavia, essendo la sequenza di transizioni "entrata nel ponte >>attraversamento del ponte >> uscita dal ponte" priva di "trigger event" (after()), tali azioni avvengono in un tempo trascurabile rispetto al resto delle azioni.

Come già accennato, le guardie poste sulle transizioni uscenti dallo stato "Bridge" insieme all'algoritmo "northOrSouth()" gestiscono la "Fairness".
Per quanto riguarda le guardie poste sulle transizioni uscenti di "Bridge" si osservano i seguenti comportamenti:

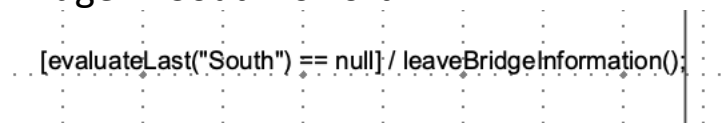
Bridge>>FarmToNorth




Bridge>> NorthToSouth




Bridge>> SouthToNorth




In tutti e tre i casi le guardie hanno lo scopo di valutare quale è stato l'ultimo robot ad attraversare il ponte.
In particolare viene valutato il valore di ritorno della funzione "evaluateLast(place : String)".

evaluateLast

```
1 return this.robots->select r (r.lastRobotField == place).nome[1];
```

**Fig 9.**

## 5.3- ask4Bridge

Come si osserva in figura (FarmToNorthSouth) vi è una "submachine": ask4BridgeFarmToNorthOrSouth. La "submachine" ha lo scopo di creare ordine oltre che aggiungere funzionalità per modellare comportamenti differenti.
La "submachine" può essere un'intera macchina a stati e deve essere scelta all'interno dell'albero dei file creati.
Nell'immagine 10 si osserva la prima parte del contenuto della "submachine" .

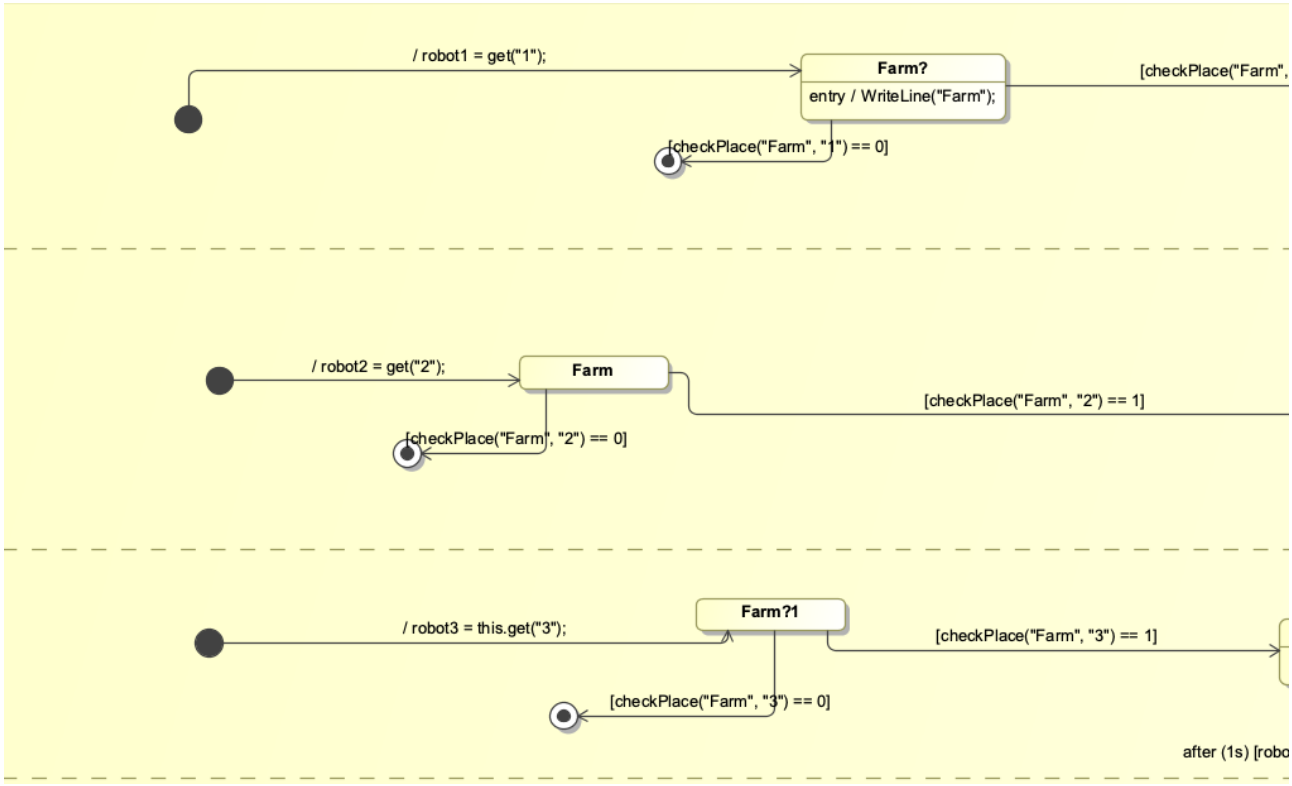



**Fig 10.**

Si tratta di un "Orthogonal State" contenente l'insieme degli stati e delle azioni di richiesta del ponte secondo il metodo 2PC.
Anche in questo caso ogni flusso parte da uno "Initial state". Sulla prima transizione, vengono impostate le variabili robot1, robot2, ecc…. attraverso il metodo this.get("1").

Il primo stato che si incontra è quello "Farm". In questo stato viene verificata la posizione geografica del robot attraverso il metodo "checkPlace(place, robotNumber)".

checkPlace

```
1 robot =  this.robots->select e (e.nome == number)[1];
2 WriteLine(robot.field ?? "null");
3 if(robot.field == place){ return 1; }
4 WriteLine("Robot"+robot.nome+" in Farm" ?? "No robot in Farm");
5 if(robot.field != place){ return 0; }
```

**Fig 11.** Nota: nel codice Alf soprastante l'argomento della funzione WriteLine contiene al suo interno i caratteri "??". Ciò sta ad indicare che qualora il valore della variabile da stampare sia null, stampi il valore alla sua destra e che stampi il valore alla sua sinistra in caso contrario. Tale concetto si estende anche ad altri casi come nella funzione "northOrSouth()" che sarà discussa in seguito.

Procedendo oltre nel flusso di esecuzione delle tracce parallele, si trova lo stato "Working?" il quale ha 2 transizioni uscenti.
Le guardie valutano la variabile che indica che tale robot sta attualmente lavorando nel Nord, Sud o dormendo in fattoria. Nel caso in cui stia lavorando, come si osserva nell'immagine, inizia un loop sullo stato "Working?".
Il flusso si blocca fin tanto che una delle azioni parallele: "LeaveNorthInformation()", "LeaveSouthInformation()" o "LeaveFarmInformation()" non si concretizza. La scelta di utilizzare una transizione che attua un loop su se stesso fin tanto che la guardia non risulti falsa, piuttosto che l'utilizzo di un "finalstate"( come nel caso precedente nello stato "Farm") è dovuta al fatto che in questo caso il robot può concludere di lavorare prima che uno degli altri robot prenda il ponte e quindi deve entrare subito in competizione con gli altri robot per la risorsa. Inoltre, in questo caso, non vi è la possibilità di finire in un caso di loop infinito, poiché come detto flussi paralleli sbloccherebbero la situazione per procedere oltre. Tale situazione non è sempre vera nel caso della valutazione dell'area geografica. Infatti, se nello stato "Farm?" fosse stato messo un loop e non un final state, il sistema avrebbe cominciato un ciclo eterno nel caso in cui nessun robot, in un certo istante t, si trovi nella fattoria.

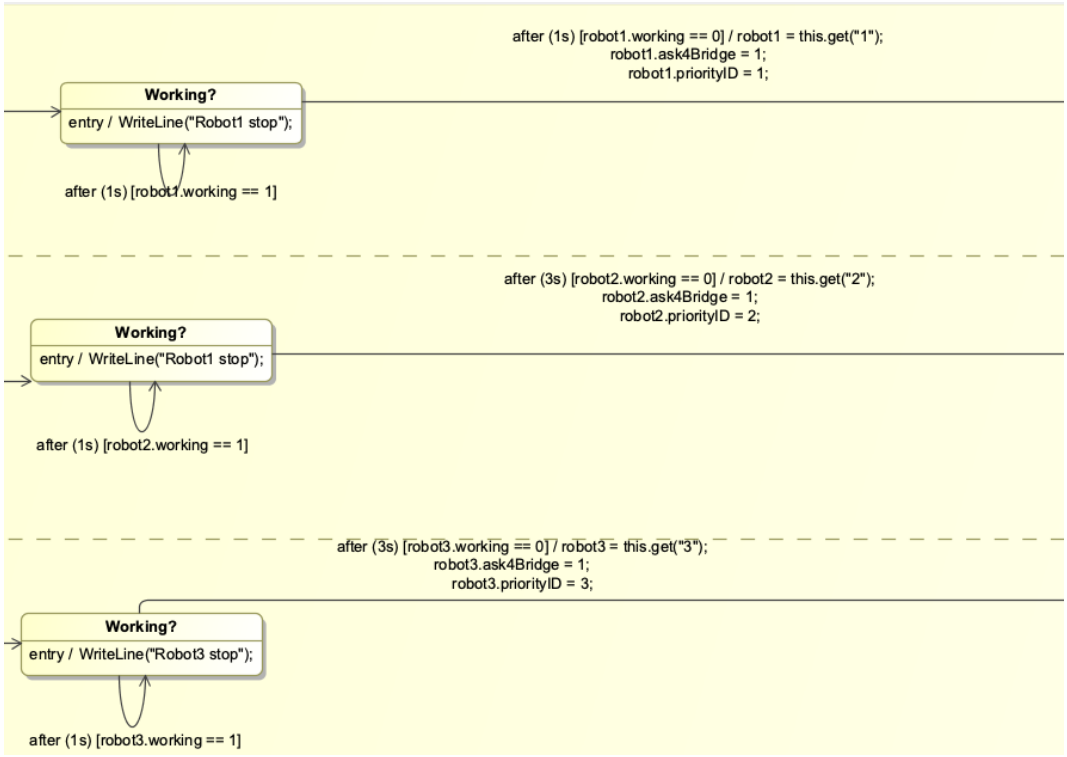



**Fig 12.**

Una volta superati i due stati “Farm?” e “Working?” i flussi paralleli iniziano la fase di 2PC. Tale fase è composta da 2 stati e 4 transizioni a esse associate.

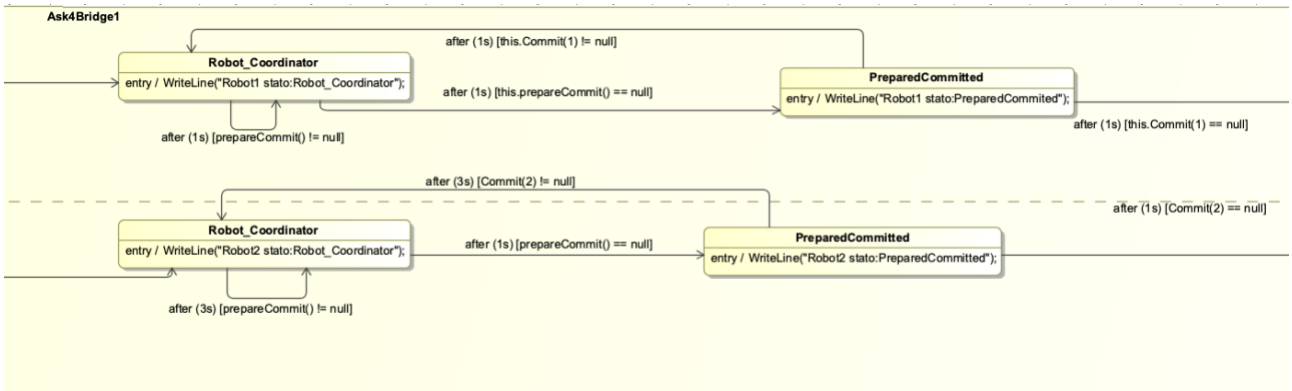



**Fig 13.**

Nel primo stato (“RobotCordinator”) vi è una transizione a se stesso che valuta la funzione “prepareCommit()”:

prepareCommit1

```
1 str = this.robots->select r (r.bridge == 1).nome[1];
2 WriteLine(str ?? "null");
3 return str;
4
5
```

**Fig 14.**

Come si osserva nel codice della figura la funzione da come valore di ritorno null o il nome del robot che è nel ponte e ha l'attributo field settato a "North". Tale condizione si verifica quando un robot proveniente dalla fattoria ha acquisito il ponte fino a che questo non viene attraversata e viene eseguita la funzione leaveBridgeInformation().

Lo stato successivo "preparedCommitted" svolge un lavoro simile a quanto fatto nello stato precedente.

Commit

```
1 return this.robots->select r (r.priorityID < priorityID && r.ask4Bridge == 1  || r.bridge == 1).nome[1];
2
```

**Fig 15.**

In questo caso viene verificata la presenza sul ponte di un robot e altre 2 condizioni. La seconda verifica dello stato del ponte risulta importante dato il contesto di parallelismo. Il doppio controllo aggiunge ridondanza nella protezione della variabile bridge rendendo meno probabile un accesso simultaneo tra più robot nel ponte.
Inoltre, la funzione valuta anche la priorità di un robot e la variabile ask4Brige. In particolare la variabile ask4Bridge deve essere uguale a 1 quando viene valutata la priorità al fine di non valutare le priorità di robot che non sono in competizione per il ponte.

Ultimo stato prima di giungere al "final state" è lo stato "Committed" (Figura 16).
Tale stato ha lo scopo di cambiare i valori alle variabili in maniera coerente al contesto. Come prima cosa viene posta la variabile bridge a 1, ask4Bridge a 0 e working a 1.

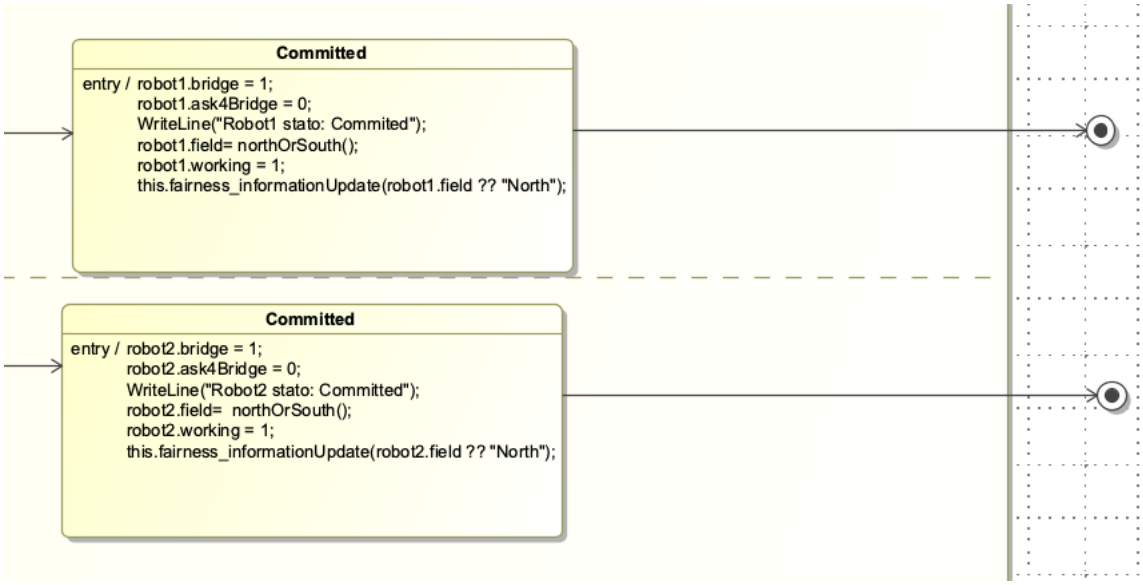



**Fig 16.**

La variabile field viene assegnata attraverso una funzione “northOrSouth()”. Infatti, se negli altri due blocchi equivalenti (“NorthToFarm” e “SouthToFarm”) l’unica direzione di uscita è la fattoria, in questo caso il robot può andare verso Nord o verso Sud. Nell’immagine 17 è possibile osservare il codice relativo a tale funzione:

northOrSourh

```
1 Integer robotInNorth = 0;
2 for(Robot in robots){
3             r = Robot;
4      if(r.field == "North"){
5
6      robotInNorth++;
7 }
8 }
9
10 if(robotInNorth >= 3) {return "South" ?? "null";}
11 else {return "North" ?? "null";}
```

**Fig 17.**

Ultima azione svolta è di chiamare il metodo “Fairness_informationUpdate(Place)”. Questa funzione aggiornerà i dati dei vari robot così che, come già accennato, quando sarà successivamente valutata la funzione “evaluateLast()”, eseguano le giuste transizioni.
Di seguito il codice della funzione:

```
fairness_informationUpdate
1 for(Robot in robots){
2         r = Robot;
3         r.lastRobotField = place;
4 }
5
```

**Fig 18.** Nota: Nel codice si osserva un altro caso in cui per passare i parametri vengono usati i due caratteri "??" per specificare l'argomento della funzione nel caso la variabile robot.field sia null.

Una volta superato lo stato "Committed" vi è un "final-state". Come spiegato precedentemente, poiché questo si trova fuori dal blocco ortogonale, la conclusione di una sola delle tracce parallele fa concludere l'intero stato ortogonale.

I due blocchi rimanenti("SouthToFarm" e "NorthToFarm") si comportano in maniera equivalente a quanto descritto fin'ora riguardo "FarmToNorthSouth". La differenza sta nelle guardie attribuite al luogo (Nord, Sud, Fattoria) che controlleranno l'eventuale presenza del robot nel Nord o Sud, anziché nella fattoria o anche il valore field attribuito all'uscita dallo stato "Committed".

Ultima nota riguardo la modellazione della macchina a stati è la seguente traccia che si ripete 4 volte.
Tale scelta è dovuta al fatto che il modello senza questi 4 cicli porterebbe il robot con priorità 1 ad acquisire in continuazione il ponte. In questa maniera vengono distribuiti invece alcuni robot tra Sud, Nord e fattoria.
In tale maniera robot con priorità basse si trovano spesso in zone differenti e quindi i robot a priorità alta hanno più possibilità di acquisire il ponte.

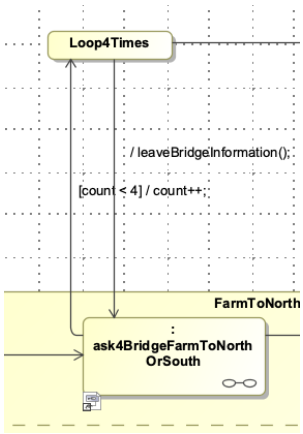



**Fig 19.**

# 6- Simulazione

Come già accennato le principali vie per simulare il comportamento sono state l'utilizzo di "Cameo simulation toolkit" e del terminale (compatibile con il linguaggio "Alf").
Prima di utilizzare Cameo Simulation toolkit per eseguire la macchina a stati, può risultare utile cambiare i colori (Option>>Environmnet>>Simulation), così da osservare meglio ciò che interessa e poter associare colori diversi a casi diversi: stati già visitati, ultimi stato visitato, stato attualmente visitato.
Nell'immagine si osserva un flusso di esecuzione della macchina a stati.

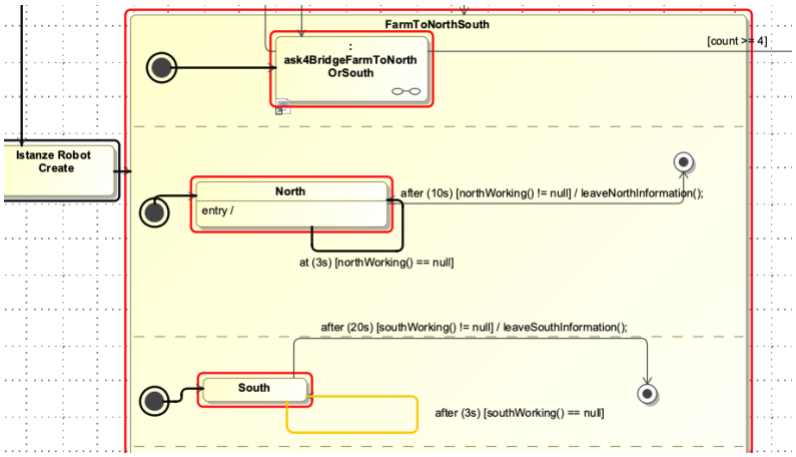



**Fig 20.**

Per quanto riguarda i test sul sistema, così da valutare i valori della Availability e Reliability va premesso che il sistema di temporizzazione dell'esecuzione del sistema non è lineare: aumentando il clock di esecuzione di azioni per secondo, il tempo di esecuzione non diminuisce in maniera proporzionale. Ciò è probabilmente causato dal tempo di esecuzione di alcune funzioni (Alf) maggiore rispetto a altre. Questo rende più difficile l'utilizzo dello strumento di Cameo per valutare la Reliability, in quanto il valore delle variabili cambia o troppo velocemente o troppo lentamente (a seconda del clock impostato), ma difficilmente le azioni si svolgono con tempi simili. Fatta tale premessa, il sistema sembra avere un comportamento coerente con le aspettative di Fairness, Safety. Per quanto riguarda l'availability e quindi la non interruzione del servizio, il sistema testato per lungo tempo sembra procedere senza terminare o andare in "Deadlock".

# *7-Particolarità tecniche*

Altra caratteristica importante sta nelle dipendenze e associazioni che ogni elemento dispone. Ad esempio, copiare e incollare un qualsiasi elemento, come ad esempio una classe o uno stato, funziona in maniera particolare: se all'interno di uno stesso diagramma si copia e incolla una classe, allora, ogni modifica a una delle due copie si ripercuoterà sull'altra. Soluzione a ciò è di copiare una classe da un diagramma che non si trovi nello stesso sottoalbero del diagramma in cui vi è il desidero copiare l'elemento. Cosi facendo i due elementi sono tra loro disgiunti e la modifica di uno non si ripercuote nell'altro.

Altra particolarità di MagicDraw e in particolare di Alf sta nel generare "Warning" quali "OpaqueBehavior Expression is ignored", risolvibili eliminando gli stati che danno tali "Warning", copiandoli in un diagramma di un differente sottoalbero, per poi ricopiarlo nella posizione originale.

Altra particolarità su cui è importante porre l'attenzione è il fatto che se uno stato si trova all'interno di un "Orthogonal State" e appare visivamente sovrapposto a questo, non necessariamente lo è logicamente e con le rispettive dipendenze. Per risolvere tale problema la soluzione è di trascinare lo stato in una direzione qualsiasi rimanendo sempre all'interno della regione. In tale maniera, se precedentemente non via era una dipendenza logica tra i due elementi, apparirà una finestra che chiederà se spostare le dipendenze dello stato all'interno del macro stato ortogonale.

Altra nota riguarda il fatto che le guardie possono confrontare solo variabili intere e booleane, ma non di tipo String. Per ovviare a ciò una soluzione è implementare un metodo separato che verifichi se due stringhe sono uguali e ritorni un valore (0 o 1) in caso di uguaglianza o disuguaglianza.

Infine, una nota riguardo il linguaggio Alf come plug-in di MagicDraw.
Tale linguaggio per essere compilato correttamente necessita, oltre che la sintassi sia rispettata e le logiche del linguaggio siano congruenti con l'algoritmo scritto, che gli spazzi siano utilizzati correttamente. Ciò in particolare si osserva nella valutazione delle guardie dove, ad esempio, la condizione "a >  b" porta il sistema a terminare. La condizione deve essere scritta nella seguente maniera: "a > b". Non vi devono essere più spazzi che il necessario.