****

**Scuola di Ingegneria**

**Corso di Laurea Triennale in**

**Ingegneria Informatica**

**Dipartimento di**

**Ingegneria dell’Informazione**

**Modellatore e Analizzatore di Stochastic Markovian Fault Tree**

Mistretta Marco

Matricola 7005065

Casciaro Emanuele

Matricola 7012347

Anno accademico 2020-2021

Sommario

**1. INTRODUZIONE** x

1.1 Brevi accenni teorici2

1.2 Motivazione e contenuti3

**2. ANALISI DEI REQUISITI** x

2.1 Use Case Diagram5

The Modeler6

The Analyst7

**3. PROGETTAZIONE** x

3.1 Event9

3.2 Simulatorx

3.3 Plotterx

3.4 DataCenterx

3.5 Masterx

**4. ANALISI DEI REQUISITI** x

Use Case Diagramx

The Modelerx

**5. INTRODUZIONE** x

1.1 Brevi accenni teoricix

1.2 Motivazione e contenutix

**6. ANALISI DEI REQUISITIx**

Use Case Diagramx

The Modelerx

**7. INTRODUZIONEx**

1.1 Brevi accenni teoricix

1.2 Motivazione e contenutix

**8. ANALISI DEI REQUISITIx**

Use Case Diagramx

The Modelerx

1. **INTRODUZIONE**
   1. **Brevi Accenni Teorici**

Uno Stochastic Markovian Fault Tree (SMFT) è un albero che rappresenta le condizioni sotto cui i fallimenti di un insieme di componenti si propagano, dalle foglie alla radice, realizzando le possibili cause di fallimento del sistema. Tale modellizzazione ci permette dunque di valutare il funzionamento dell’intera macchina osservando il solo stato di funzionamento del nodo radiche, che prende il nome di Top Event. Ciascun nodo prende il nome generico di Event; ciascuna foglia, detta Basic Event, rappresenta lo stato di fallimento di una singola componente del sistema; i nodi intermedi, detti Gates, rappresentano la condizione di propagazione di guasto all’interno dell’albero. Questi ultimi si classificano in due sotto-categorie: gates statici e gates dinamici, a seconda che la funzione di propagazione dipenda o meno dall’ordine in cui le componenti figlie abbiamo propagato il guasto. Un Event (Basic Event o Intermediate Event che sia) può presentarsi in ingresso a più Intermediate Event simultaneamente, questa condizione prende il nome di repeated Event.

**1.2 Motivazioni e Contenuti**

L’obbiettivo di questo progetto era quello di modellizzare uno SMFT generico nel quale: i Basic Event alternano stati di funzionamento e malfunzionamento secondo un semplice modello di Gilber Elliot (i.e. ad intertempo esponenziale), possono essere presenti repeated Event e i Gate possono essere di natura sia statica che dinamica. Tutto ciò ai fini di: simulare l’esecuzione per un dato stato di inizializzazione, e poter ripetere più volte la simulazione in condizioni atte a poter calcolare gli indici di confidenza sul valore medio dello stato di funzionamento del sistema (e.g. tramite test di Student) e sperimentare la natura ergodica del modello.

1. **ANALISI DEI REQUISITI**

La prima fase di progettazione si è basata sulla stesura dello Use Case Diagram (figura2.1)

Inserisci use case qui

**2.1 Use Case Diagram**

Sono stati identificati due Attori fondamentali: Modeler e Analyst.

Nell’ottica di come è stato ideato il programma, Modeler rappresenta l’utente con esperienza di modellizzazione di SMFT che si occuperà appunto, tramite il programma, della fase di modellizzazione dell’intera struttura. Fornire un’interfaccia che permetteva tale libertà di modellazione non era però tra gli obbiettivi del programma, per questo motivo Modeler appartiene al dominio logico del programma stesso, ed in particolare, non ne sta al di fuori.

Analyst rappresenta invece un utente con sufficiente conoscenze di analisi e studio di SMFT, il quale, tramite il programma, sarà in grado di valuatare la natura ergodica del sistema e calcolare gli indici di confidenza sul valore medio, istante per istante, della reliability del sistema.

Per maggiori informazioni, analizziamo il diagramma più nel dettaglio.

* **The Modeler**

Come abbiamo detto, The Modeler ha il principale caso d’uso di modelizzare l’intera struttura. Per fare questo può quindì definire nuovi Basic Event e sulla base dei Basic Event già definiti può dunque andare a definire nuovi Intermediate Event (notare che non esistono Intermediate Event senza figli, un Event senza figli è necessariamente una foglia, dunque è un Basic Event per mera definizione).

La definizione di un nuovo Basic Event può essere effettuata in modo, per così dire, manuale oppure randomico. In modo manuale scegliendo dunque quali valori dare rispettivamente alle due caratteristiche delle due esponenziali che caratterizzano l’intertempo di un Basic Event che segue un modello di funzionamento-malfunzionamento di Gibert-Elliot, e scegliendo in quale dei due stati (funzionamento o malfunzionamento) questo debba partire. In modo randomico, lasciando ad un generatore pseudo-casuale l’onore di scegliere tali valori. Avviso di notazione: si è scelto di chiamare lambda la caratteristica dell’esponenziale che rappresenta la probabilità di passare dallo stato di funzionamento a quello di malfunzionamento, e mu la rispettiva controparte.

La definizione di un nuovo Intermediate Event (come quella del Top Event stesso, che altro non è che uno speciale Intermediate Event senza padri) come la precedente può essere eseguita in modo manuale o randomico. In modo manuale scegliendo quale porta logica, statica o dinamica che sia, si desidera realizzare, in modo randomico lasciando ad un medesimo generatore pseudo-casuale l’onore di tale scelta.

* **The Analyst**

The Analyst ha principalmente bisogno di simulare più e più volte l’esecuzione del SMFT, a partire da inizializzazioni atte a raggiungere i vari scopi rappresentati dai suoi casi d’uso. Per valutare la natura ergodica ha bisogno infatti di simulare più volte il sistema a partire da stati iniziali differenti e verificare, in via d’approssimazione, che dopo un “tempo sufficientemente lungo” (ci soffermeremo successivamente sulla definizione di tempo sufficientemente lungo), tutte le simulazioni si aggirino intorno allo stesso valore percentuale di reliability. Per il calcolo degli indici di confidenza, invece, ha semplicemente di simulare più e più volte un sistema ma a partire dal medesimo stato di inizializzaione (funzionamento del sistema o malfunzionamento del sistema che sia) per fornire in modo sperimentale indici che aiutino nella previsione e stima del comportamento di macchine che presentino la stessa struttura e stesso stato di partenza.

1. **PROGETTAZIONE**

In prima fase ci siamo concentrati sulla progettazione del Class Diagram (UML). Di questo ultimo è possibile visionare uno storica che raccoglie tutte le modifiche apportate dalla prima versione alla versione definitiva (figura 3.1)

**3.1 Event**

* **Composite**

In prima battuta ci siamo concentrati sulle scelte implementative che coinvolgevano la modellazione dell’albero stesso , ossia la modellizzazione di un Directed Acyclic Graph (DAG). Data la natura del sistema, dove l’utente finale non è tenuto a sapere com’è formato l’albero, ma deve interfacciarsi con esso tramite il Top Event, una soluzione adatta a raggiungere tale obbiettivo era senza dubbio l’implementazione tramite design Pattern Composite, nella sua più totale versione ed implementazione in assenza di trasparenza, dove ogni Event fornisce un servizio all’Event immediatamente superiore o, in caso di radice dell’albero, all’utente. Per questo motivo ci siamo mossi nella direzione di rappresentare la classe base Event (la nostra ComponentClass del pattern) come interfaccia. Essa non poteva senza ombra di dubbio essere una classe astratta per raggiungere tale scopo. Abbiamo dunque deciso di definire al suo interno i soli metodi comuni alle sue due classi figlie: BasicEvent e IntermediateEvent, rispettivamente la nostra LeafClass e CompositeClass del nostro Pattern, al semplice scopo di non permettere all’utente di notare differenza di interfaccia nell’utilizzo di una qualsiasi classa che estendesse la ComponentClass Event. Non ci siamo per questo motivo serviti della possibilità di implemntare tali metodi nella classe base tramite l’ausilio del modificatore default.

**TBD: Aggiungere Dettagli parla dei metodi , di Basic, Intermediate… etc…**

* **Factory (occupatene tu please)**

Avevamo tra le mani la struttura di classi che pensavamo meglio avrebbe potuto rappresentare il SMFT, il nostro successivo step logico si è mosso in direzione di affrontare la stesura delle classi che avrebbero modellato questo albero e lo avrebbero gestito. La risposta a queste due questioni era semplice: avevamo bisogno di una Classe EventManager (nata con lo scopo di gestire l’inizializzazione, operazione ripetuta più volte in fase di analisi e studio del SMFT, e setting dei parametri) e di una classe EventFactory (originariamente EventModeler) nata con lo scopo di instanziare i Basic Event e gli Intermediate Event.

* **TreeManager**

Una volta definiti i componenti del SMFT, avevamo dunque bisogno di una classe che ne rappresentasse le principali proprietà e potesse dunque occuparsi della loro gestione. Dato che le componenti dell’albero rilevanti all’utente sono i soli Basic Event ed il Top Event, questi devono i suoi attributi fondamentali. Una tale struttura consente di effettuare delle operazioni semplici sul sistema (come ottenere informazioni sullo stesso o manipolare lo stato delle foglie) senza doverci curare della sua composizione. La scelta di tenere in memoria il suo Λ ci permette di aggiornarlo in tempo costante, evitando così di doverlo ricalcolare ogni volta, operazione di complessità lineare rispetto al numero di foglie

**3.2 Simulator**

Sia la stima dell’ergodicità sia il calcolo degli intervalli do confidenza richiedono l’esecuzione di una serie di simulazioni del sistema: in particolare, ciascuna simulazione è indipendente delle altre, e inoltre lo scopo della simulazione influenza solo lo stato iniziale delle foglie dell’albero. Allora diventa necessaria una classe che si occupa di eseguire simulazioni tra loro indipendenti, accomunate tra loro solamente dalla durata, in modo da poter eseguire successivamente la media di tutte le simulazioni.

* **Timer**: il fatto che tutte le simulazioni abbiano pari durata si ripercuote sul codice tramite un timer unico, condiviso tra tutte le simulazioni. Ciò lo rende allora un parametro del simulatore, e si farà carico di conservare le informazioni sul tempo attuale e di calcolare l’istante successivo.

**3.3 DataCentre**

I dati restituiti dalle singole simulazioni presi singolarmente non sono significativi. Occorre allora avere un’entità che si occupi di immagazzinare i risultati di tutte le simulazioni e quantizzarli, a prescindere dal numero di campionamenti delle simulazioni.

Per l’inserimento e l’estrazione dei dati, dato che sia i campionamenti originali sia quelli quantizzati contengono dati eterogenei è divenuto necessario l’impiego delle classi wrapper **Pair** e **Triplet.**

* **Triplet**: rappresenta un singolo campione non quantizzato, dunque necessita di immagazzinare tre informazioni: il tempo, lo stato dell’albero e lo stato delle foglie.
* **Pair**: per rappresentare un campionamento quantizzato, potevamo usare la classe Triplet, codificandolo sopra, però, essendo i dati quantizzati ed il quanto noto, salvare il tempo diventa superfluo, si salva allora solo lo stato dell’albero

**statistic**

1. **STRUTTURA DEL CODICE**

Dalla stesura del Class Diagram è risultato subito evidente la convenienza dell’impiegare il Design Pattern Composite per rappresentare lo SMFT e del Pattern Observer per rendere possibile alla Classe EventManager di rimanere sembre aggiornato sulla struttura del modello.

**4.1 Composite**

**4.2 Singleton**

Alcuni componenti del programma devono essere unici, in quanto forniscono lo stesso servizio in modo indipendente dai chiamanti o dell’ambiente in cui offrono i propri servizi. Queste qualità sono riscontrate nell’Event Factory e in HarryPlotter, che forniscono un servizio (creazione componenti e plotting) stateless

1. **TESTING**

Per garantire il funzionamento del package basta garantire che funzioni le sue componenti più nevralgiche, ossia quelle relative alla simulazione e l’elaborazione dei dati.

Il testing della simulazione si riconduce a testare il corretto funzionamento della classe timer e dell’albero stesso. Data la natura composite dell’albero, è sufficiente garantire il corretto funzionamento dei singoli componenti di esso. Provato ciò il corretto funzionamento dell’albero, e quindi, della simulazione, è garantito

1. **MOCKUPS**
2. **CONSIDERAZIONI FINALI**