

## **Indice della documentazione**

La documentazione di compone dei seguenti paragrafi:

- **Metodologia**: si descrive brevemente l'approccio usato nell'analisi dei requisiti, nella progettazione e nello sviluppo;
- **Ipotesi e assunzioni preliminari**: si dettagliano le implicazioni di alcune assunzioni fatte nello Schema della Simulazione, in particolare in riferimento ai tipi di dati usati e alle relative problematiche di overflow;
- **Diagramma delle classi**: si riporta il diagramma UML della modellazione degli oggetti in gioco nella simulazione;
- **File headers commentati**: si riportano e commentano i file di intestazione (headers C++) delle classi sopra descritte.

## Metodologia

Di seguito si descrive la metodologia utilizzata nella realizzazione del software, a partire dai requisiti, passando per la progettazione e concludendo con lo sviluppo.

### *Analisi dei requisiti*

L'analisi dei requisiti e' stata effettuata descrivendo uno schema della simulazione, in cui si modella la simulazione in modo indipendente da quanto indicato sul testo “*Simulazione*”, non ritenendo sufficiente quanto li' esposto ai fini della progettazione del software.

Si rimanda pertanto all'allegato “**Schema simulazione**” per la definizione dei requisiti, in termini di come realizzare la simulazione.

### *Progettazione del software*

Il progetto del software e' stato realizzato con un approccio evolutivo/incrementale:

- si e' partiti con la modellazione delle classi rappresentative degli oggetti in gioco nella simulazione, adottando un approccio OOA/OOD (*Object Oriented Analysis and Design*), il cui risultato e' riportato di seguito come **diagramma delle classi**;
- si e' proseguito raffinando tali classi mediante un procedimento iterativo, basato essenzialmente su un approccio TDD (*Test Driven Development*), di cui resta traccia nel codice (una serie di metodi di test, scritti al fine di verificare il funzionamento degli oggetti e delle loro relazioni);
- non e' stata prodotta la documentazione UML “completa” (in particolare non si e' andati oltre il diagramma delle classi), in quanto oltre gli scopi del progetto.

### ***Sviluppo del software***

Coerentemente con l'approccio evolutivo/incrementale adottato, lo sviluppo del software e' proceduto come segue:

- in una prima fase sono state definite ed implementate le classi coinvolte nella simulazione, di cui sono stati abbozzati metodi e proprieta';
- successivamente, mediante TDD, il software scritto e' stato validato e quindi corretto ed esteso, raffinando al contempo sia requisiti che progettazione;
- lo sviluppo e' stato quindi completato con l'implementazione completa di tutti le classi (in particolare, l'implementazione dei metodi per la generazione casuale dei tempi di inter-arrivo e di quelli di servizio, coerentemente con le distribuzioni di probabilita' assegnate, e' stata una delle ultime).

## ***Ipotesi e assunzioni preliminari***

Sono state fatte le seguenti assunzioni:

- la simulazione puo' essere eseguita, con un livello di precisione accettabile, in riferimento ad un tempo di clock simulato, rappresentabile come tipo di dato **long**; l'intervallo di valori in uso e'  $0..2^{31}-1$ ;
- la generazione di numeri pseudo-casuali non richiede la realizzazione di apposite funzioni (quali quelle descritte sul testo “*Simulazione*” nel paragrafo “*Metodo della congruenza lineare*”), ma si puo' far uso delle funzioni standard **rand()**/**srand()**, le cui implementazioni si assumono gia' ottimizzate per quanto riguarda la generazione di sequenze di numeri pseudo-casuali che siano:
  - statisticamente indipendenti;
  - uniformemente distribuiti.
- in merito al **problema dell'overflow per il tipo di dato utilizzato per il clock**, si ha quanto segue:
  - i numeri pseudo-casuali generati da **rand()** sono di tipo **long** e compresi nell'intervallo  $0..2^{31}-1$ ;
  - il passaggio attraverso le funzioni che implementano le distribuzioni di probabilita' caratteristiche dei centri di servizio (esponenziale, k-erlangiana, iperesponenziale) comportano:
    - l'uso di numeri pseudo-casuali di tipo **double**, compresi in  $0..1$ ;
    - la produzione di numeri pseudo-casuali di tipo **double**, compresi in  $0..INF$  (estremi inclusi);
  - e' necessario convertire tali numeri prodotti dalle distribuzioni di probabilita' in un intervallo molto minore di  $0..2^{31}-1$ , al fine di evitare **overflow**;

stante quanto sopra, si e' deciso di procedere come segue:

- utilizzare **rand()** per produrre numeri pseudo-casuali di tipo **long** in  $0..2^{31}-1$ ;
- convertire tali numeri in **double** e traslarli in  $0..1$ , con una precisione di 8 cifre decimali (vedere funzione “*getRandomUniform*” e relative costanti);
- utilizzare i risultati di tipo **double** di “*getRandomUniform*” per implementare le funzioni “*getRandomExponential*”, “*getRandomKappaErlang*”, “*getRandomIperExponential*”, che ritornano sempre numeri di tipo **double**, limitando ove possibile l'insorgenza di valori INF;
- convertire i numeri **double** prodotti dalle funzioni exp,k-erl,ipexp in **long** e scalarli (vedere funzione “*getRandomToClockScaled*” e relative costanti), in modo da ottenere numeri che siano, nella quasi totalita' dei casi, molto piu' piccoli di  $2^{31}-1$ ;

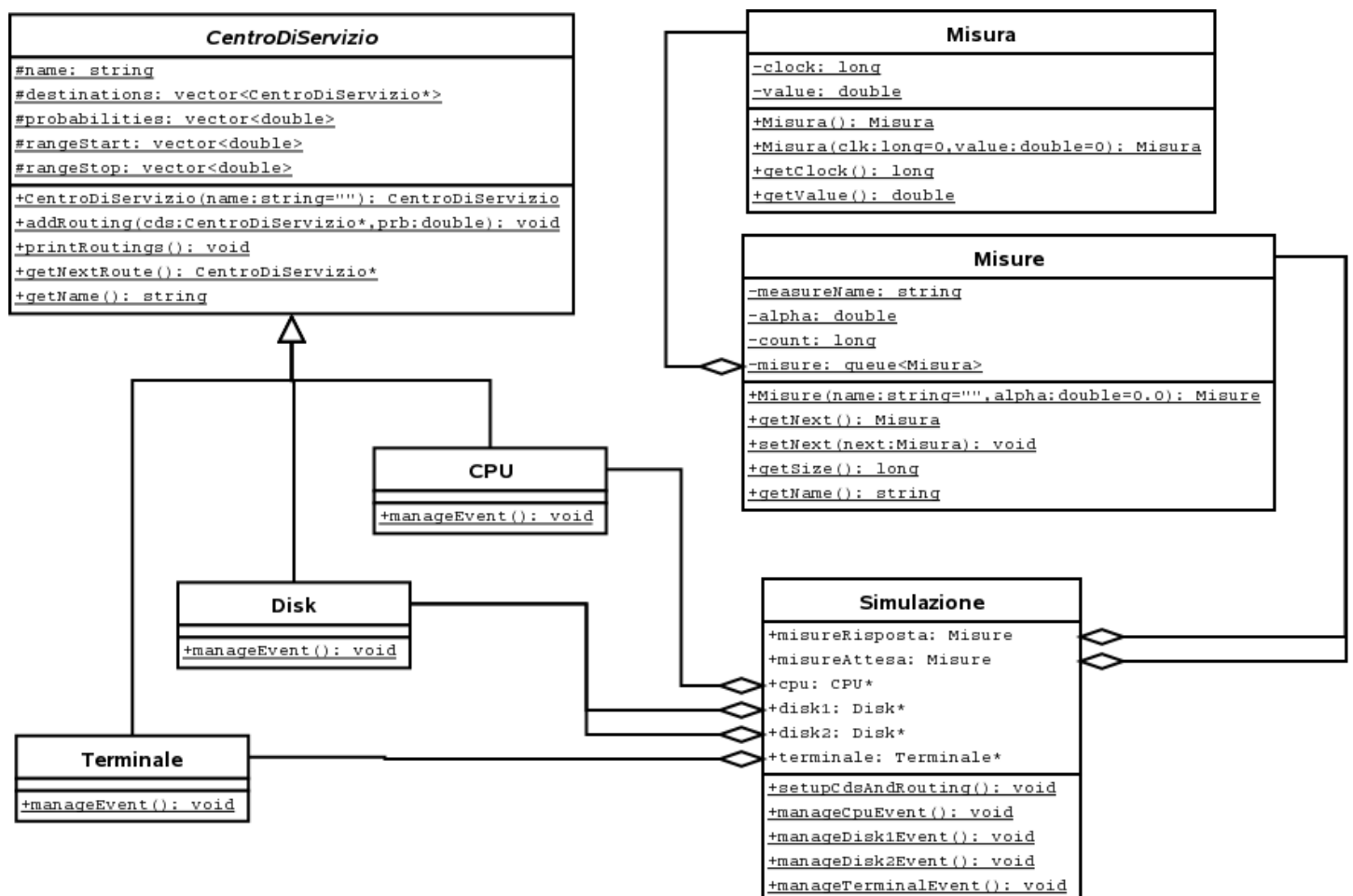
- **monitorare i risultati** per verificare che valori eccessivamente alti di tali numeri pseudo-casuali non comportassero overflow nel clock; la verifica e' immediata sulle stampe di output dei vari run:
  - il problema si verifica nella generazione degli eventi, dove al clock corrente va *sommato* un clock generato randomicamente dal centro di servizio; poiche' un long in overflow diventa negativo, l'evento (o gli eventi) corrispondente comparirebbe all'inizio del calendario degli eventi stampato a fine simulazione;
  - tale problema non si verifica con i calcoli relativi alle misure, trattandosi per queste di differenze tra il clock corrente e clock di altri eventi;
- per il calcolo di media e varianza delle varie misure effettuate, si sono utilizzati variabili di tipo double, onde evitare i problemi di overflow sopra citati.

## Diagramma delle classi

Di seguito si riporta il diagramma UML inizialmente prodotto, per la progettazione delle classi rappresentative degli oggetti in gioco nella simulazione.

Per la descrizione dei metodi e delle proprietà effettivamente implementate, dopo il processo di revisione basato sul TDD, si veda il paragrafo “**File headers commentati**”.

Il principale cambiamento è stata l'introduzione delle classi Evento e Calendario, nonché la revisione delle responsabilità inizialmente assegnate alla classe Simulazione e successivamente trasferite alla classe Calendario.



## File di intestazione (headers C++) commentati

Di seguito si riportano e si commentano i file di intestazione delle classi.

### CentroDiServizio

```
class CentroDiServizio
{
// questa classe rappresenta il generico Centro di Servizio, specializzato dalle
// sottoclassi Cpu, Disk e Terminali
// utilizza il template Vector

public:
// elenco dei metodi con visibilita' pubblica
// (nessuna proprieta', ove necessario vengono usati metodi di accesso)

// costruttore, i cui parametri rappresentano:
// - flag per le stampe di debug
// - nome del CdS
// - primo parametro della distribuzione di probabilita'
// - eventuale secondo parametro della distribuzione di probabilita'
CentroDiServizio(int=0, string="CentroDiServizio", double=0, double=0);

// distruttore, virtuale affinche' venga chiamato quello delle sottoclassi
virtual ~CentroDiServizio();

// metodo per aggiungere un routing al CdS, i parametri rappresentano:
// - CdS su cui inoltrare il job
// - probabilita' di scegliere tale CdS
void addRouting(CentroDiServizio*, double);

// metodo di stampa dei routing, usato per verifica
void printRoutings();

// metodo per recuperare il prossimo routing: ritorna il CdS a cui verra' passato
// il job;
// - il parametro e' un numero casuale nell'intervallo 0..1
CentroDiServizio* getNe tRoute(double);

// metodo per recuperare il clock a cui verra' completato il job relativo al
// prossimo evento da inserire che ha per SRC il Cds
// virtuale affinche' sia invocata l'implementazione della sottoclasse
virtual long getNe t!ventC"o#$( );

// metodo per gestire le code del CdS, nel caso sia DST di un evento
// virtuale affinche' sia invocata l'implementazione della sottoclasse
virtual void register%sDestination();
```

---

```
// metodo per gestire le code del CdS, nel caso sia SRC di un evento
// virtuale affinche' sia invocata l'implementazione della sottoclasse
virtual void register%sSour#e();

// metodo che ritorna un booleano per verificare che il Cds sia di tipo Terminale
// oppure Cpu/Disk (ritorna 1 nel primo caso, 0 nel secondo)
int is&er' ina"(rCpuDis$());

// metodo che ritorna il numero di job in attesa (se Cpu/Disk) o il numero di
// richieste inviate ed in attesa di risposta (Terminale)
long get)ueued(r*aiting());

// metodo di accesso, che ritorna il primo parametro della distribuzione
// assegnata al CdS
double getDistr+ara' ();

// metodo di accesso che ritorna il nome assegnato al CdS
string getNa'e();
```

**protected:**

```
// elenco di proprieta' visibili alle sole sottoclassi

// primo parametro della distribuzione assegnata al CdS
double dDistr+ara' ;

// eventuale secondo parametro della distribuzione assegnata al CdS
double dDistrSu, +ara' ;

// flag per le stampe di debug
int iDe, ug;

// numero di job in attesa (se Cpu/Disk) o il numero di
// richieste inviate ed in attesa di risposta (Terminale)
long i)ueued;

// nome assegnato al CdS
string na'e;

// booleano per verificare che il Cds sia di tipo Terminale
// oppure Cpu/Disk (1 nel primo caso, 0 nel secondo)
int is&er' ina";
```

**private:**

```
// elenco di proprieta' e metodi visibili alla sola classe CentroDiServizio

// metodo per la verifica di coerenza dei vettori del routing (vedere sotto)
// ritorna 1 se i quattro vettori hanno lo stesso numero di elementi, 0 altrimenti
int #-e#$$.e#tors();
```



```
// metodo per la stampa di un messaggio di errore nel caso checkVectors ritorni 0
void printError();

// vettore delle DST assegnate al CdS
vector<CentroDiServizio*> destinations;

// vettore delle probabilita' associate alle DST
vector<double> probabilities;

// vettore dei range di inizio intervallo, associati alle DST
// utilizzati dal metodo getNextRoute
vector<double> rangeStart;

// vettore dei range di fine intervallo, associati alle DST
// utilizzati dal metodo getNextRoute
vector<double> rangeStop;

1;
```

## Terminale

```
class &er'ina"e : public CentroDiServizio
{
// questa classe rappresenta un blocco di Terminali, specializzando la classe
// CentroDiServizio

public:
// elenco dei metodi con visibilita' pubblica
// (nessuna proprieta', ove necessario vengono usati metodi di accesso)

// costruttore: vedere superclasse; unica differenza: imposta isTerminal a 1
&er'ina"e(int=0, string="&er'ina"e", double=0, double=0);

// distruttore: vedere superclasse
virtual ~&er'ina"e();

// metodo inserito in fase OOD, ma non implementato: in fase TDD e' emerso che
// il processamento degli eventi e' di competenza del Calendario
void 'anage!vent();

// specializzazione del metodo della superclasse:
// coerentemente con la distribuzione di probabilita' assegnata,
// invoca la funzione getRandomExponential
virtual long getNe t!ventC"o#$( );

// specializzazione del metodo della superclasse:
// coerentemente con le assunzioni fatte, decrementa il numero di richieste
// inviate ed in attesa di risposta
virtual void register%sDestination();

// specializzazione del metodo della superclasse:
// coerentemente con le assunzioni fatte, incrementa il numero di richieste
// inviate ed in attesa di risposta
virtual void register%sSour#e();

1;
```

## Cpu

```
class C+2 : public CentroDiServizio
{
// questa classe rappresenta una C+2, specializzando la classe CentroDiServizio

public:
// elenco dei metodi con visibilita' pubblica
// (nessuna proprieta', ove necessario vengono usati metodi di accesso)

// costruttore: vedere superclasse; unica differenza: imposta isTerminal a 0
C+2(int=0, string="&er' ina"e", double=0, double=0);

// distruttore: vedere superclasse
virtual ~C+2();

// metodo inserito in fase OOD, ma non implementato: in fase TDD e' emerso che
// il processamento degli eventi e' di competenza del Calendario
void ' anage! vent();

// specializzazione del metodo della superclasse:
// coerentemente con la distribuzione di probabilita' assegnata,
// invoca la funzione getRandomIperExponential
virtual long getNe t!ventC"o#$( );

// specializzazione del metodo della superclasse:
// coerentemente con le assunzioni fatte, incrementa il numero di richieste
// inviate ed in attesa di risposta
virtual void register%sDestination();

// specializzazione del metodo della superclasse:
// coerentemente con le assunzioni fatte, decrementa il numero di richieste
// inviate ed in attesa di risposta
virtual void register%sSour#e();

1;
```





















