

Appendice A

Progettazione in UML del simulatore

In questo capitolo viene affrontata la fase di progettazione a oggetti del simulatore, utilizzando il linguaggio di modellazione UML.

Unified Modeling Language è un linguaggio per la costruzione e la documentazione di sistemi software complessi. Le notazioni di questo linguaggio consentono la realizzazione di diagrammi in grado di rappresentare diverse astrazioni del modello studiato. I diagrammi standard di UML sono i seguenti:

1. diagramma dei casi d'uso (use case diagram);
2. diagramma delle classi (class diagram);
3. diagramma di sequenza (sequence diagram);
4. diagramma di stato (state diagram);
5. diagramma di collaborazione (collaboration diagram);
6. diagramma di attività (activity diagram);
7. diagramma dei componenti (component diagram);
8. diagramma di allocazione (deployment diagram).

Questi diagrammi forniscono nel loro insieme molteplici prospettive sul sistema che viene sviluppato. La scelta di quali aspetti del modello evidenziare ha quindi un'influenza profonda per la comprensione del progetto.

Per la modellazione a oggetti del simulatore, si è pensato di utilizzare i diagrammi indicati ai punti 2 e 3 dell'elenco. La caratteristica di queste rappresentazioni grafiche consiste nella capacità di fornire delle viste *statiche* e *dinamiche* del sistema in esame. Le rappresentazioni statiche descrivono gli attributi, i metodi e le relazioni tra le classi, mentre le rappresentazioni dinamiche analizzano la parte di controllo e di comunicazione (in termini di scambio di messaggi) del sistema. Sono inoltre previsti due diversi livelli di sviluppo dei diagrammi: il *micro-livello*, che include le singole classi con le loro relazioni e il *macro-livello*, che include raggruppamenti delle varie classi.

A.1 Diagramma delle classi

Un diagramma delle classi mostra la struttura statica del modello del sistema in fase di studio. In particolare esso evidenzia le classi, la loro struttura interna e le loro relazioni.

Data la complessità del simulatore, si è pensato di raggruppare le classi in moduli o package per facilitare l'analisi del modello e per realizzare un'architettura modulare del sistema in esame. Ciascun modulo rappresenta così un sottosistema, costituito da gruppi di classi correlate tra loro.

Il criterio seguito per creare i package si basa sulle relazioni di *generalizzazione-specializzazione* e di *aggregazione* esistenti tra le classi. Il primo tipo di relazione permette di individuare le superclassi, che raccolgono comportamenti comuni, e le classi derivate, che specializzano i comportamenti delle classi padre, aggiungendo nuove responsabilità. La seconda relazione invece consente di rappresentare singoli oggetti come aggregazione di diverse entità, alle quali vengono delegati alcuni compiti dell'oggetto composto.

A.1.1 Node_B e Terminali mobili

Il package rappresentato in figura evidenzia i legami esistenti tra quelle entità, che definiscono il comportamento dei mobili e delle stazioni radio base nel simulatore UMTS.

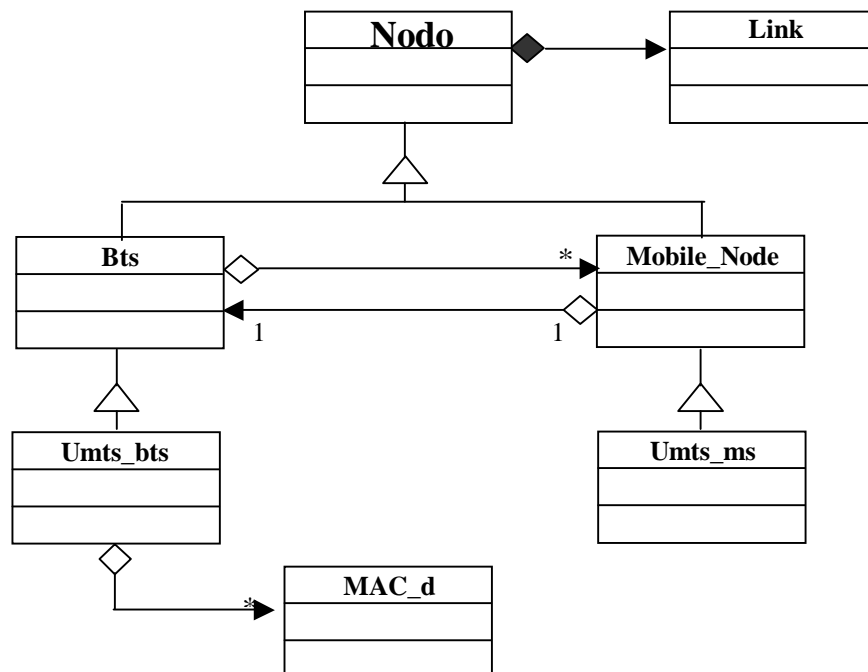


Figura A.1 : Diagramma delle classi

La classe *Nodo* è progettata come classe astratta fondamentale. Essa fornisce un'interfaccia pubblica per gestire l'intera gerarchia di calssi derivate.

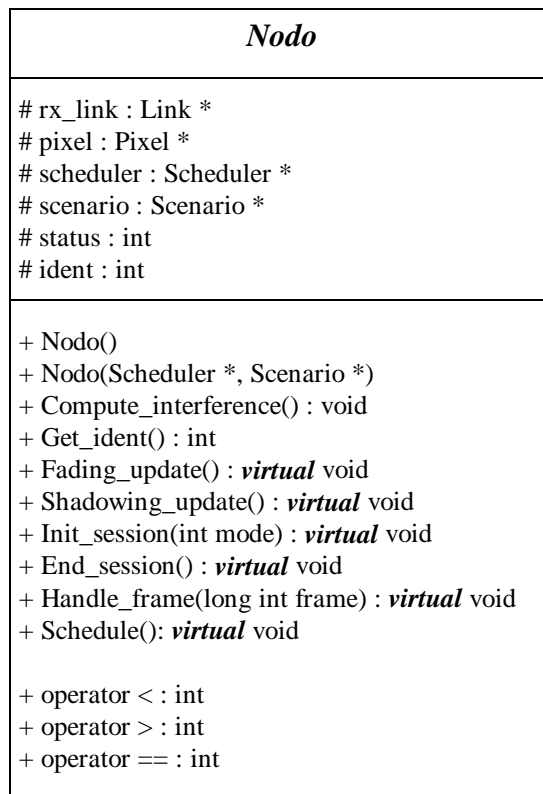
Le classi *Bts* e *Mobile_Node* sono istanze particolari della classe *Nodo*. In esse vengono ridefinite le funzioni virtuali della classe padre.

Le calssi *Umts_bts* e *Umts_ms* infine rappresentano un grado ulteriore di specializzazione, in quanto si trovano all'ultimo livello della gerarchia di ereditarietà.

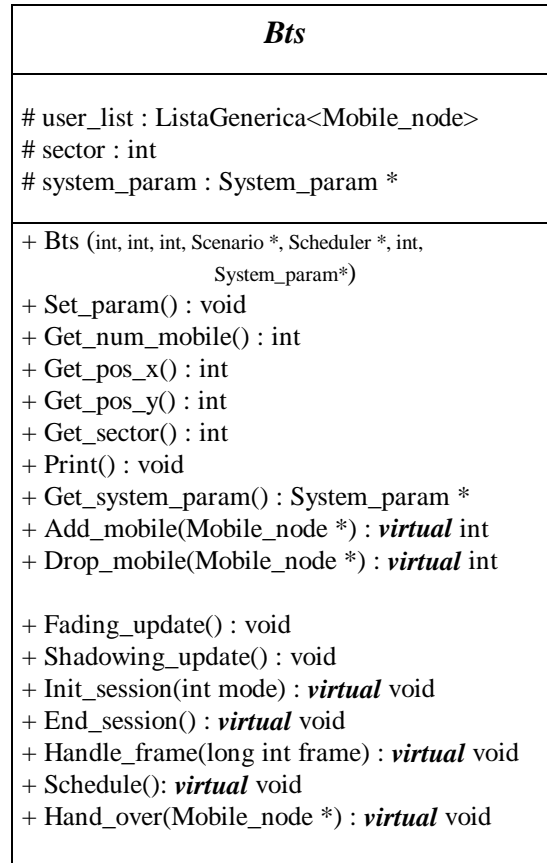
A.1.1.1 Struttura statica delle classi con dettagli di livello implementativo

Si propone adesso una descrizione dell'interfaccia di ogni classe del sottosistema, specificando esaustivamente, per ogni entità, gli attributi (tipo e visibilità) e i metodi (visibilità, tipo di ritorno, tipo e nome dei parametri formali).

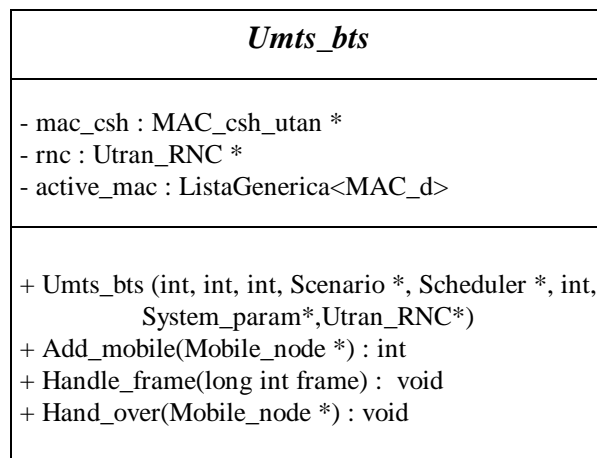
- *Classe Nodo:*



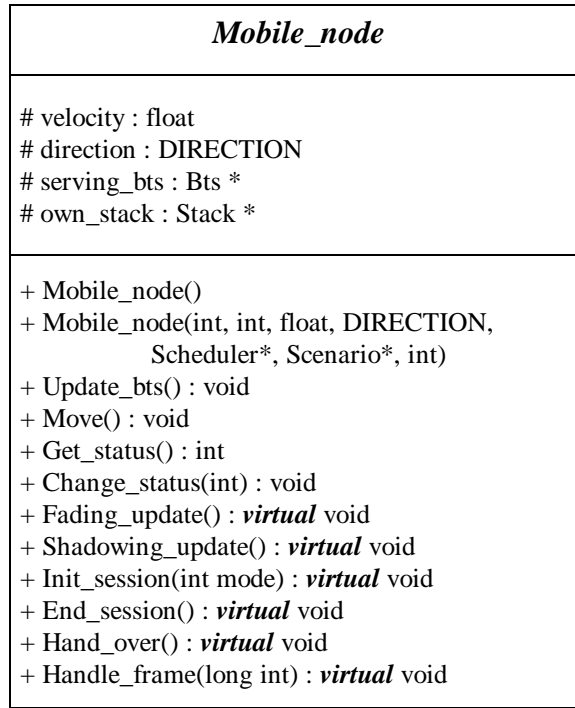
- *Classe Bts:*



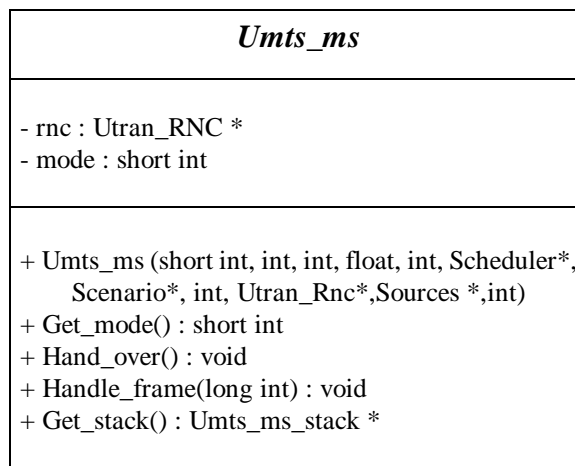
- *Classe Umts_bts:*



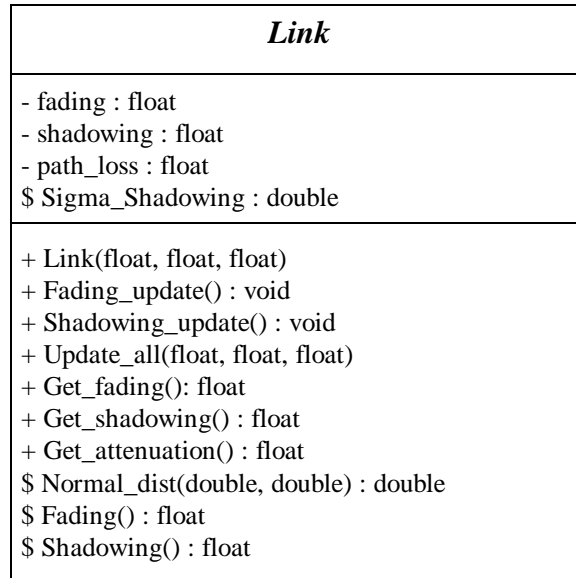
- Classe *Mobile_node*:



- Classe *Umts_ms*:



- *Classe Link:*



A.1.2 Calendario degli Eventi

Questo nuovo modulo prende in considerazione quelle entità che si occupano di coordinare il succedersi degli eventi.

Una prima entità, di nome *Scheduler*, ha il compito di gestire un insieme variabile di eventi, che potrebbero accadere durante il periodo della simulazione. Tali eventi sono organizzati in una lista circolare, ordinata in base ai tempi in cui gli eventi devono capitare. La ricerca della posizione corretta in cui lo Scheduler deve inserire un nuovo evento è di tipo dicotomico, per ottimizzare la velocità di esecuzione.

La classe *Evento* è definita come classe astratta fondamentale. Essa fornisce un'interfaccia pubblica per gestire l'intera gerarchia di classi derivate. Queste ultime sono istanze particolari con il compito di specializzare i comportamenti della classe padre.

In figura si riporta il diagramma delle classi di questo sottosistema, mostrando come la soluzione adottata consenta di aggiungere nuovi eventi, senza modificare le classi già esistenti.

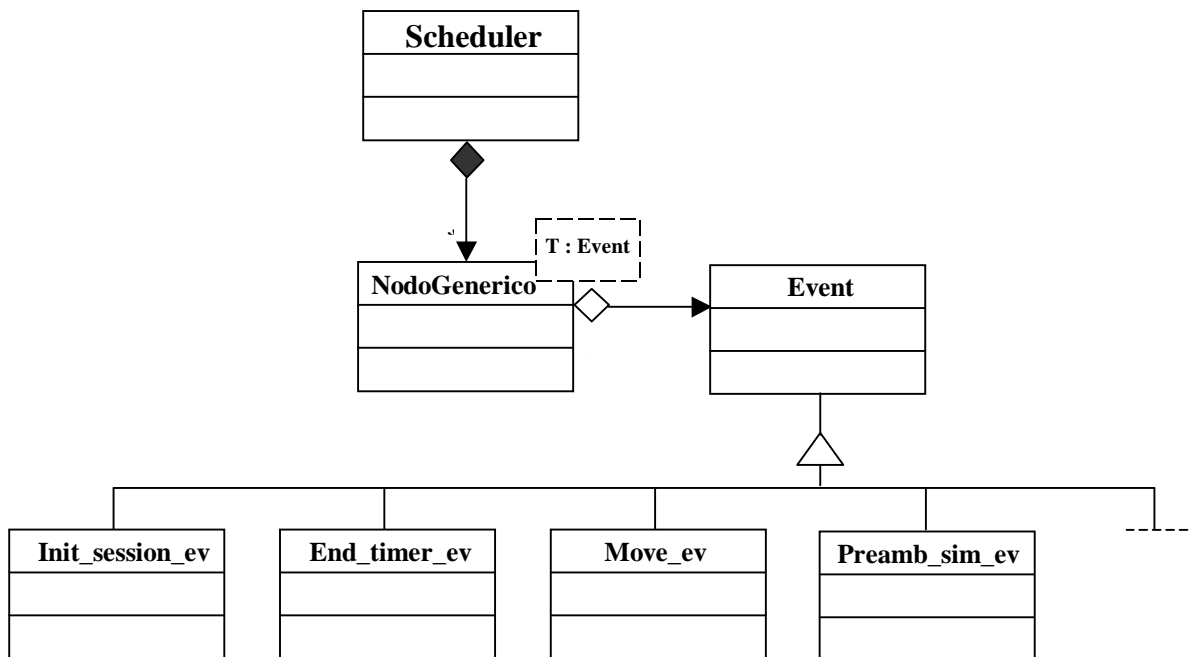


Figura A.2 : Diagramma delle classi

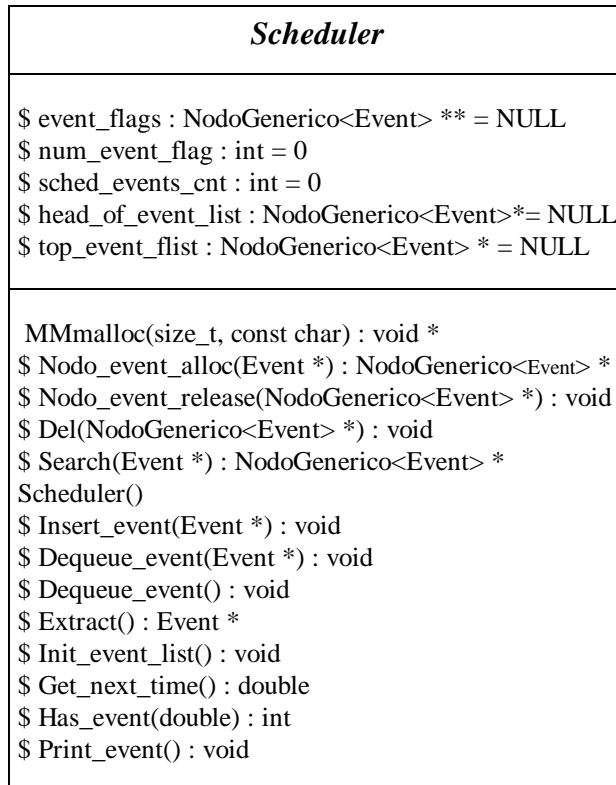
Un elenco completo degli eventi definiti è qui riportato:

- *Init_session_ev*, che determina l'inizio di una nuova sessione per un mobile.
- *End_timer_ev*, che segnala la scadenza di un timer.
- *Move_ev*, che indica l'istante di tempo in cui i mobili di una certa classe di velocità devono aggiornare la loro posizione.
- *Preamb_sim_ev*, per segnalare alla pari entità MAC nell'UTRAN che un UE ha cercato di accedere alla rete.
- *Phy_acc_conf_ev*, che invoca la primitiva PHY_acc_conf sull'opportuna entità MAC-c/sh.
- *Connection_SETUP_ev*, chiamato dall'RRC nell'UTRAN per simulare il ritardo di trasmissione sul FACH.
- *Connection_REJECT_ev*, chiamato dall'entità RRC presente nell'UTRAN per simulare il ritardo di trasferimento del messaggio di rilascio della connessione sul canale FACH.
- *Switch_source_ev*, per segnalare il cambiamento di stato di una sorgente di traffico dati. E' invocato dalla sorgente stessa.
- *Send_pck_ev*, per indicare l'istante di generazione del prossimo pacchetto in una sorgente di traffico.
- *Switch_channel_ev*, per segnalare il cambiamento di stato del canale fisico.
- *Change_source_state_ev*, per segnalare il cambiamento di stato della sorgente STREAMING. E' invocato dalla sorgente stessa.

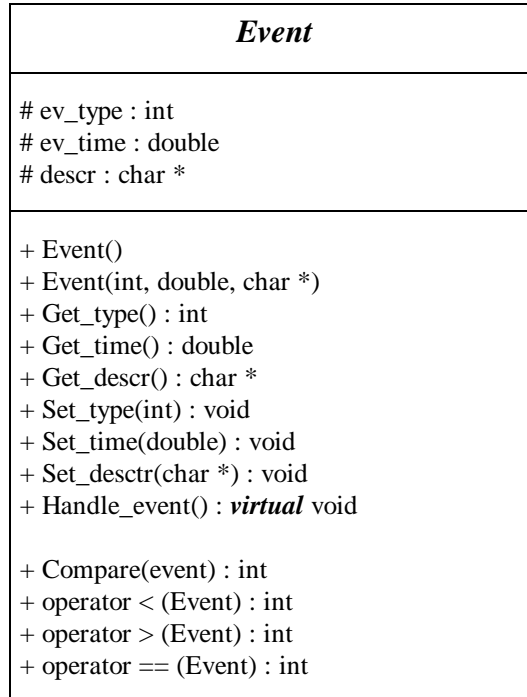
A.1.2.1 Descrizione statica delle classi con dettagli di livello implementativo

Si focalizza adesso l'attenzione su quegli elementi delle classi che ne definiscono le caratteristiche (gli attributi) e il comportamento (i metodi).

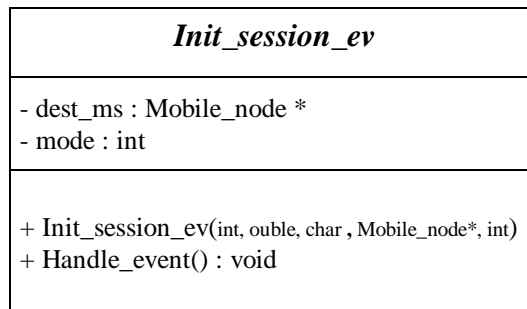
- *Classe Scheduler:*



- *Classe Event:*



- *Classe Init_session_ev:*



A.1.3 Scenario simulativo

Il package rappresentato in figura A.3 descrive la struttura dell'ambiente di simulazione utilizzato per studiare le prestazioni dei sistemi di comunicazione mobile a pacchetto UMTS.

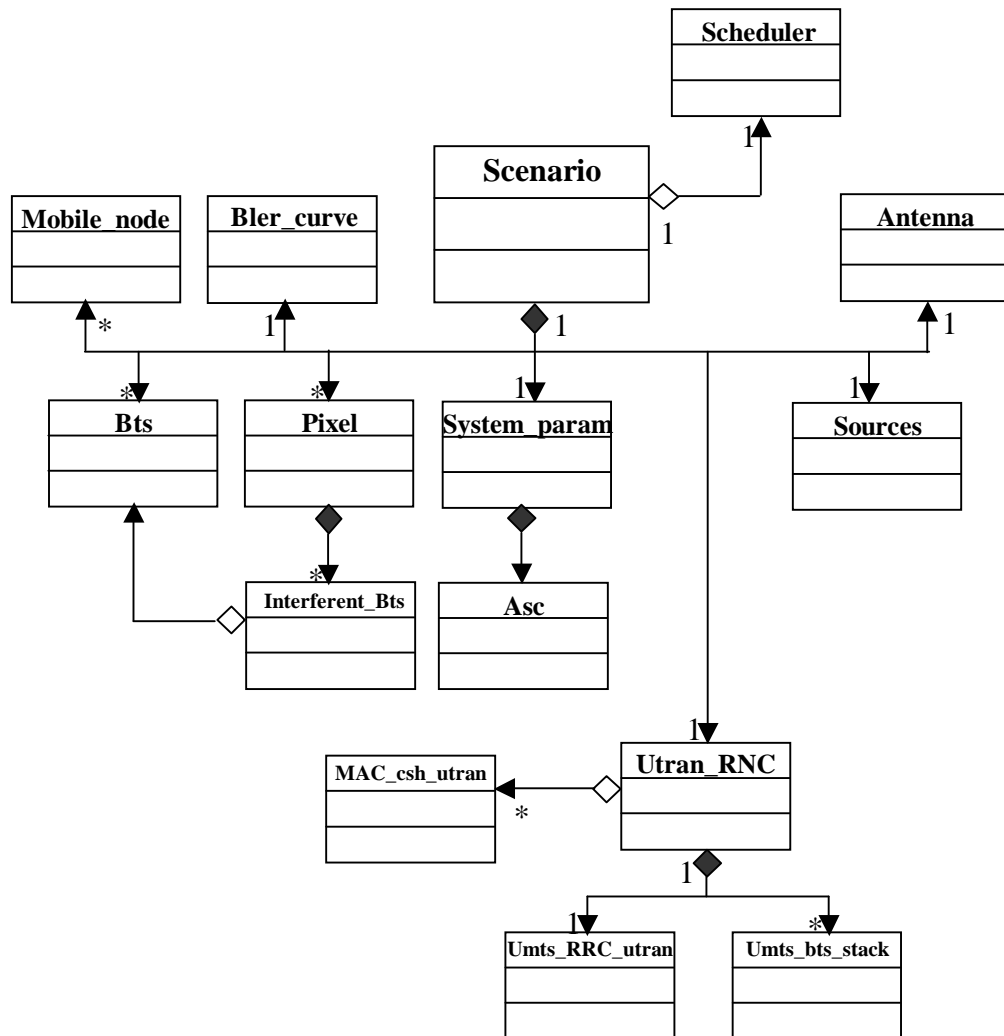


Figura A.3 : Diagramma delle classi

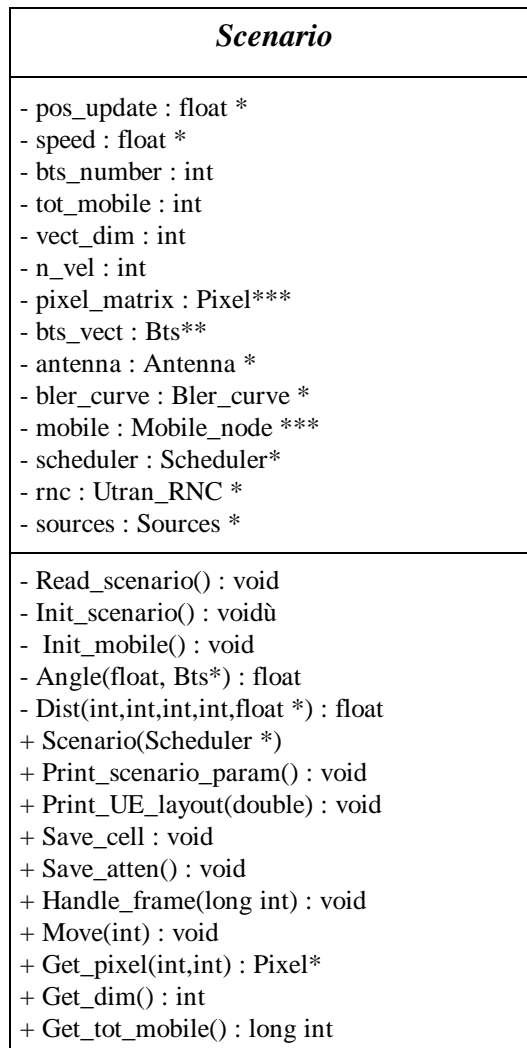
Le caratteristiche dello scenario simulativo realizzato sono qui riportate:

- Topologia ripiegata.
- Suddivisione dello spazio di simulazione in pixel di dimensione opportuna per contenere i tempi necessari all'elaborazione dei risultati.
- Ambiente di simulazione basato sul modello macrocellulare, ma adattabile ad ambienti diversi.
- Disposizione delle BTS coerente con la rappresentazione di celle esagonali.
- Numero di base station determinato dal numero di quelle posizionate sui due assi di riferimento.
- Possibilità di utilizzare base station con antenne omnidirezionali e trisettoriali.

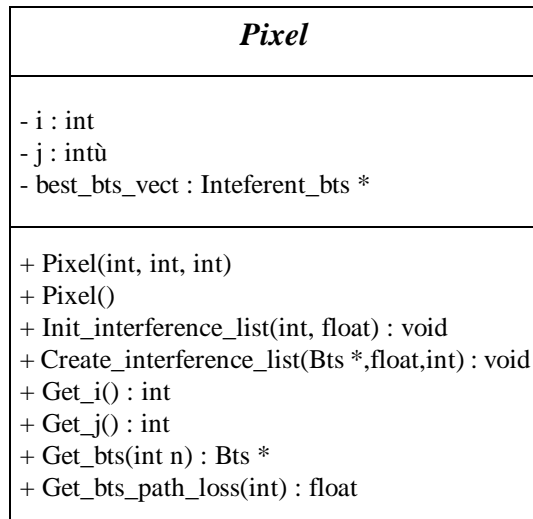
- Attenuazione del segnale dovuta alla propagazione dipendente dal modello usato per studiare un determinato ambiente.
- Aggiornamento di posizione dei mobili dipendente dalla velocità; viene effettuato ogni volta che un mobile percorre una distanza pari alla dimensione dei pixel.
- Aggiornamento dello shadowing indipendente da quello di posizione ed effettuato ogni volta che un UE percorre la distanza di correlazione propria dell'ambiente di simulazione
- Aggiornamento del fading veloce effettuato ad ogni trama.

A.1.3.1 Descrizione statica delle classi con dettagli di livello implementativo

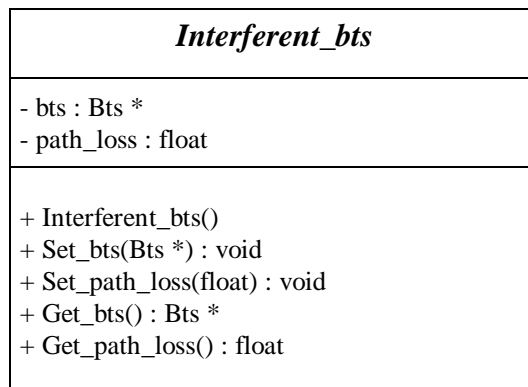
- *Classe Scenario:*



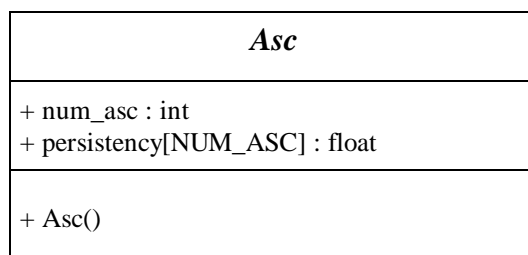
- *Classe Pixel:*



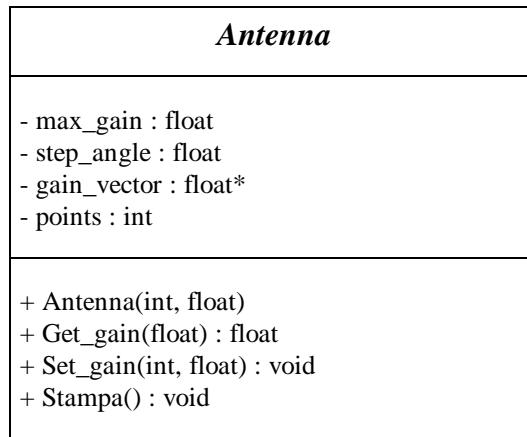
- *Classe Interferent_bts:*



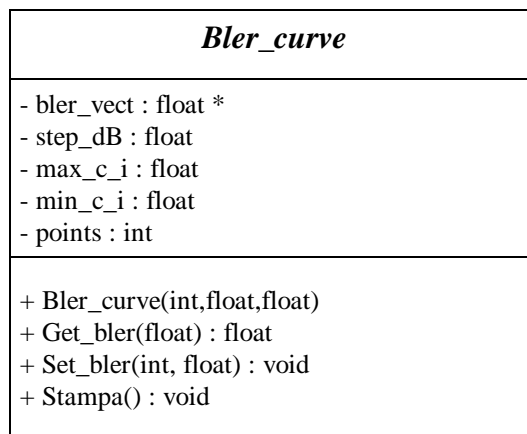
- *Classe Asc:*



- *Classe Antenna:*



- *Classe Bler_curve:*



A.1.4 Pile Protocolлари

In figura A.4 vengono rappresentate le entità con il compito di gestire lo stack protocollare nel mobile e nell'UTRAN.

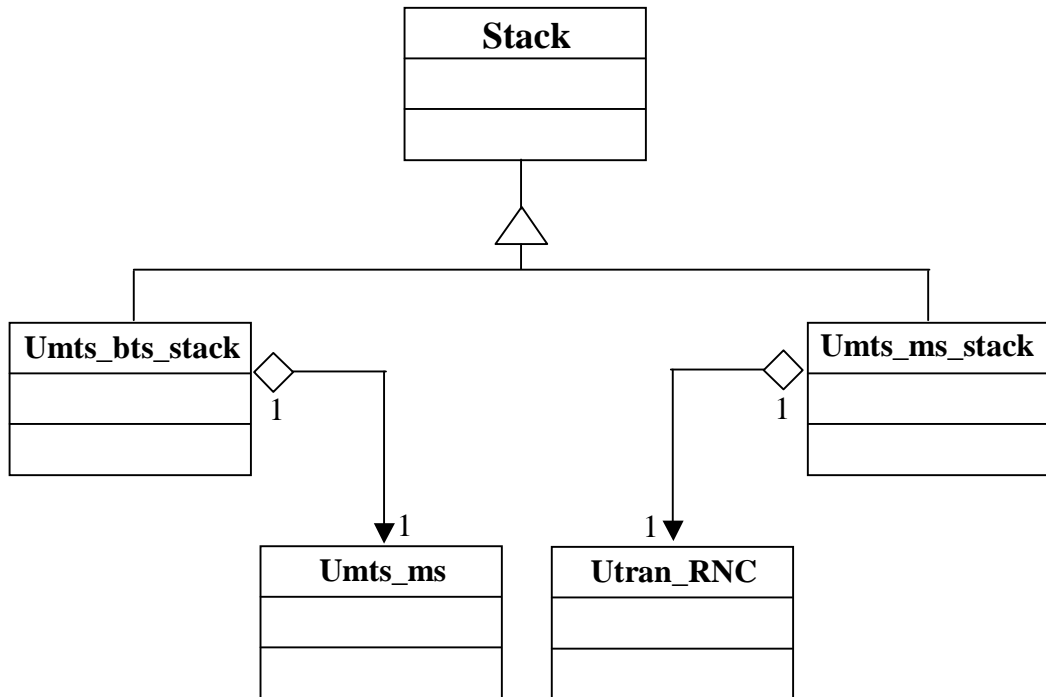


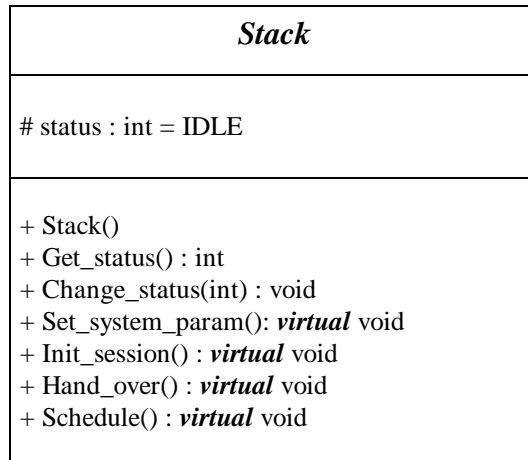
Figura A.4 : Diagramma delle classi

Stack è una classe astratta che dichiara un'interfaccia generica per gestire una pila protocollare. Molti metodi in essa definiti sono funzioni virtuali da ridefinire nelle classi derivate secondo lo scopo.

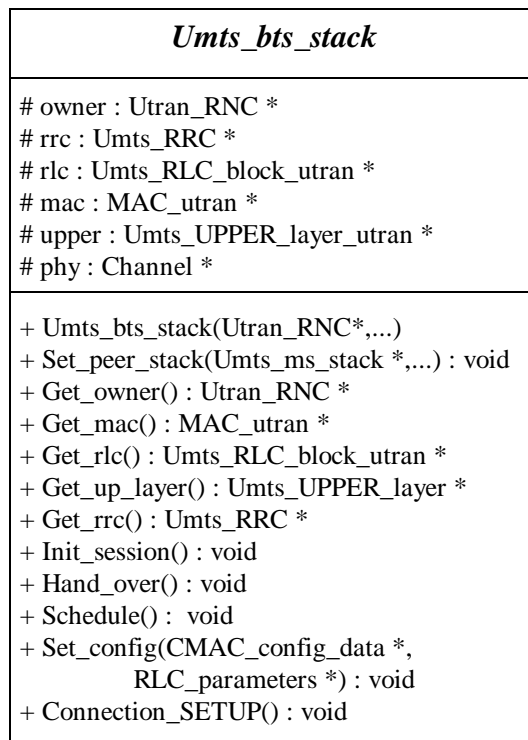
Umts_bts_stack e *Umts_ms_stack* sono istanze particolari della classe *Stack*. Esse devono coordinare l'insieme di livelli che formano la pila rispettivamente nelle BTS e negli UE.

A.1.4.1 Descrizione statica delle classi con dettagli di livello implementativo

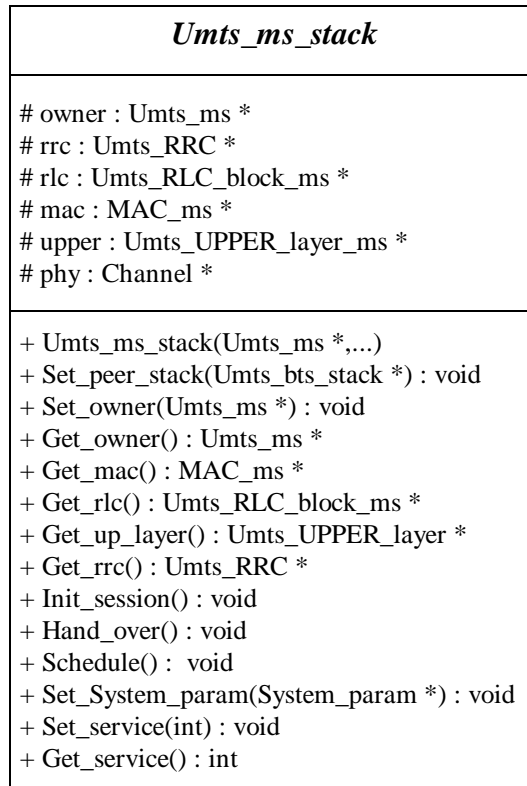
- Classe Stack :



- Classe Umts_bts_stack :



- Classe *Umts_ms_stack* :



A.1.5 Timer di sistema

Questo modulo illustra la gerarchia di derivazione degli orologi usati dai vari livelli delle pile protocollari.

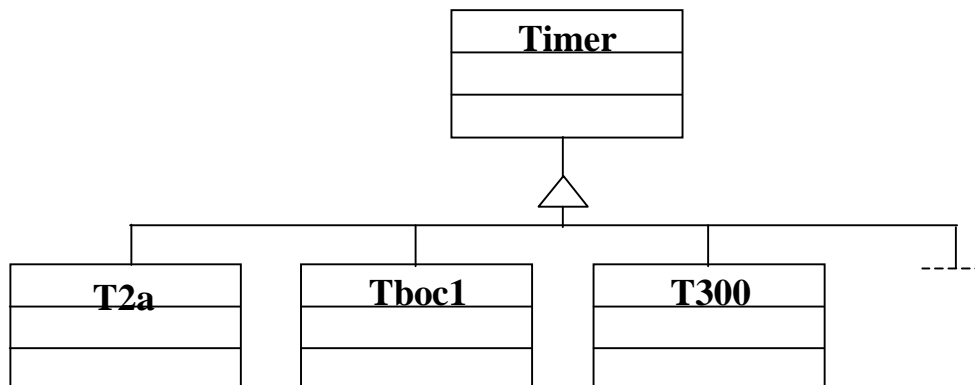


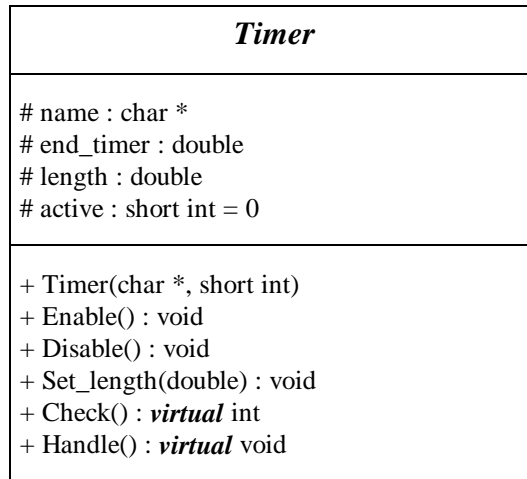
Figura A.5 : Diagramma delle classi

Timer è una superclasse che definisce un comportamento comune per gestire ogni tipo di orologio. Il metodo "*Handle()*" è dichiarato come funzione astratta, in quanto deve essere ridefinito nelle classi figlie, a seconda della particolare soluzione adottata.

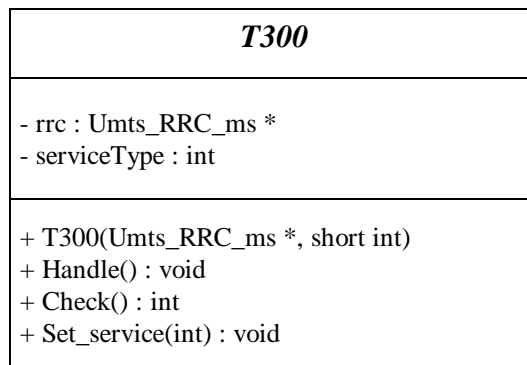
A.1.5.1 Descrizione statica delle classi con dettagli di livello implementativo

Dato il numero piuttosto elevato di "Concrete Timer" presenti nel simulatore, si definisce nei particolari la struttura statica della classe astratta e di una sola classe derivata.

- *Classe Timer:*



- *Classe T300:*



A.1.6 Livello MAC

Il package in figura visualizza le entità che compongono il livello MAC nel simulatore.

Quello che si cerca di mettere in evidenza dal diagramma è il diverso rapporto di comunicazione tra entità MAC-c/sh e MAC-d nell'UTRAN, rispetto al rapporto esistente nell'UE. Infatti all'interno di ogni mobile vi è un solo MAC-d, la cui gestione è affidata all'unico MAC-c/sh presente nel terminale stesso. Nell'UTRAN, invece, vi sono tanti MAC-d quanti sono i mobili e un MAC-c/sh per ogni cella dell'UTRAN. In particolare ogni MAC-c/sh ha in carico l'insieme delle entità MAC-d, in corrispondenza di quei mobili che si trovano all'interno della cella da esso servita.

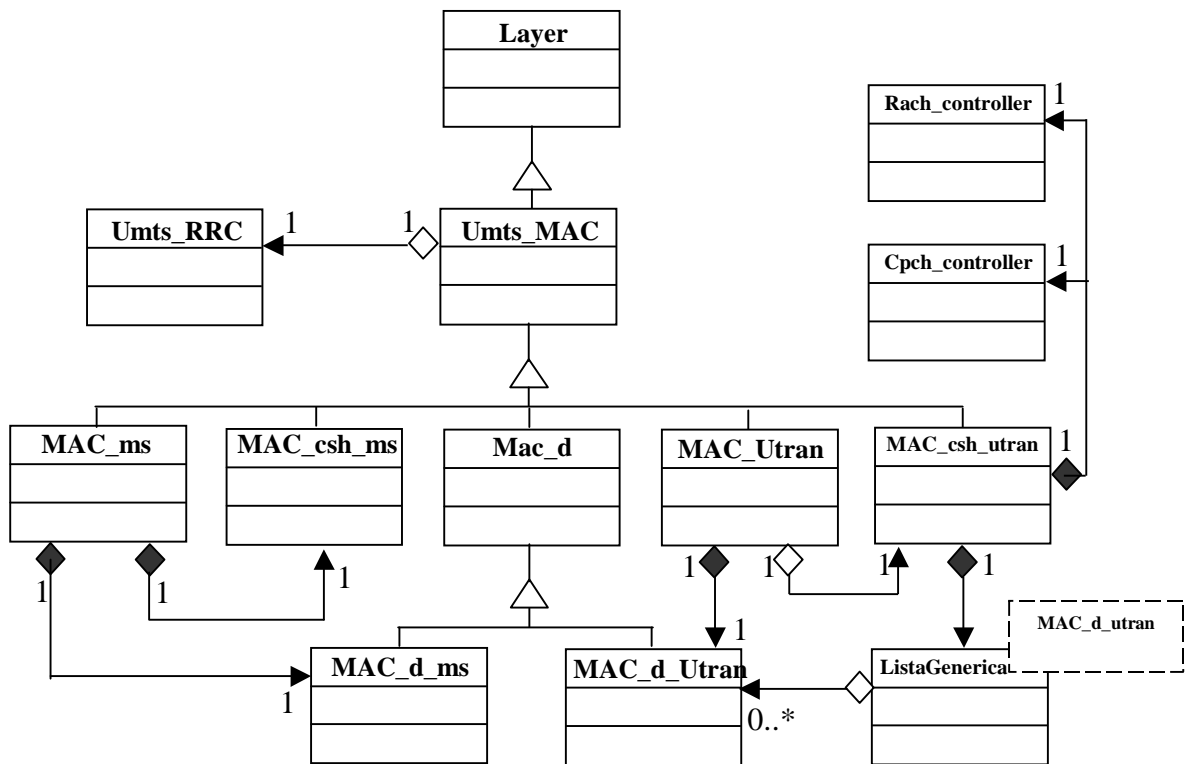


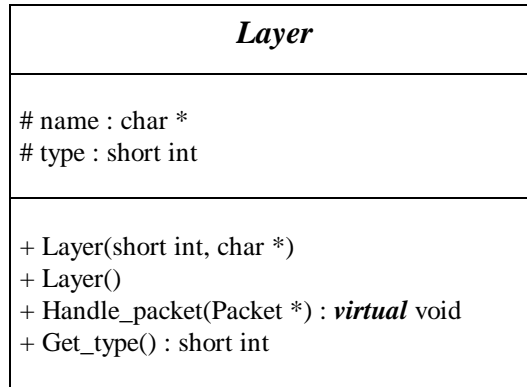
Figura A.6 : Diagramma delle classi

Layer è una classe astratta che dichiara un'interfaccia generica per gestire ogni possibile livello della pila protocollare.

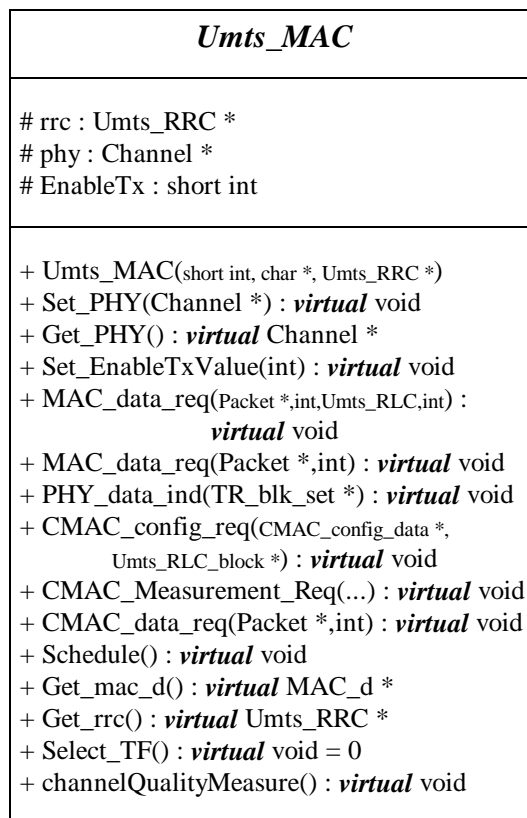
Umts_MAC deriva dalla classe *Layer* e serve per introdurre le caratteristiche comuni a tutte le entità che realizzano il livello MAC nel simulatore. Anche questa è definita come classe fondamentale astratta.

A.1.6.1 Descrizione statica delle classi con dettagli di livello implementativo

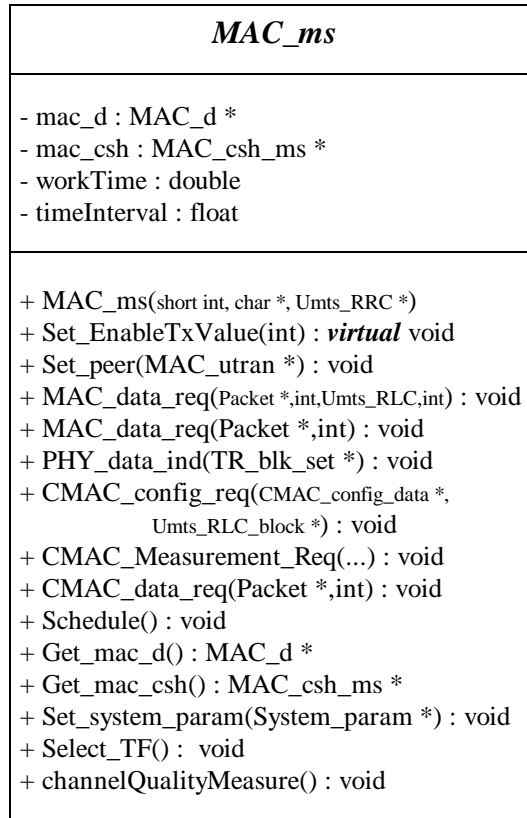
- *Classe Layer:*



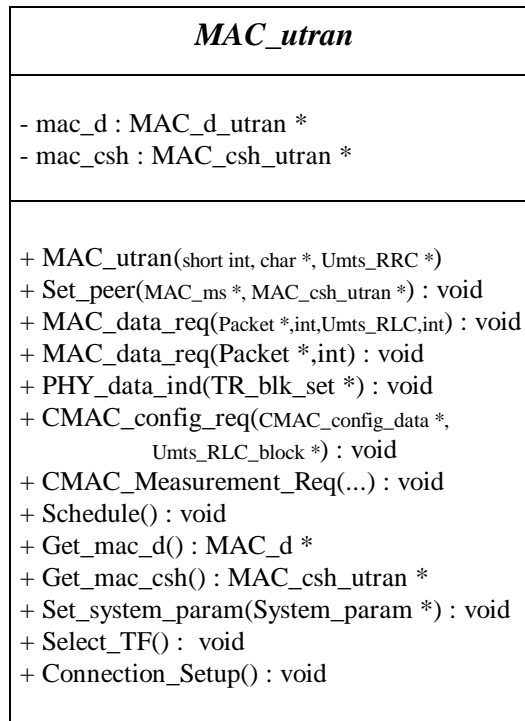
- *Classe Umts_MAC:*



- Classe *MAC_ms*:



- Classe *MAC_utran*:



Viene qui tralasciata una descrizione particolareggiata delle singole entità che compongono i livelli MAC sia dal lato dell'UE sia dal lato dell'UTRAN.

A.1.7 Controller di livello fisico sul RACH e CPCH

Questo modulo illustra le relazioni esistenti tra gli elementi introdotti all'interno dei MAC-c/sh nell'UTRAN, per simulare gli eventi di livello fisico sui canali RACH e CPCH.

Controller è la superclasse che definisce un insieme di metodi virtuali da ridefinire in modo opportuno nelle classi derivate.

Rach_controller è una classe figlia con il compito di simulare e gestire le collisioni sui preamboli e sui transport block set, che sfruttano il RACH come canale di trasporto. Le collisioni sono determinate utilizzando le caratteristiche proprie dei canali slotted-aloha.

Cpch_controller è un'altra classe derivata che ha invece il compito di controllare gli eventi di livello fisico associati al canale di trasporto CPCH. Poichè la trasmissione su quest'ultimo canale è di tipo DSMA-CD, il controller dovrà gestire eventuali collisioni sulle sequenze di preambolo e in caso negativo assegnare un codice dedicato al mobile opportuno.

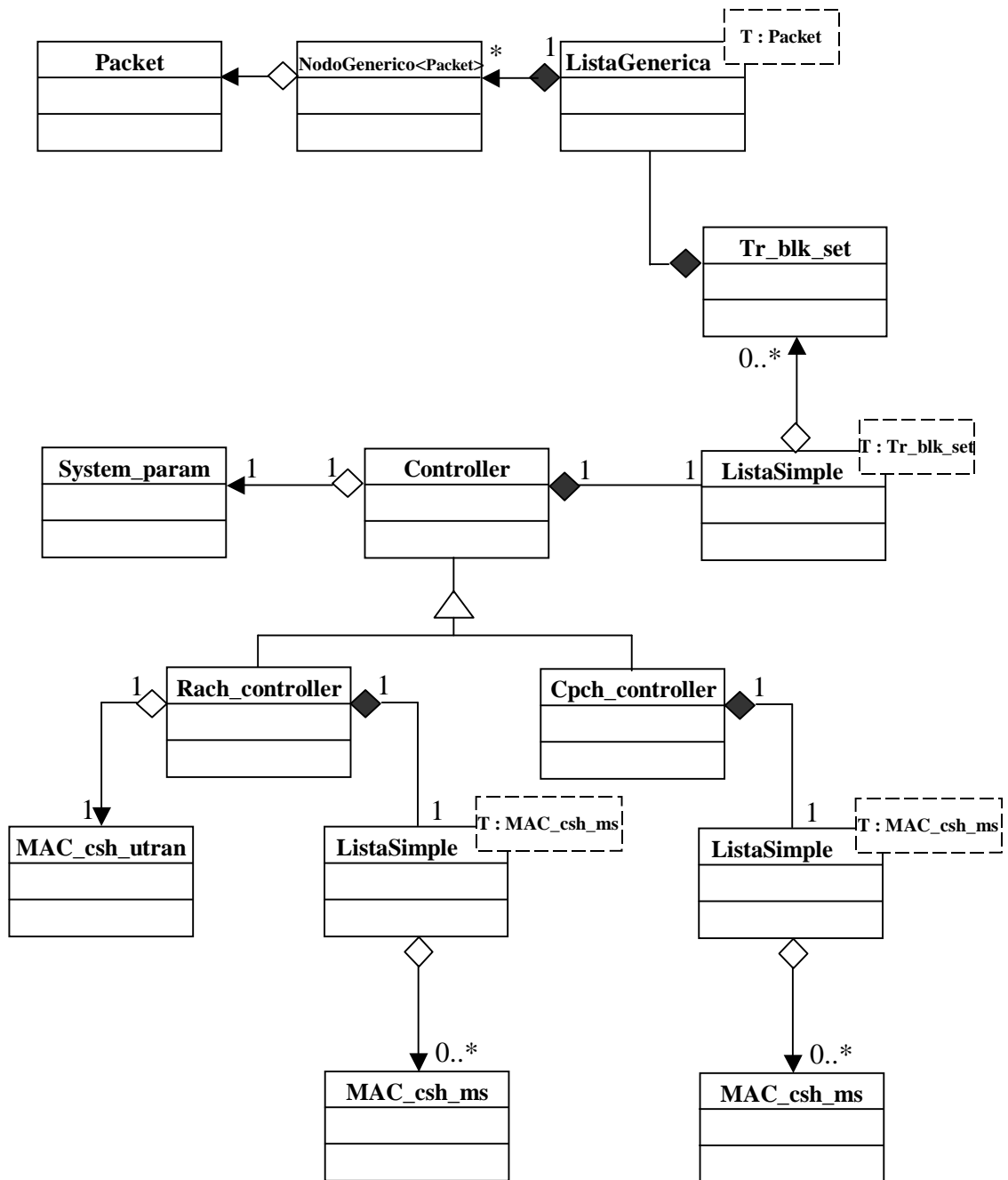
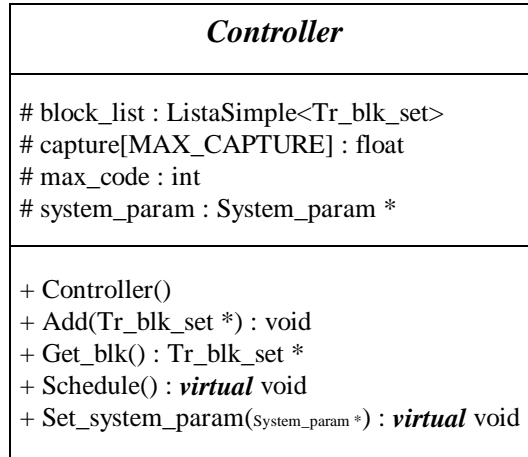


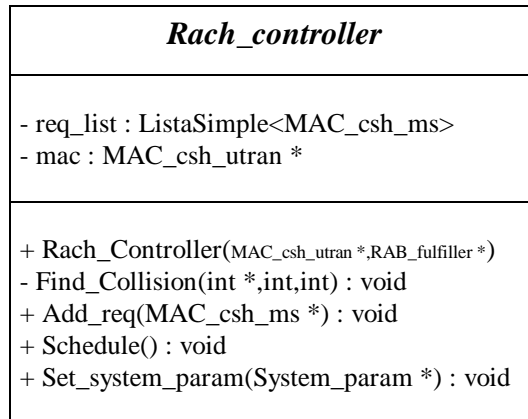
Figura A.7 : Diagramma delle classi

A.1.7.1 Struttura statica delle classi con dettagli di livello implementativo

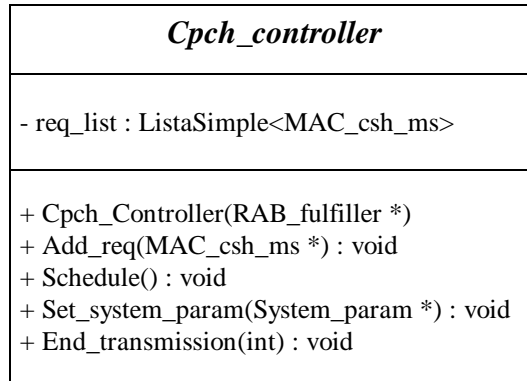
- *Classe Controller:*



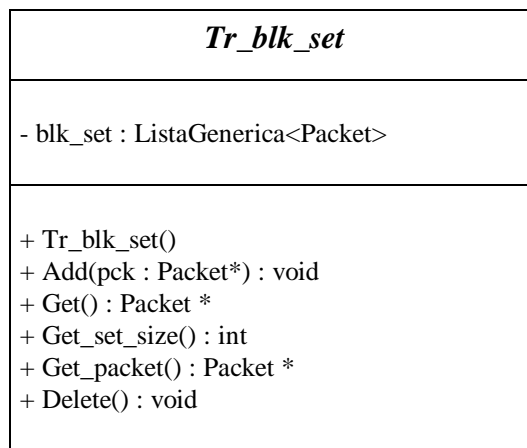
- *Classe RACH_controller:*



- Classe *CPCH_controller*:



- Classe *Tr_blk_set*:



A.1.8 Livello RLC

Il package in figura A.8 rappresenta la struttura di livello RLC.

La classe *Umts_RLC_block* è una classe astratta pensata come un “contenitore” che può raccogliere al suo interno un numero di entità RLC pari al numero massimo di Radio Bearers che si possono instaurare per ogni connessione RRC.

Le classi *Umts_RLC_block_ms* e *Umts_RLC_block_utran* sono le classi figlie per specializzare il livello RLC rispettivamente nel lato UE e nell’UTRAN.

I vantaggi che derivano da una scelta di questo tipo sono:

1. la possibilità di assegnare in modo dinamico diverse modalità di trasferimento per servizi diversi;
2. la possibilità di instaurare più *Radio Bearers* per uno stesso utente.

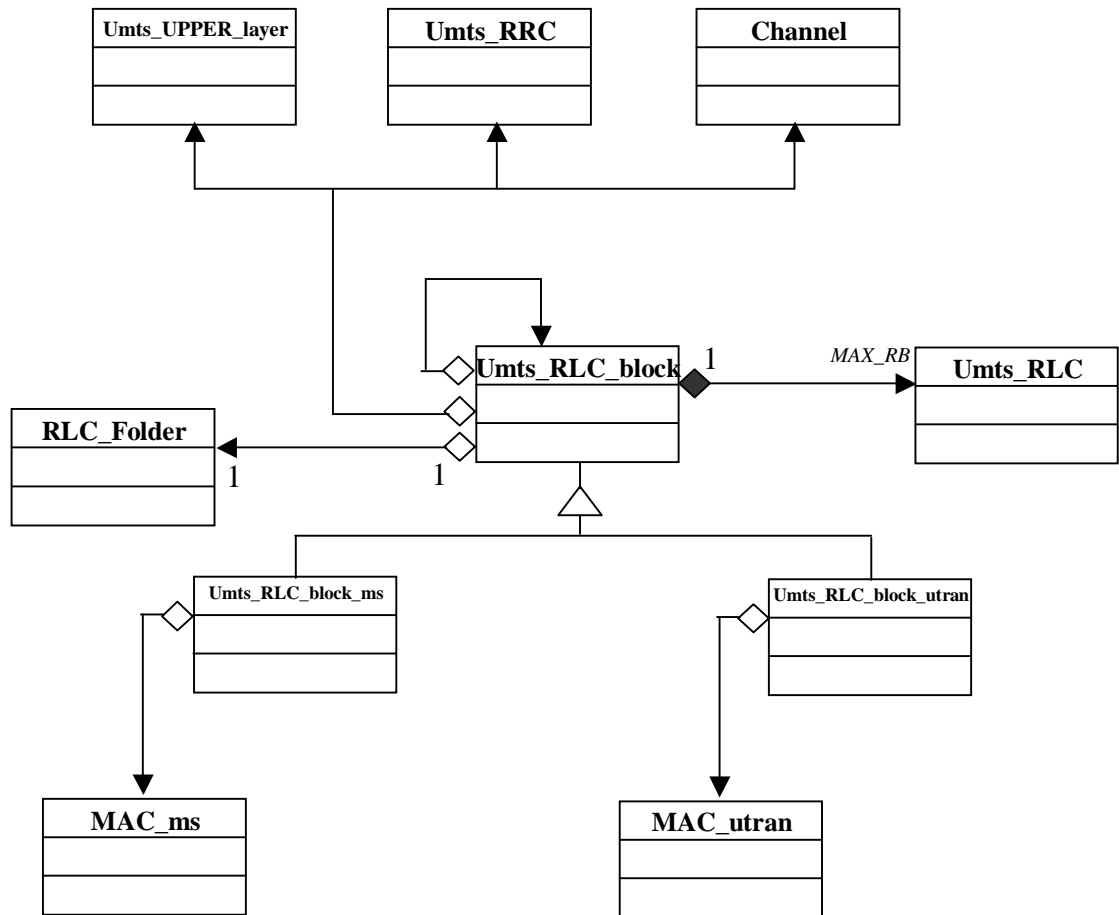


Figura A.8 : Diagramma delle classi del livello RLC

Per quanto riguarda la singola entità RLC, si deve far riferimento al diagramma riportato in figura A.9.

La classe *Umts_RLC* è pensata come classe astratta fondamentale. Essa fornisce un'interfaccia pubblica per l'intera gerarchia di classi derivate.

Le classi *Umts_RLC_TrD*, *Umts_RLC_UMD* e *Umts_RLC_AMD* sono istanze particolari della classe *Umts_RLC* e servono per definire rispettivamente la modalità di trasferimento transparent, unacknowledged e acknowledged del livello RLC.

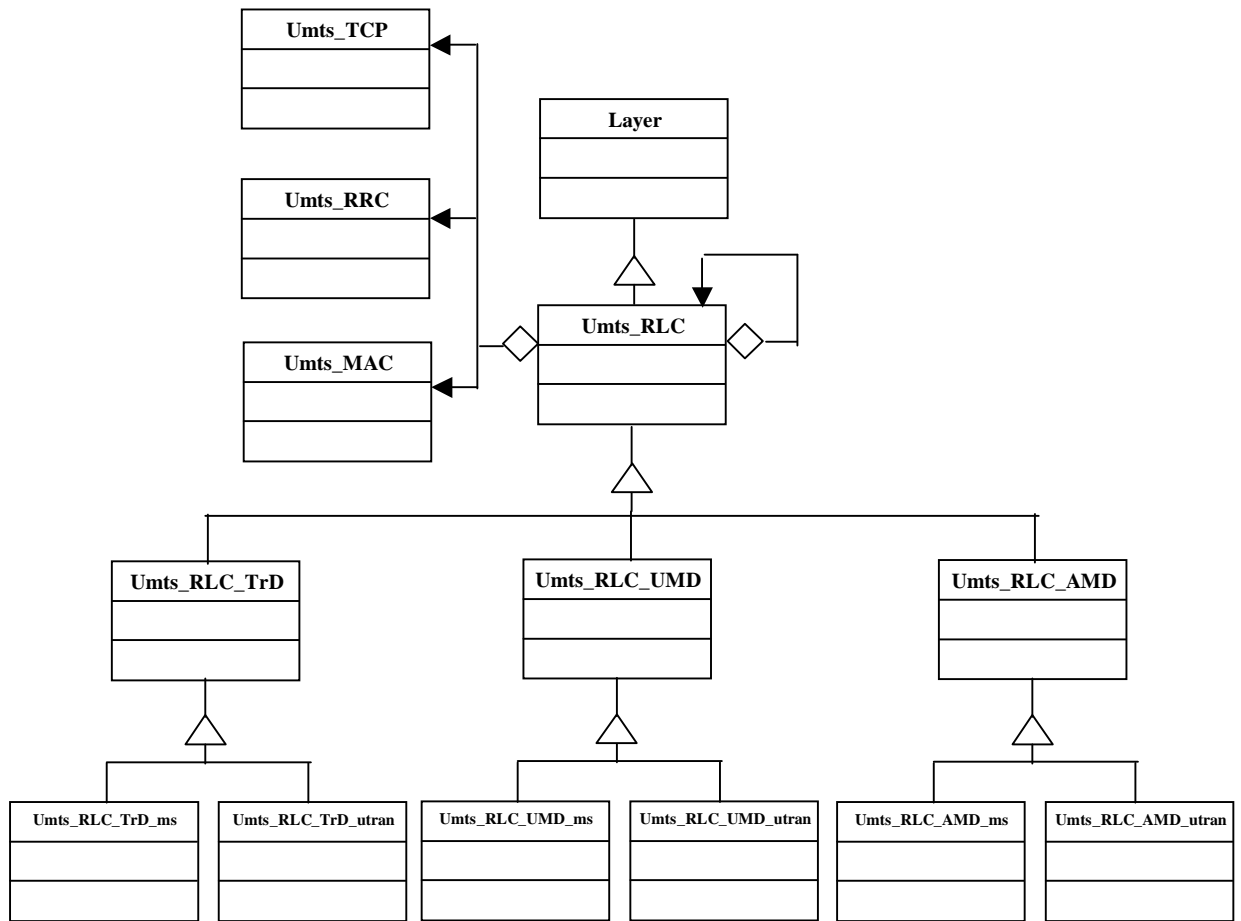
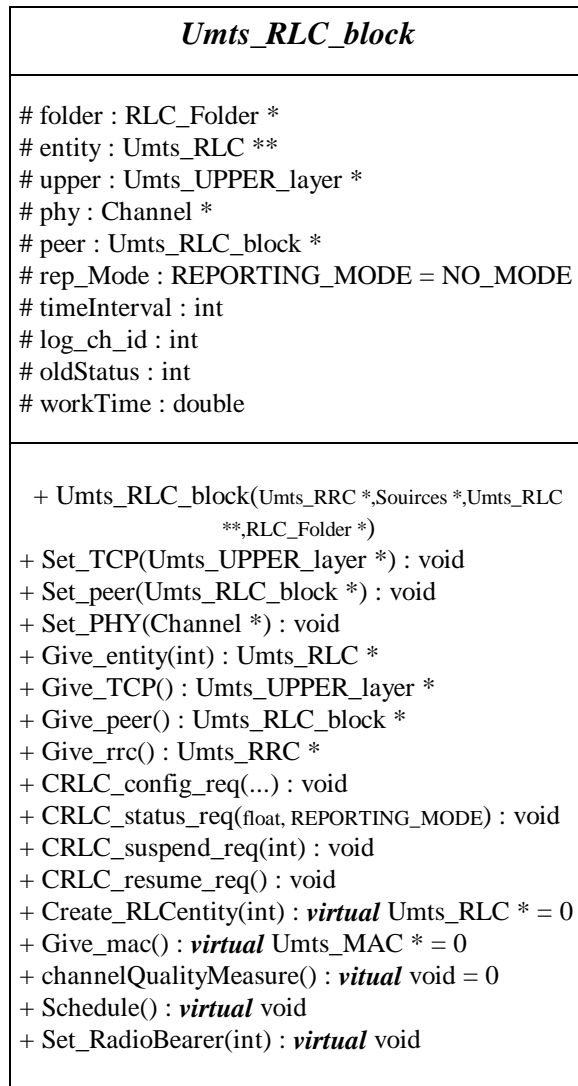


Figura A.9 : Diagramma delle classi della singola entità RLC

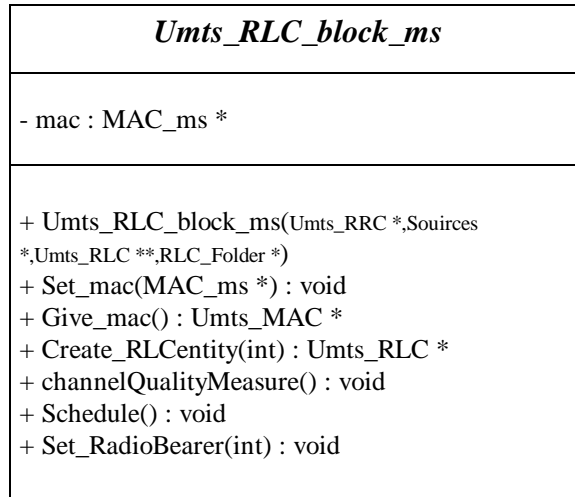
A.1.8.1 Descrizione statica delle classi con dettagli di livello implementativo

In questo sotto-paragrafo si descrive la struttura statica della gerarchia di classi legate al blocco *Umts_RLC_block*, mentre non vengono trattate le varie entità che possono trovarsi all'interno del livello RLC.

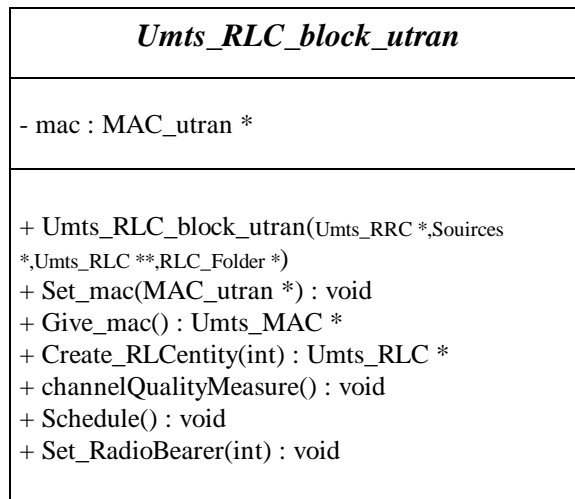
- Classe *Umts_RLC_block*:



- Classe *Umts_RLC_block_ms*:



- Classe *Umts_RLC_block_utran*:



A.1.9 Livello RRC

Il package in figura A.10 visualizza le entità che compongono il livello RRC.

Umts_RRC è una classe astratta da cui derivano due classi figlie che caratterizzano il livello RRC nel lato UE e nell'UTRAN.

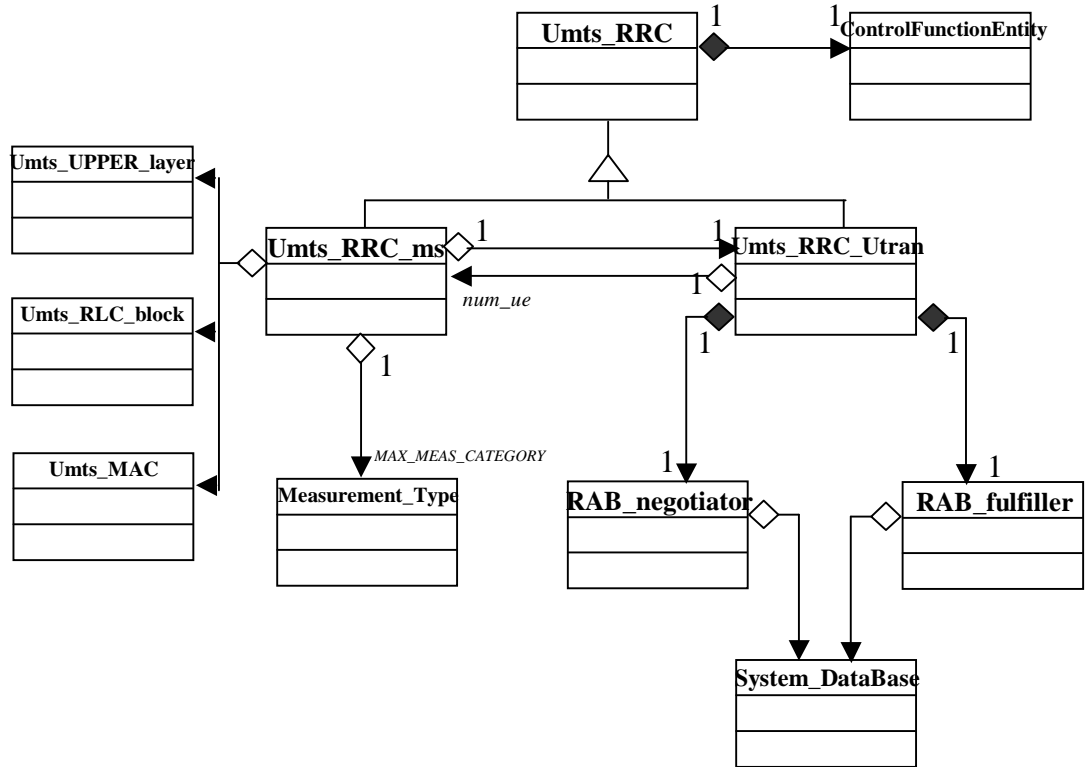


Figura A.10 : Diagramma delle classi del livello RRC

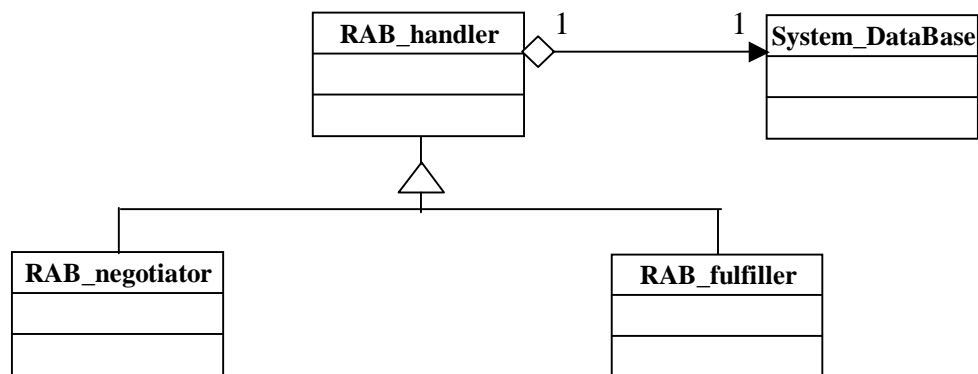


Figura A.11 : Diagramma delle classi del gestore dei RAB

In figura A.11 è rappresentata la relazione tra le entità che devono gestire i *Radio Access Bearers* durante tutto il periodo della connessione RRC. Si distingue una classe padre astratta da cui derivano due classi figlie.

In figura A.12 viene illustrata la struttura del package relativo al DataBase di sistema, che viene usato dai gestori del RAB per assegnare i codici e ricavare tutte le informazioni relative alla configurazione dei livelli delle pile protocollari.

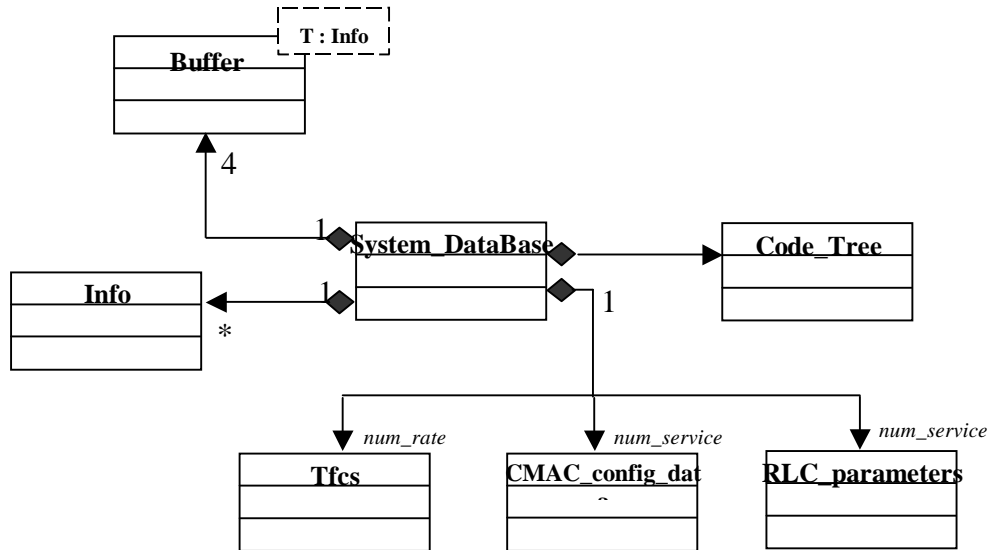


Figura A.12 : Diagramma delle classi del DataBase di sistema

Dall'analisi delle specifiche sul livello RRC, tratte dal documento *3GPP TS 25.331 v4.1.0* redatto dall'organismo di standardizzazione *3rd Generation Partnership Project*, è sembrato opportuno realizzare tale livello mediante il *pattern Command*. Questa soluzione è giustificata se si tengono presenti le caratteristiche peculiari del pattern citato (e riportate per comodità nell'elenco qui sotto):

- i. disaccoppiamento tra l'oggetto che invoca l'operazione da quello che la deve eseguire;
- ii. possibilità di inviare richieste a degli oggetti, senza avere alcuna informazione sull'oggetto a cui deve essere inviata la richiesta o sul tipo di richiesta stesso;
- iii. possibilità di creare comandi completamente indipendenti dalle classi esistenti, implementando ogni cosa nel comando stesso senza delegare nulla alle altre entità;
- iv. semplicità nell'aggiungere nuovi *Command* senza modificare quanto già esiste;
- v. possibilità di creare una *history list* dei comandi che sono stati o devono ancora essere eseguiti.

L'idea base di questo pattern è quindi quella di incapsulare ogni richiesta in un oggetto. Le entità fondamentali che intervengono in questa architettura sono:

- *Command* : dichiara un'interfaccia per eseguire un'operazione;
- *Concrete Command* : ad un estremo definisce solo un legame tra l'oggetto Receiver e un'azione. All'altro implementa ogni cosa lui stesso senza affidare nulla al ricevitore;
- *Client (Application)* : crea un oggetto Concrete Command e setta il ricevitore;
- *Invoker* : richiede il comando per eseguire la richiesta;
- *Receiver (Application)* : può conoscere o meno il modo di eseguire le operazioni associate alla richiesta.

In figura A.13 si rappresenta il diagramma delle classi che realizza il pattern Command. In particolare si evidenzia la corrispondenza tra classi del simulatore e classi generiche proprie del pattern in esame.

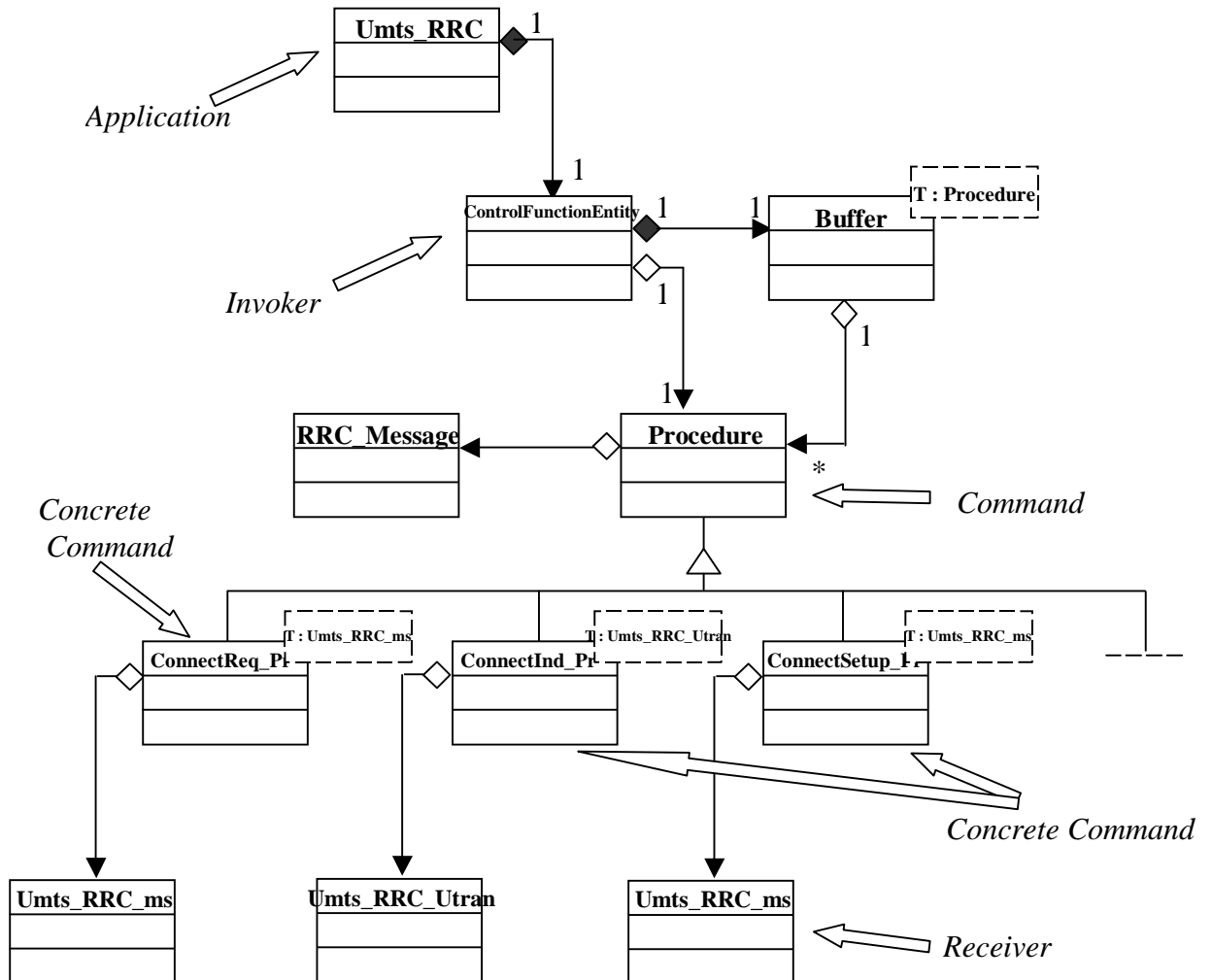


Figura A.13 : Diagramma delle classi del pattern Command

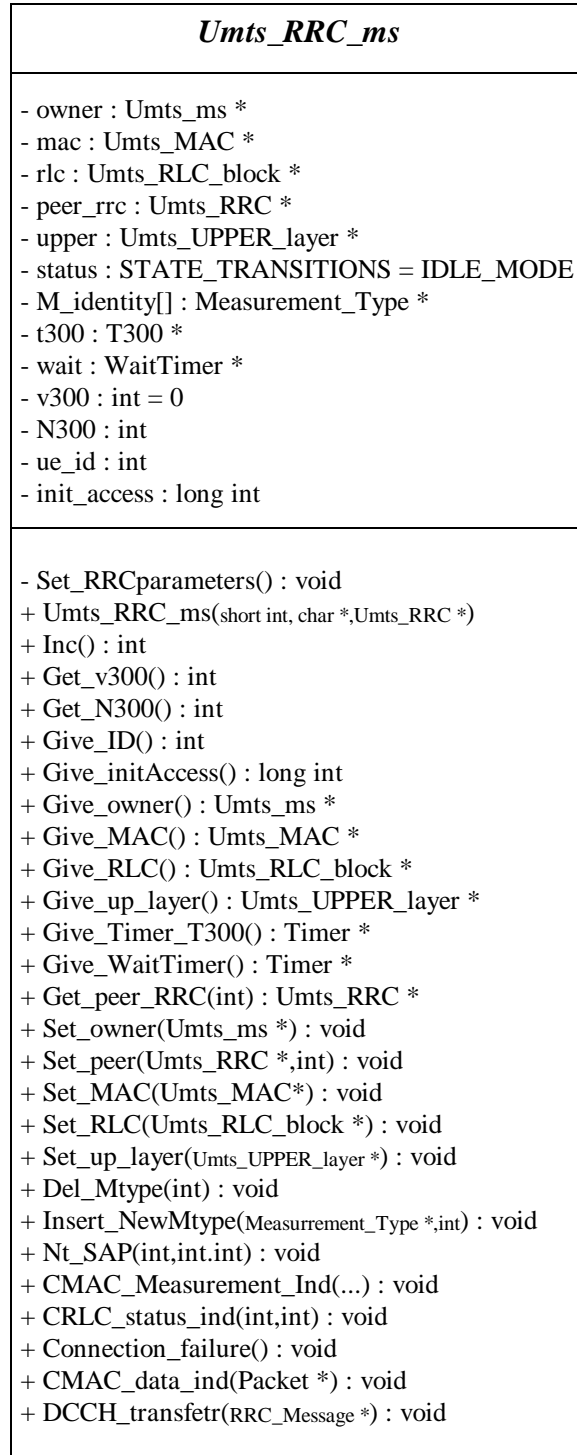
A.1.9.1 Descrizione statica delle classi con dettagli di livello implementativo

Di tutti i possibili Concrete Command definiti all'interno del simulatore, ne verrà analizzato uno solo come esempio.

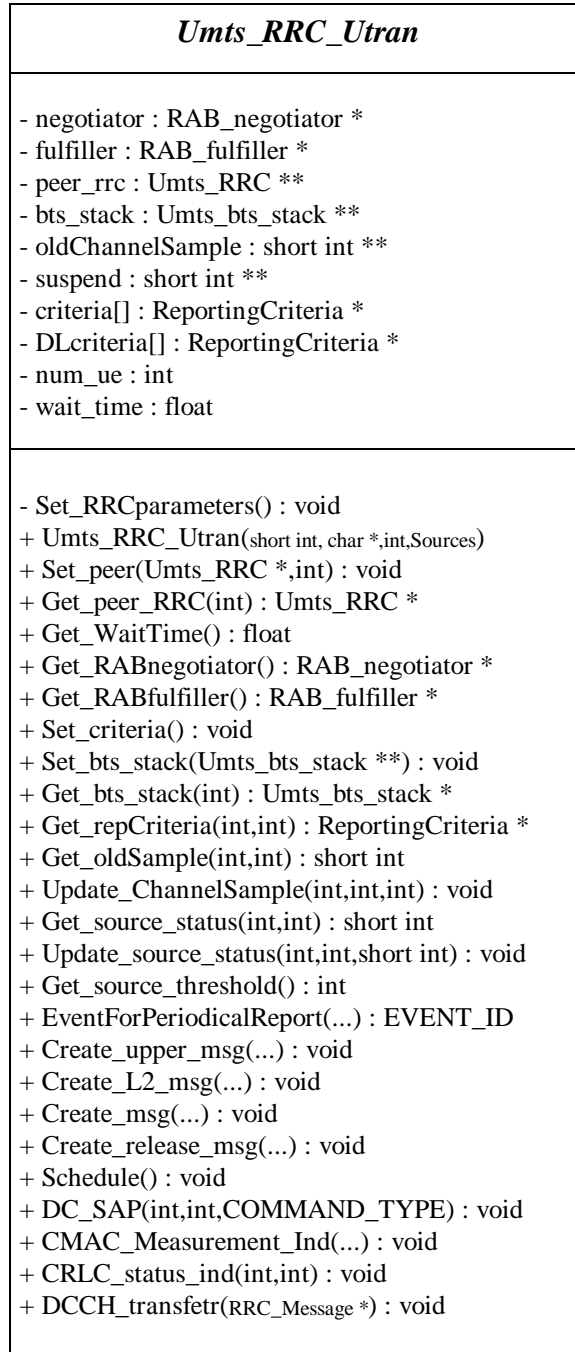
- Classe *Umts_RRC*:

<i>Umts_RRC</i>
CFE : ControlFunctionEntity *
+ Umts_RRC(short int, char *) + GivePerformer() : ControlFunctionEntity * + Schedule() : <i>virtual</i> void + Set_owner(Umts_ms *) : <i>virtual</i> void + Give_owner() : <i>virtual</i> Umts_ms * + Set_MAC(Umts_MAC*) : <i>virtual</i> void + Set_RLC(Umts_RLC_block *) : <i>virtual</i> void + Set_up_layer(Umts_UPPER_layer *) : <i>virtual</i> void + Nt_SAP(int,int.int) : <i>virtual</i> void + GC_SAP(int,int) : <i>virtual</i> void + DC_SAP(int,int,COMMAND_TYPE) : <i>virtual</i> void + CMAC_Measurement_Ind(...) : <i>virtual</i> void + CRLC_status_ind(int,int) : <i>virtual</i> void + CMAC_data_ind(Packet *) : <i>virtual</i> void = 0 + DCCH_transfetr(RRC_Message *) : <i>virtual</i> void = 0 + Give_ID() : <i>virtual</i> int + Get_RABnegotiator() : <i>virtual</i> RAB_negotiator * + Get_RABfulfiller() : <i>virtual</i> RABfulfiller * + Connection_failure() : <i>virtual</i> void + Set_peer(Umts_RRC *,int) : <i>virtual</i> void = 0 + Get_peer_RRC(int) : <i>virtual</i> Umts_RRC * = 0

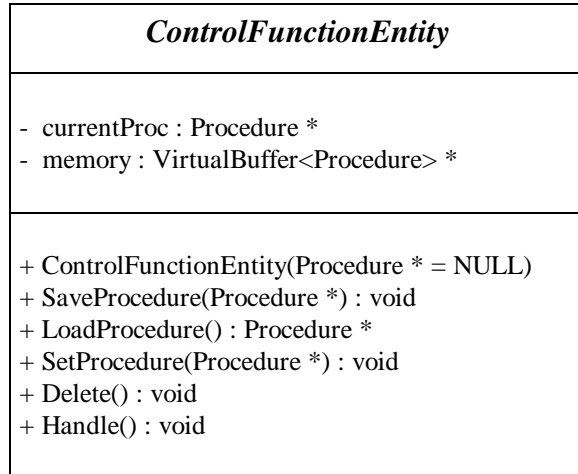
- Classe *Umts_RRC_ms*:



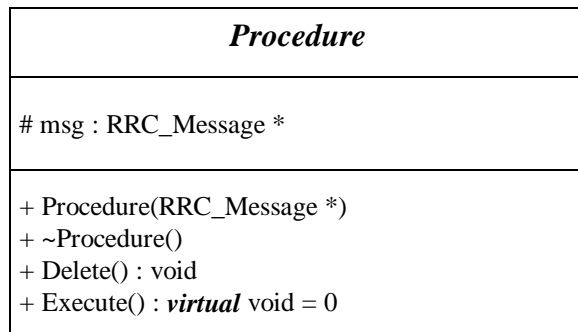
- Classe *Umts_RRC_Utran*:



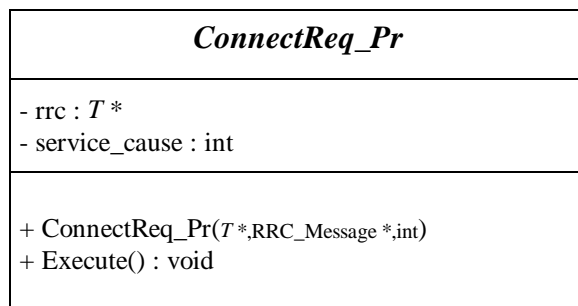
- *Classe ControlFunctionEntity:*



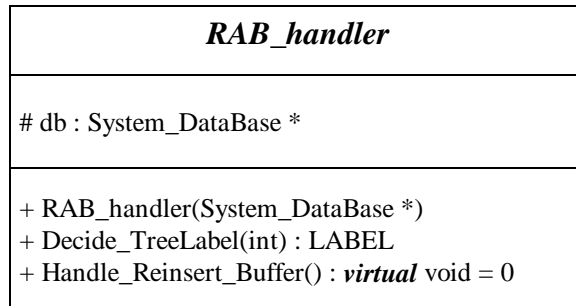
- *Classe Procedure:*



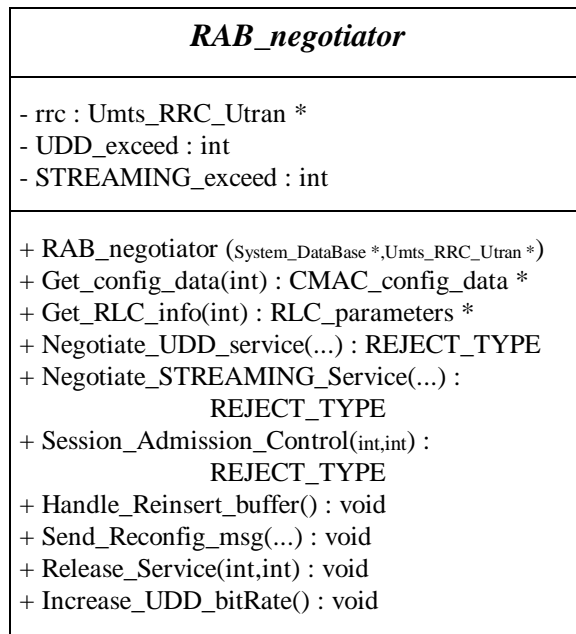
- *Classe ConnectReq_Pr:*



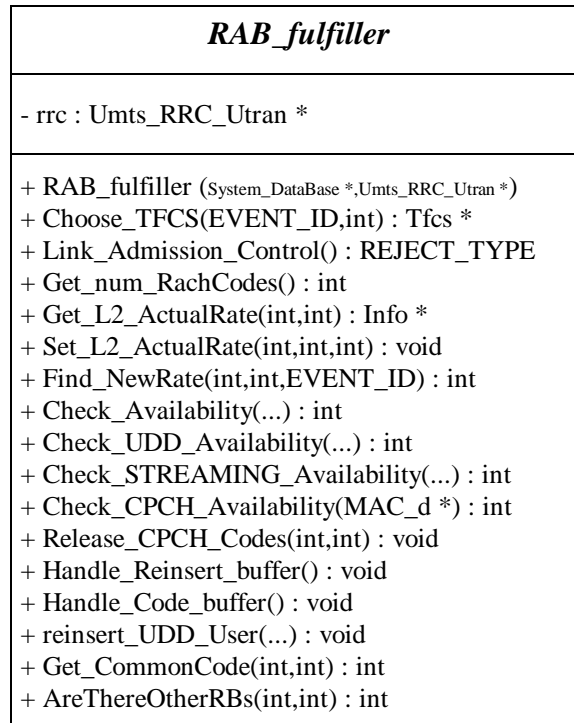
- Classe *RAB_handler*:



- Classe *RAB_negotiator*:



- Classe *RAB_fulfiller*:



A.1.10 Livello TCP/UDP

Il diagramma di figura A.14 rappresenta le relazioni di aggregazione e generalizzazione-specializzazione tra quelle classi che realizzano il livello chiamato genericamente *UPPER layer*.

La classe *Umts_UPPER_layer* è la classe fondamentale astratta. Essa fornisce un'interfaccia pubblica per gestire tutte le classi derivate.

Le classi *Umts_UPPER_layer_ms* e *Umts_UPPER_layer_utran* specializzano il comportamento del livello in esame rispettivamente dal lato UE e dal lato dell'UTRAN.

UPPER_layer è pensato come un livello che giace sia nel piano di controllo sia nel piano utente, in modo da interagire e con il livello RRC e con il livello RLC. Inoltre esso può contenere tante entità *TCP e/o UDP* quanti sono i Radio Bearers instaurabili per ogni UE. E' inoltre importante sottolineare la corrispondenza uno a uno tra queste entità e le entità RLC sottostanti.

Questa scelta è strettamente legata alla modellizzazione del livello RLC e consente di simulare contemporaneamente i protocolli di livello TCP e UDP, anche nello stesso utente.

L'entità *UPPER_Folder* verrà analizzata nei particolari al paragrafo A.1.11.

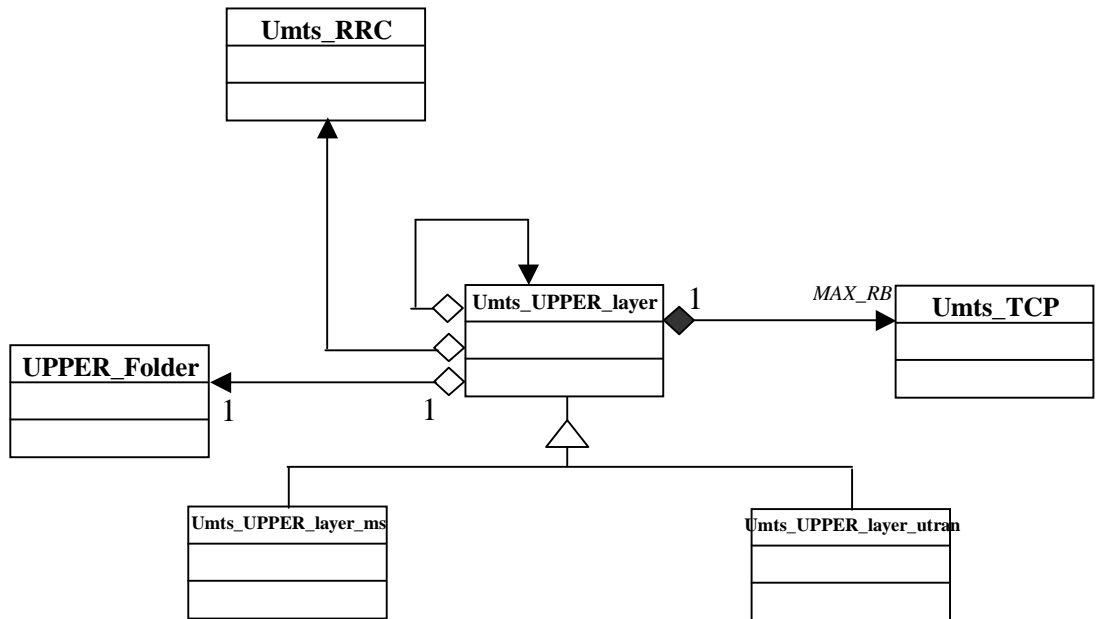


Figura A.14 : Diagramma delle classi del livello 4

Per quanto riguarda le singole entità TCP/UDP che possono trovarsi in questo livello, si deve far riferimento alla figura A.15.

Si distingue anche qui una super-classe denominata *Umts_TCP* da cui derivano le classi figlie *Umts_TCP_ms* e *Umts_TCP_Utran*.

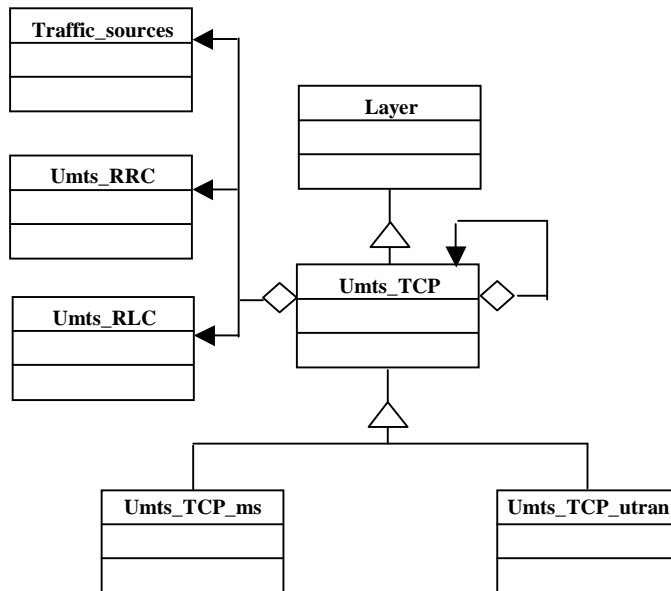
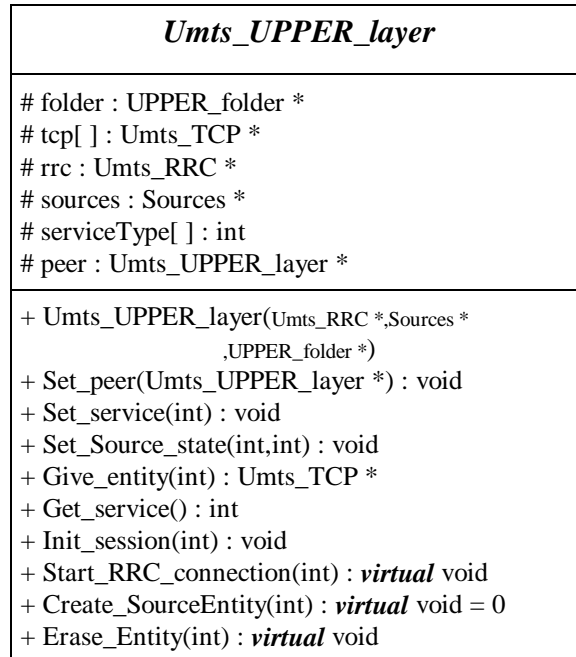


Figura A.15 : Diagramma delle classi delle entità TCP/UDP

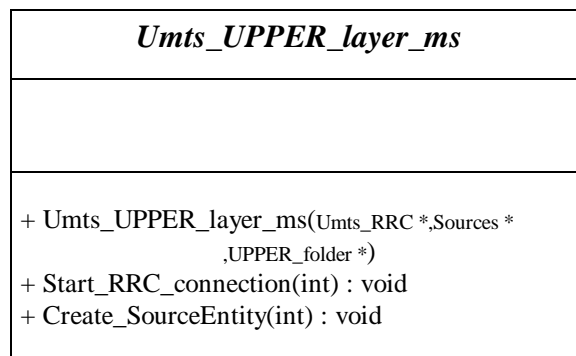
A.1.10.1 Descrizione statica delle classi con dettagli di livello implementativo

In questo sotto-paragrafo non verrà analizzata la struttura statica delle singole entità TCP/UDP, ma solo del livello UPPER_layer che le contiene.

- Classe *Umts_UPPER_layer*:



- Classe *Umts_UPPER_ms*:



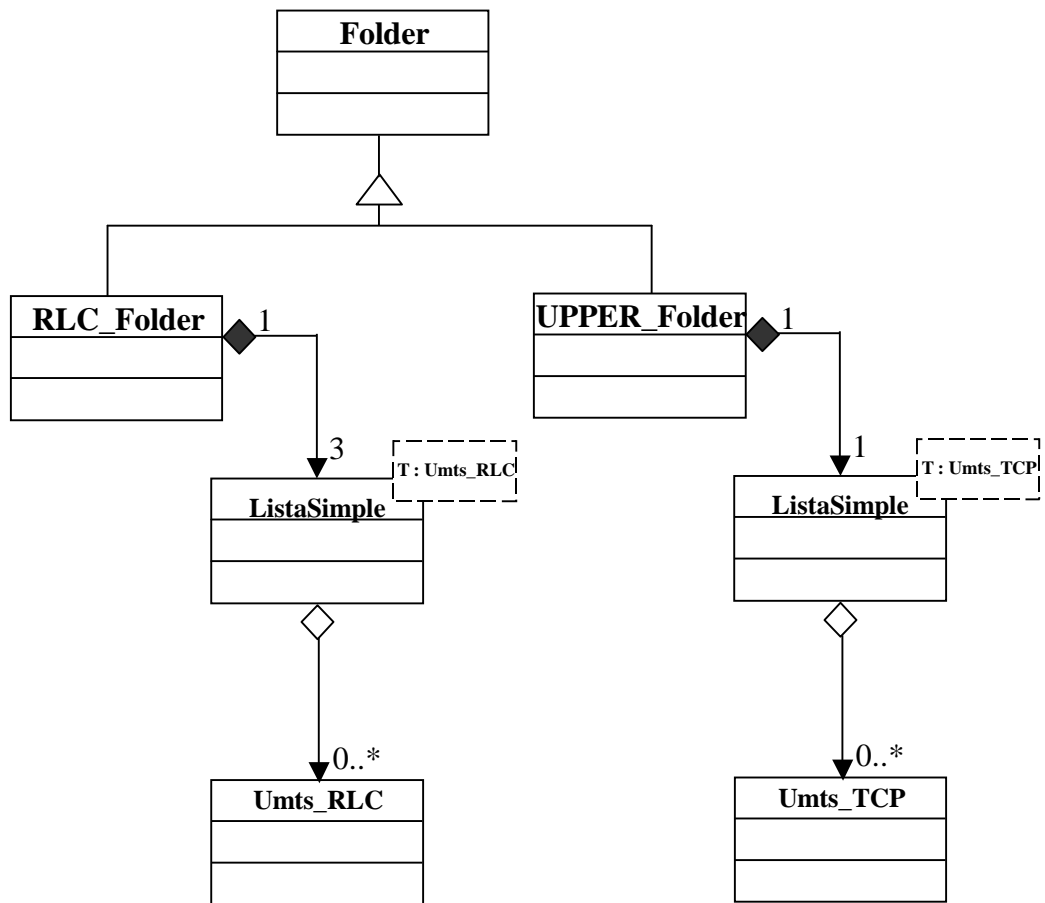
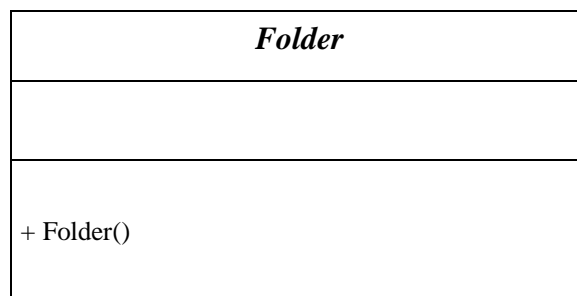


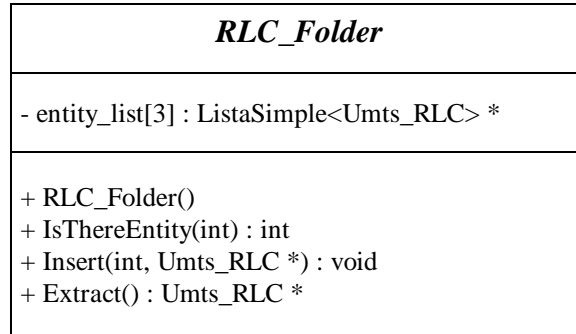
Figura A.16 : Diagramma delle classi

A.1.11.1 Descrizione statica delle classi con dettagli di livello implementativo

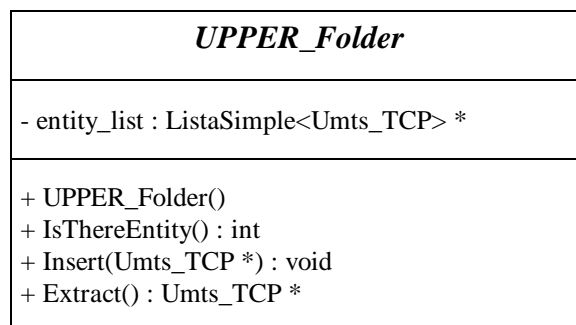
- *Classe Folder:*



- Classe *RLC_Folder*:



- Classe *UPPER_Folder*:



A.1.12 Pacchetti Dati, Header e Messaggi di Controllo

Il diagramma di figura A.17 illustra come sono strutturati il generico pacchetto dati di un certo livello nel simulatore e la struttura dei messaggi di controllo gestiti dalle entità RRC.

Il *pacchetto dati* è definito come una lista di *header* (intestazioni), nei quali vengono memorizzate le informazioni proprie dei vari livelli. Ogni volta che un pacchetto passa da uno strato ad un altro della pila protocollare, viene manipolato da procedure appropriate che possono portare all'aggiunta o alla rimozione delle intestazioni, alla sua segmentazione o riassetto, alla lettura di campi presenti nell'header opportuno. Il pacchetto non contiene uno spazio proprio per il campo dati, ma della sua presenza se ne tiene conto tramite una variabile nell'intestazione, che memorizza la lunghezza dei dati.

I *messaggi di controllo* sono invece definiti come header specializzati che vengono scambiati tra le pari entità di livello RRC.

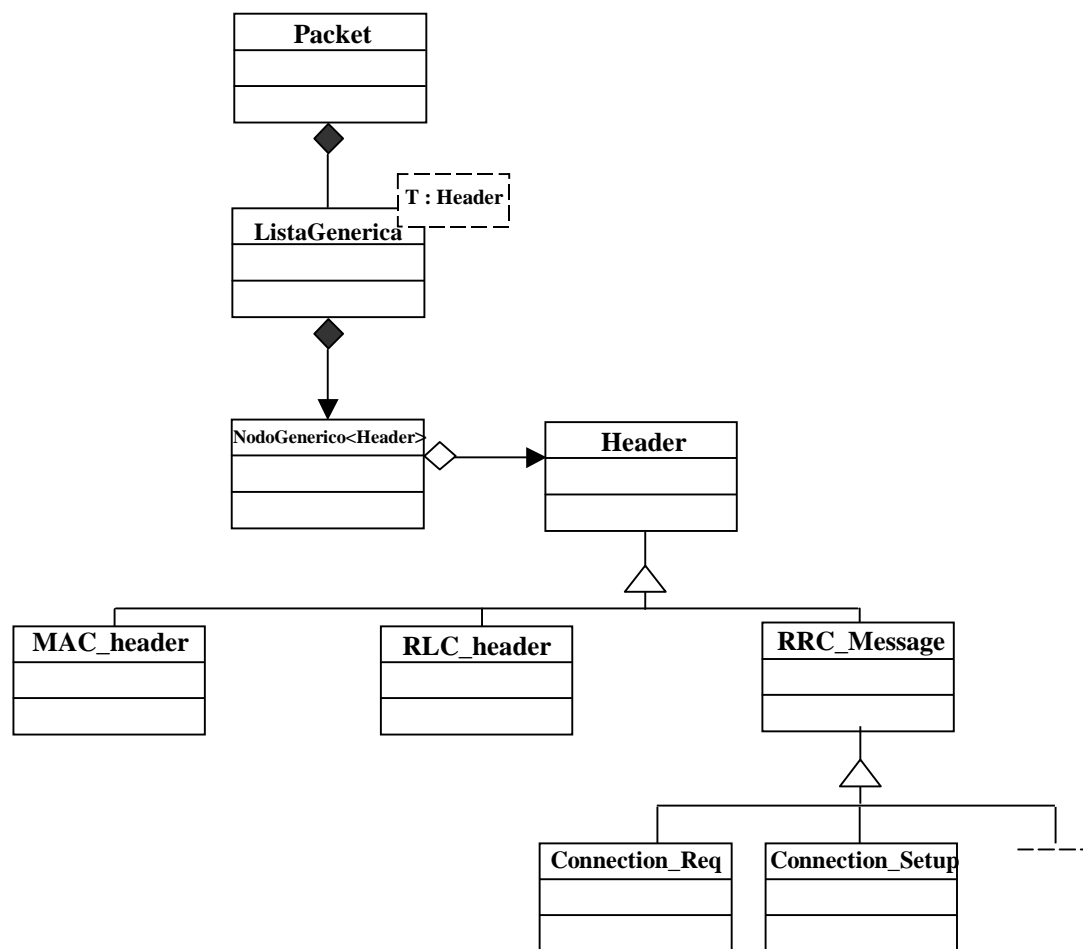


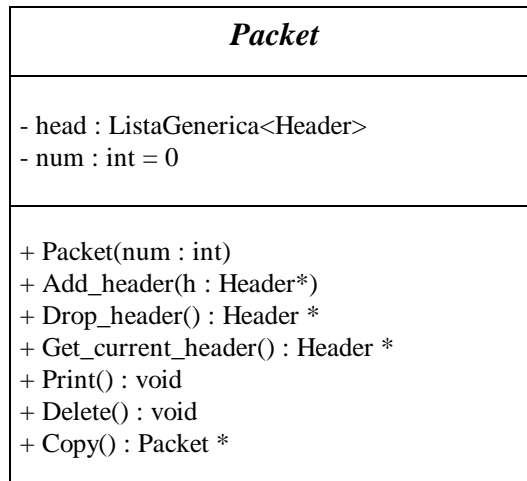
Figura A.17 : Diagramma delle classi

I messaggi di controllo di livello RRC sono qui sotto riportati:

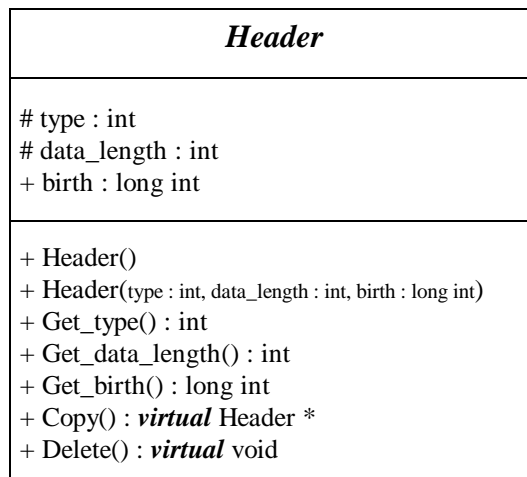
- Connection Request;
- Connection Setup;
- Connection Setup Complete;
- Connection Reject;
- Connection Release;
- Connection Release Complete;
- Measurement Control;
- Measurement Report;
- Radio Bearer Reconfiguration;
- Radio Bearer Reconfiguration Complete;
- Change Source State;
- Source Activity.

A.1.12.1 Descrizione statica delle classi con dettagli di livello implementativo

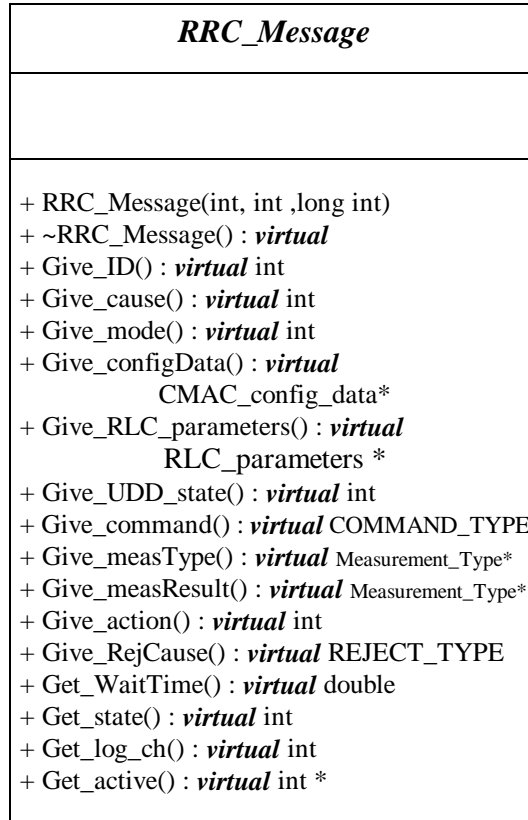
- *Classe Packet:*



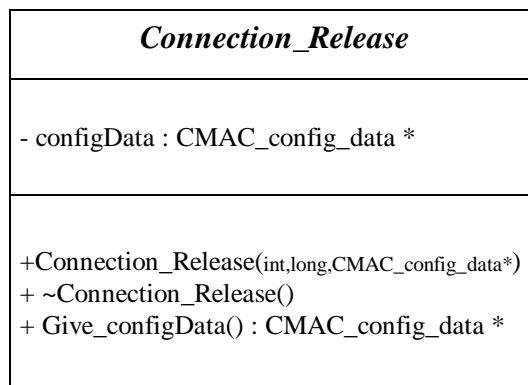
- *Classe Header:*



- Classe *RRC_Message*:



- Classe *Connection_Release*:



A.1.13 Sorgenti per servizi Streaming e Interactive

Il package A.18 mostra il diagramma di quelle classi che devono simulare le sorgenti di traffico Streaming e Interactive.

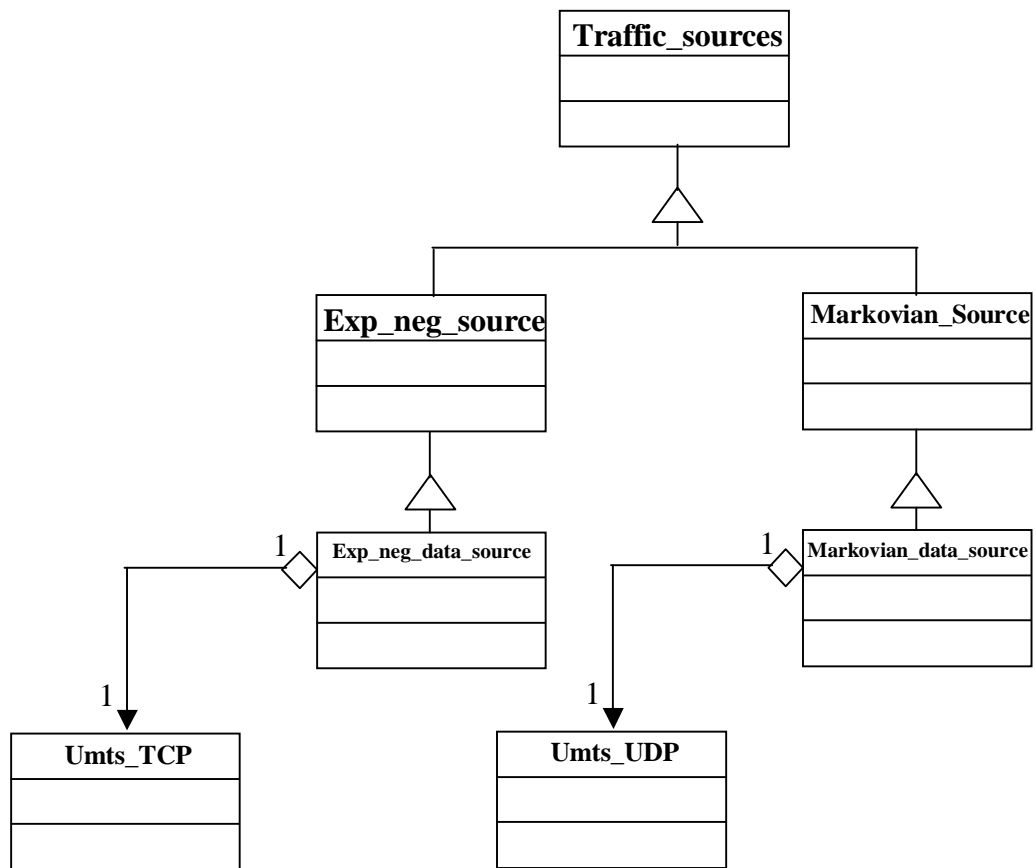
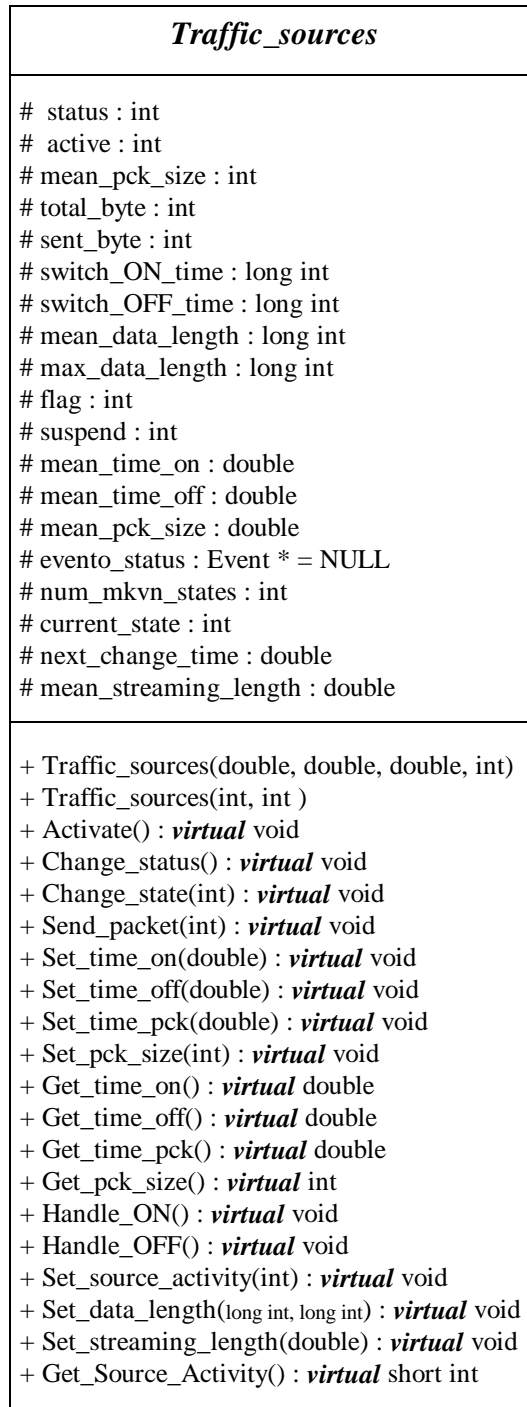


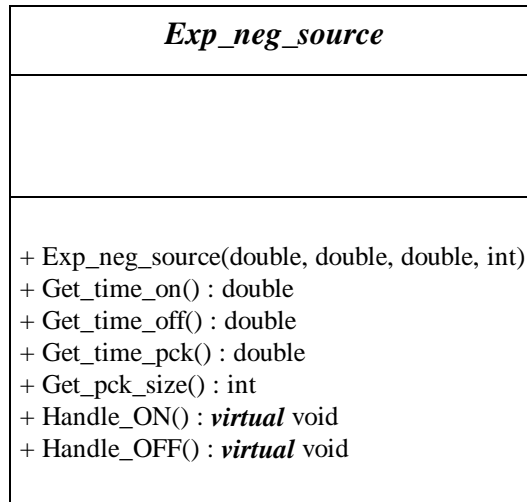
Figura A.18 : Diagramma delle classi

A.1.13.1 Descrizione statica delle classi con dettagli di livello implementativo

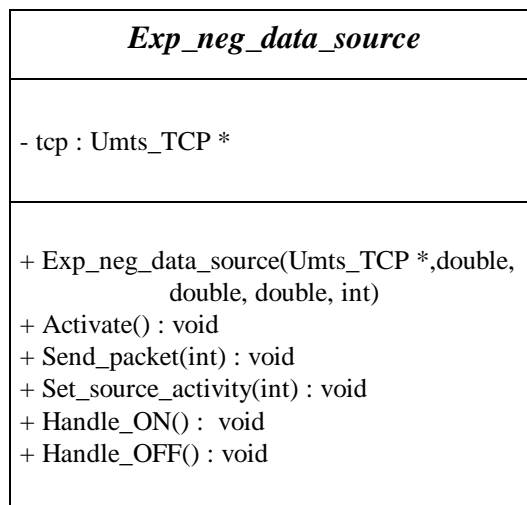
- Classe *Traffic_sources*:



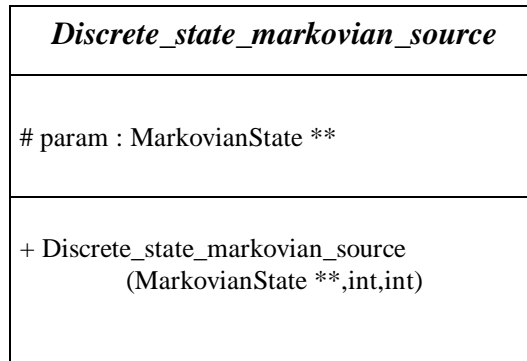
- Classe *Exp_neg_source*:



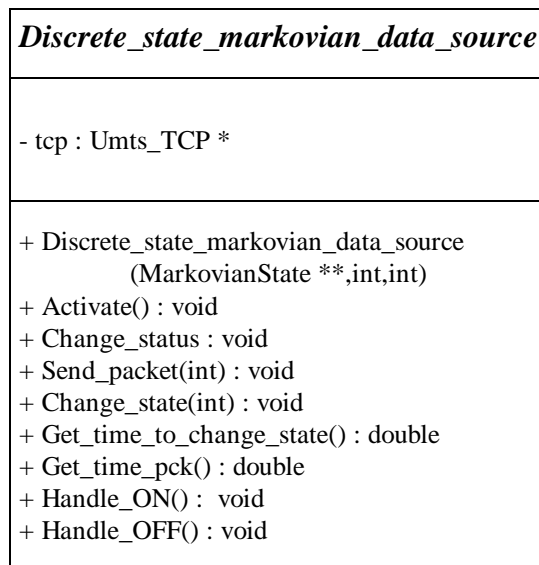
- Classe *Exp_neg_data_source*:



- Classe *Discrete_state_markovian_source*:



- Classe *Discrete_state_markovian_data_source*:



A.2 Diagrammi di sequenza

Per completare la descrizione del progetto, si deve associare alla strutturazione statica, definita al paragrafo A.1, un modello dinamico capace di rappresentare le dinamiche inter-oggetto.

Per questo scopo si introducono i diagrammi di sequenza. Essi mostrano come avviene un'interazione tra diverse entità, evidenziando le classi partecipanti e i messaggi scambiati nel tempo.

In seguito all'elevato numero di classi presenti nel simulatore e all'ancor maggiore numero di interazioni tra le entità, si ritiene opportuno soffermare l'attenzione su quei diagrammi che illustrano il simulatore nei suoi aspetti più generali.

A.2.1 Gestione di eventi asincroni

In figura è riportato un diagramma di interazione relativo alla gestione degli eventi nel simulatore.

Il programma principale, dopo aver verificato l'esistenza di eventi nel frame corrente, li estrae dallo *Scheduler* e ordina ad ognuno di essi di eseguire l'azione per cui è stato creato.

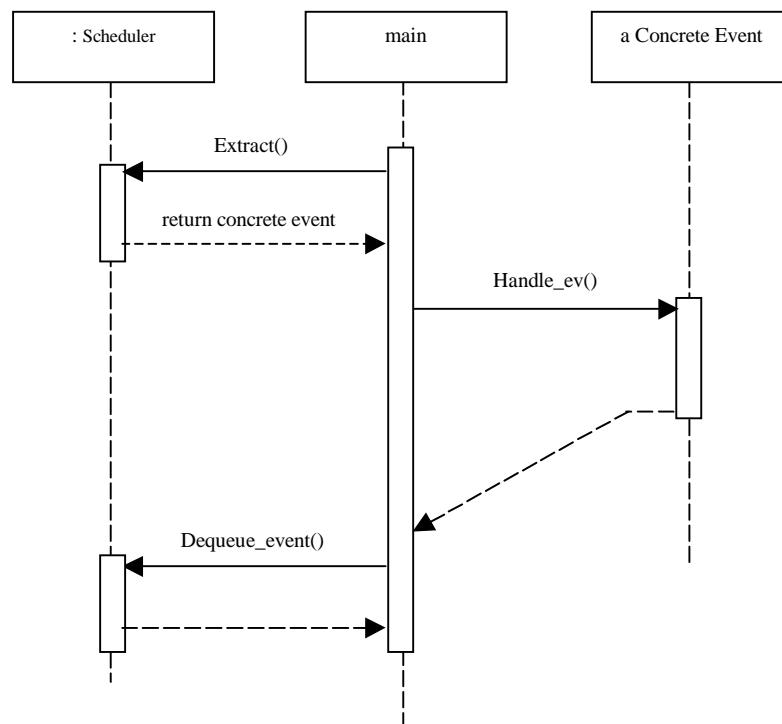


Figura A.19 : Diagramma di sequenza

Data la molteplicità di eventi, si analizzano i casi particolari in cui gli eventi concreti da gestire, dei quali è stato invocato il metodo *Handle_ev*, siano *Init_session_ev* e *Phy_acc_ev*.

A.2.1.1 Init_session_ev

Questo evento è associato alla richiesta di un determinato UE di instaurare una connessione RRC.

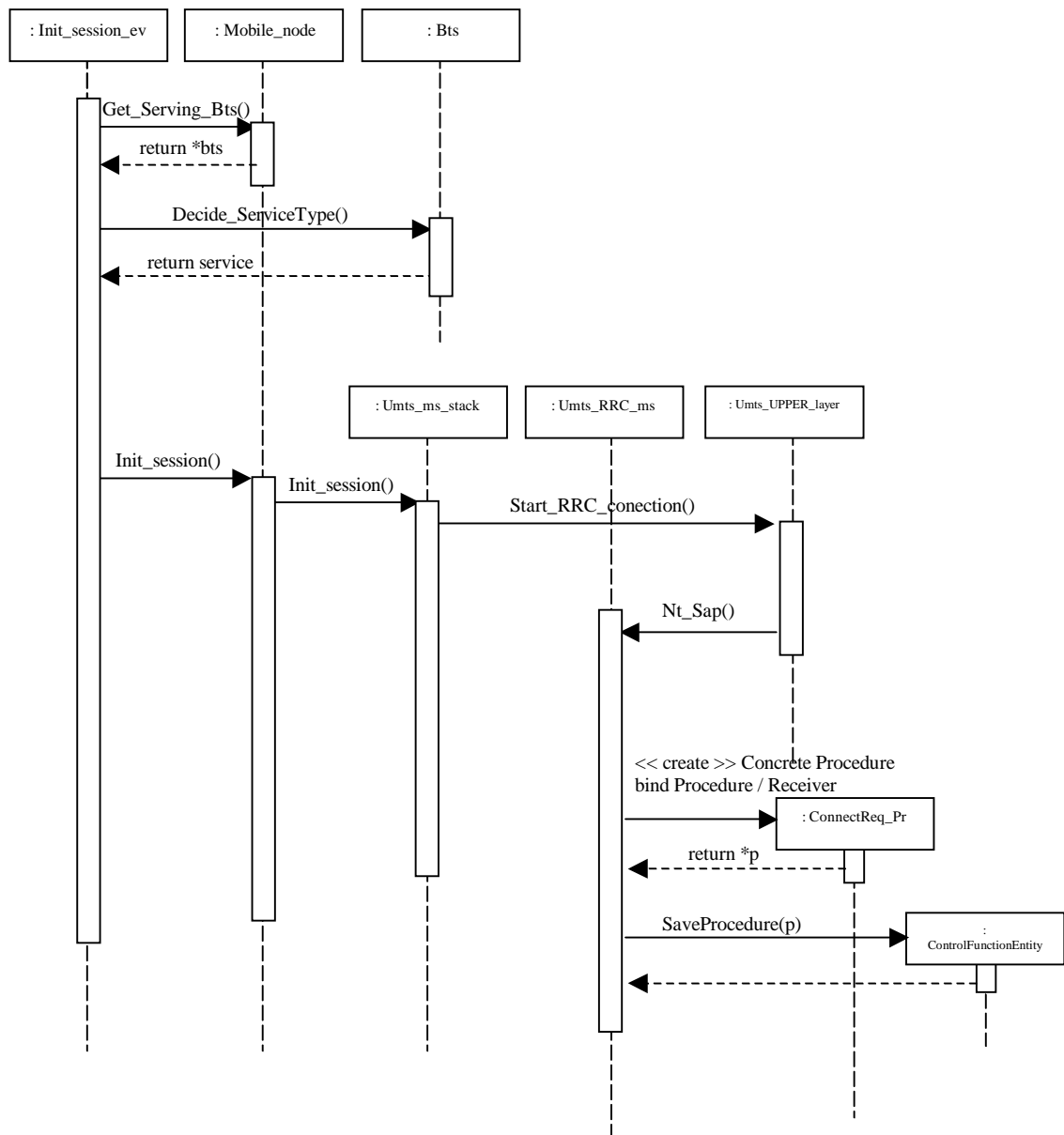


Figura A.20 : Diagramma di sequenza

L'evento di inizio sessione stabilisce per prima cosa il servizio che il mobile ha intenzione di iniziare (Streaming o Interactive) tenendo presente:

- l'informazione sulla *Serving BTS* a cui l'utente è agganciato;
- le percentuali sui tipi di serizi, fissate per ogni bts.

A questo punto l'oggetto *Init_session_ev* avverte il mobile opportuno che è arrivato il momento di iniziare la procedura di connessione RRC alla rete. Pertanto il livello superiore (*Umts_Upper_layer_ms*) notifica all'RRC presente nell'UE di avviare la procedura di *RRC CONNECTION REQUEST*. Il diagramma di sequenza in figura A.20 illustra come interagiscono gli oggetti per creare tale procedura:

- Il *client*, corrispondente all'Application del pattern Command ed identificato dall'entità RRC del mobile, crea l'oggetto *ConnectReq_Pr*, associando a questi un puntatore al Receiver, che in questo caso è il client stesso.
- Poi il *client* deposita nell'entità *ControlFunctionEntity CFE* (corrispondente all'Invoker del pattern Command), la procedura creata.

La CFE è ora in grado di eseguire la procedura che gli è stata consegnata, senza sapere niente sulla procedura stessa. I dettagli su queste operazioni sono illustrati al punto A.2.2.1, dove si parla dello *schedule* dell'entità RRC.

A.2.1.2 *Phy_acc_conf_ev*

Questo oggetto serve a simulare il tempo necessario a realizzare le operazioni di livello fisico per decidere se trasmettere, o no, sui canali RACH e CPCH.

Concentrando l'attenzione sul solo canale comune RACH, si ottengono i diagrammi illustrati nelle figure A.21 e A.22 .

La prima sequenza mostra come si comporta un mobile quando riceve un riscontro positivo (*ACK*) dalla rete. In questo caso l'evento *Phy_acc_conf_ev* comunica al MAC-c/sh nell'UE che l'UTRAN ha risposto con un *ACK* e che ha la possibilità di trasmettere i dati memorizzati nel *RACH_buffer*. Creato quindi un opportuno transport block set, lo passa direttamente alla pari entità nell'UTRAN, usando il metodo *PHY_transfer*. Per finire, il MAC-c/sh nell'UTRAN inserirà il set ricevuto nel *RACH_controller*, che dovrà stabilire secondo curve opportune quali dei TB set hanno subito collisione, o sono arrivati correttamente. Queste curve sono modellate in base ai diagrammi di efficienza tipici dei canali Slotted-Aloha.

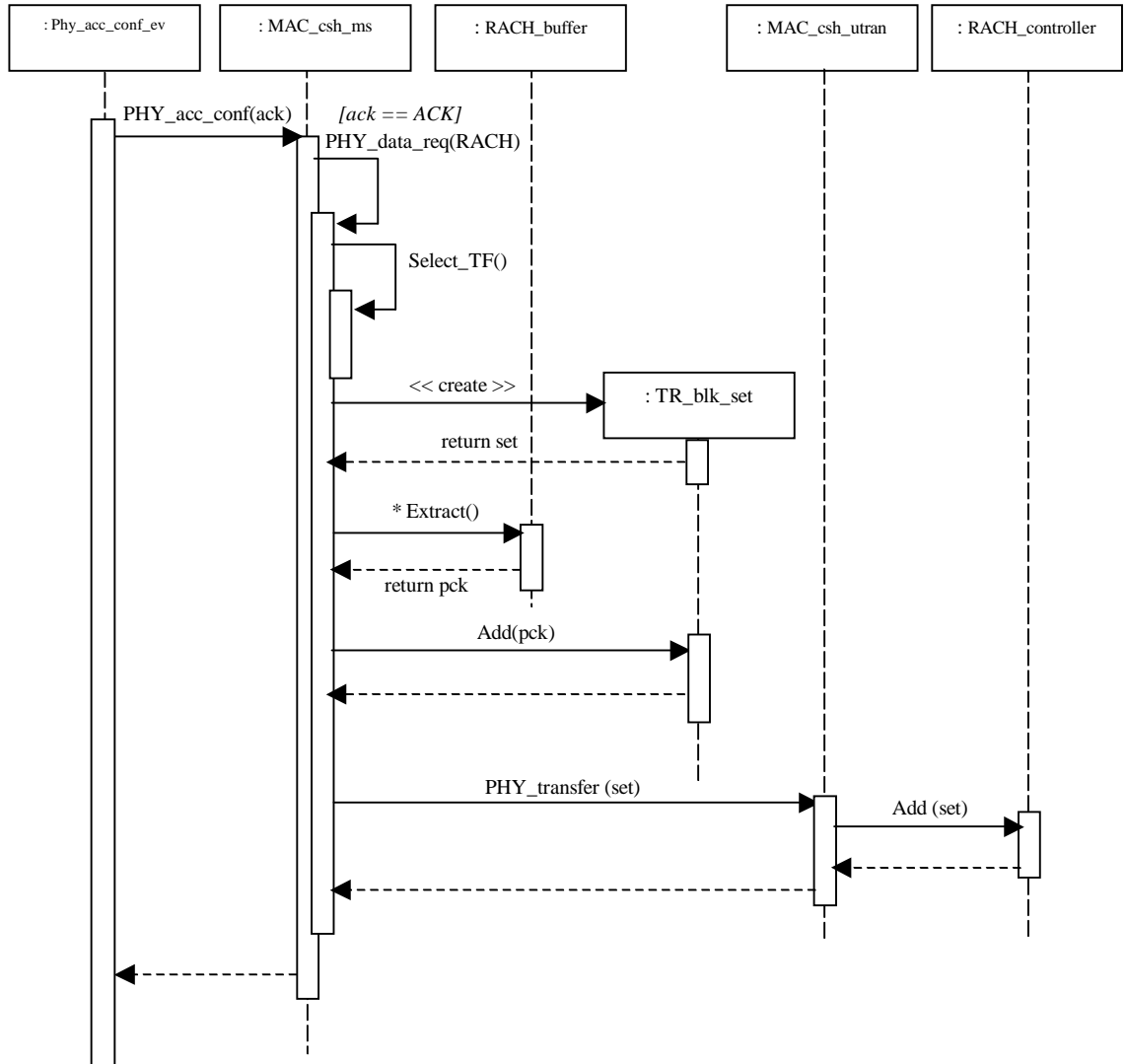


Figura A.21 : Diagramma di sequenza

La figura A.22 mostra invece come si comporta un mobile quando riceve un riscontro negativo (*NACK*) dalla rete o non riceve alcuna risposta (*NO ACK*). In caso di *NACK*, l'entità *MAC-c/sh* nell'UE attiva il timer *T2c* che crea un evento della classe *End_timer_ev*, con l'informazione sull'istante in cui riprovare a trasmettere sul canale *RACH*. In caso di *NO ACK*, il *MAC-c/sh* si comporta come sopra, ma attiva il timer *T2a*. In entrambi i casi si inserisce l'evento creato nello *Scheduler*.

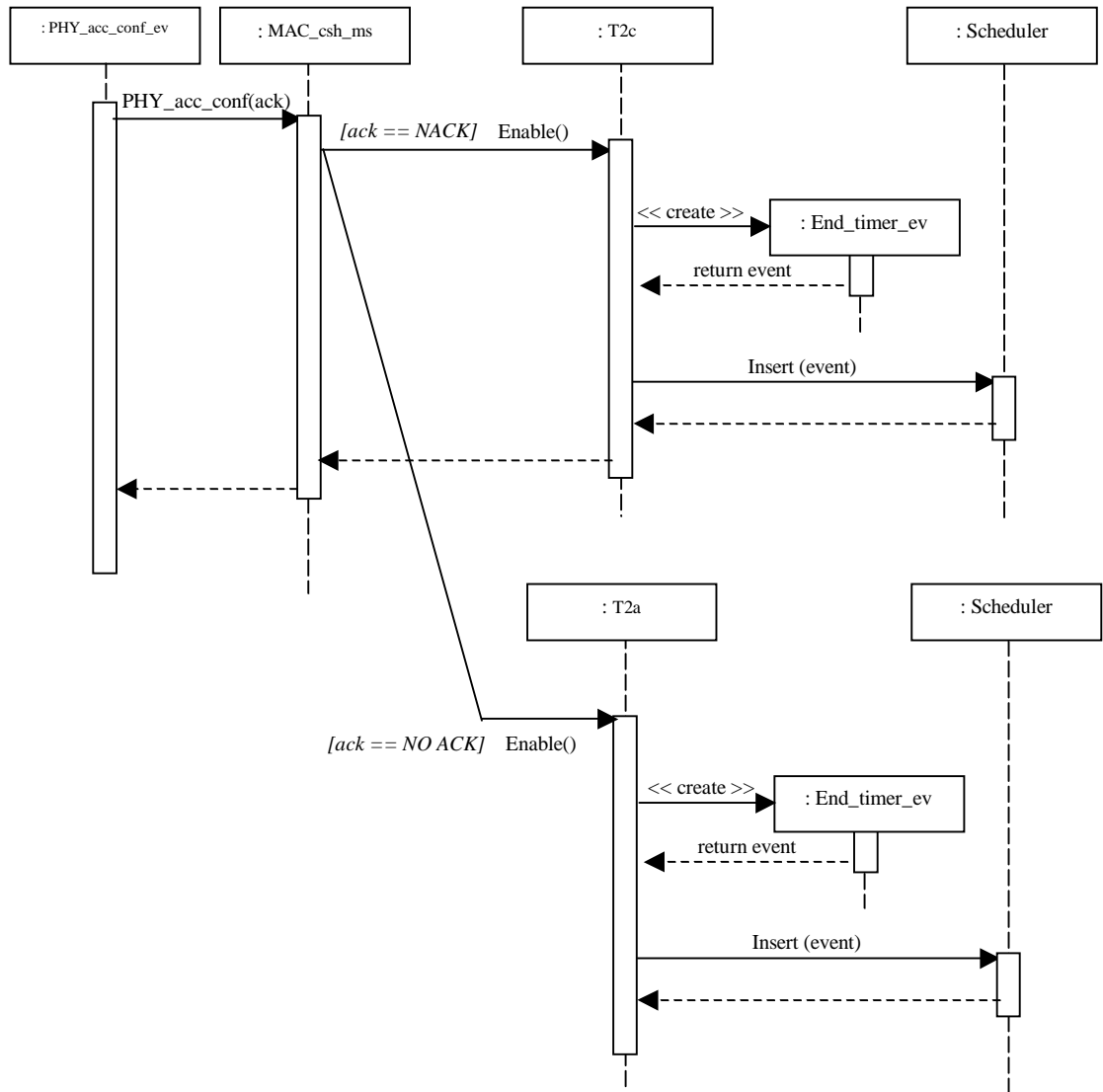


Figura A.22 : Diagramma di sequenza

A.2.2 Gestione del frame corrente

Si vuole mostrare come si comportano i livelli delle pile protocollari presenti nel simulatore in un ben preciso frame temporale.

Il diagramma di macro-livello riportato in figura A.23 deve così essere interpretato. Il programma principale comanda allo scenario simulativo di gestire per primi gli stack dei mobili, che su esso si muovono. Pertanto ordina ad ogni stazione radio base di invocare la procedura di *Handle_frame()* su tutti gli UE sotto il suo controllo. Di conseguenza ciascun

A - Progettazione in UML del simulatore

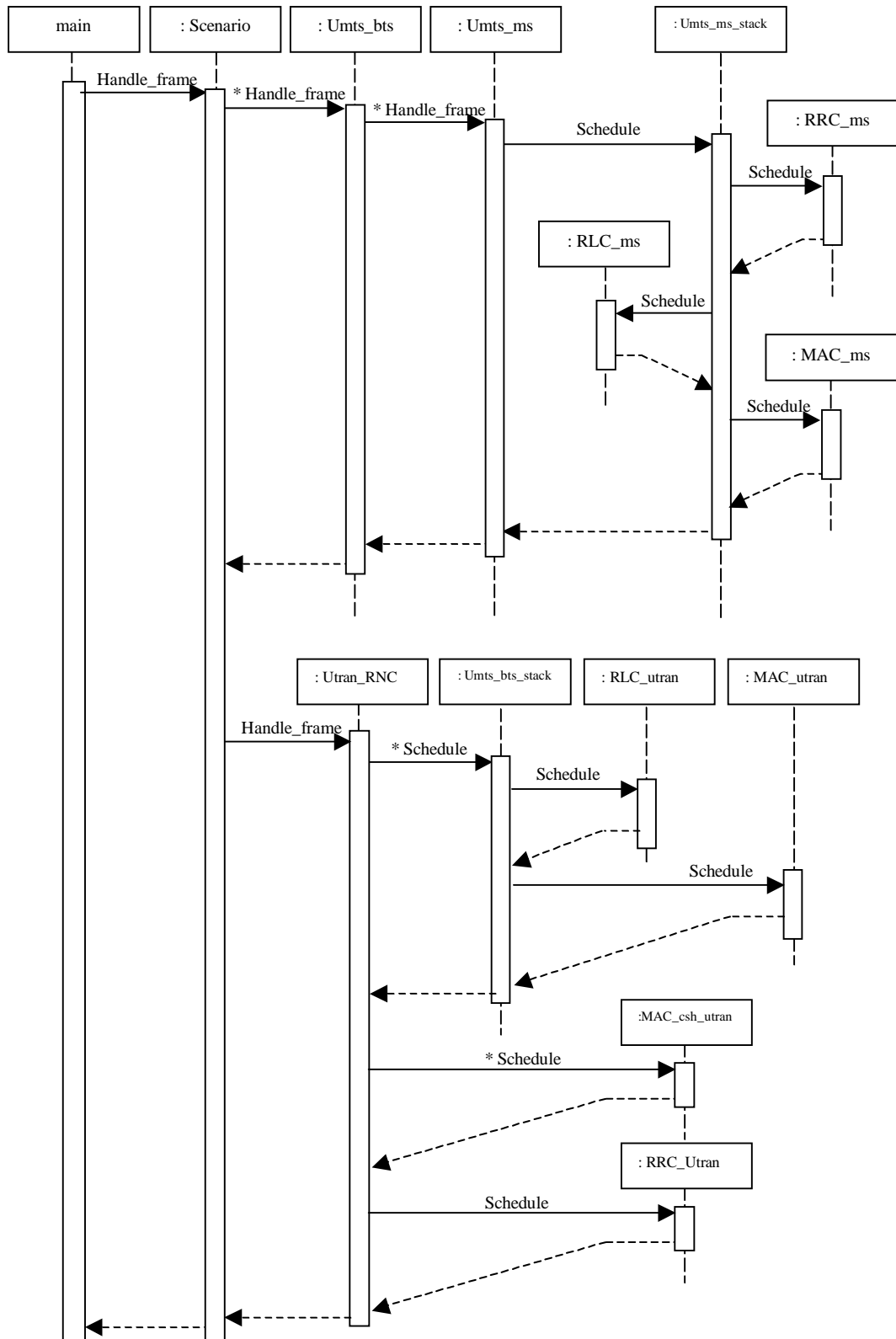


Figura A.23 : diagramma di sequenza di macro-livello

mobile itera tale comando a tutti i livelli della sua pila protocollare. Una volta che il controllo è tornato allo scenario simulativo, questi passa a gestire l'UTRAN RNC.

L'entità RNC del simulatore è composta da tanti stack quanti sono i mobili e da tante entità MAC-c/sh quante sono le celle da esso controllate. Pertanto prima invoca il metodo *Schedule()* su tutti i livelli che compongono le varie Umts_bts_stack e solo dopo chiama lo stesso metodo su tutte le entità MAC-c/sh presenti nelle celle.

Dopo questa descrizione sommaria, è necessario uno sviluppo di micro-livello del diagramma, per comprendere il comportamento dettagliato di ogni livello delle pile protocollari.

A.2.2.1 Livello RRC lato UE

In figura A.24 si rappresenta il funzionamento del livello RRC nel mobile per ogni frame temporale. Questo discorso è valido anche per l'entità RRC lato UTRAN, che non verrà più considerata.

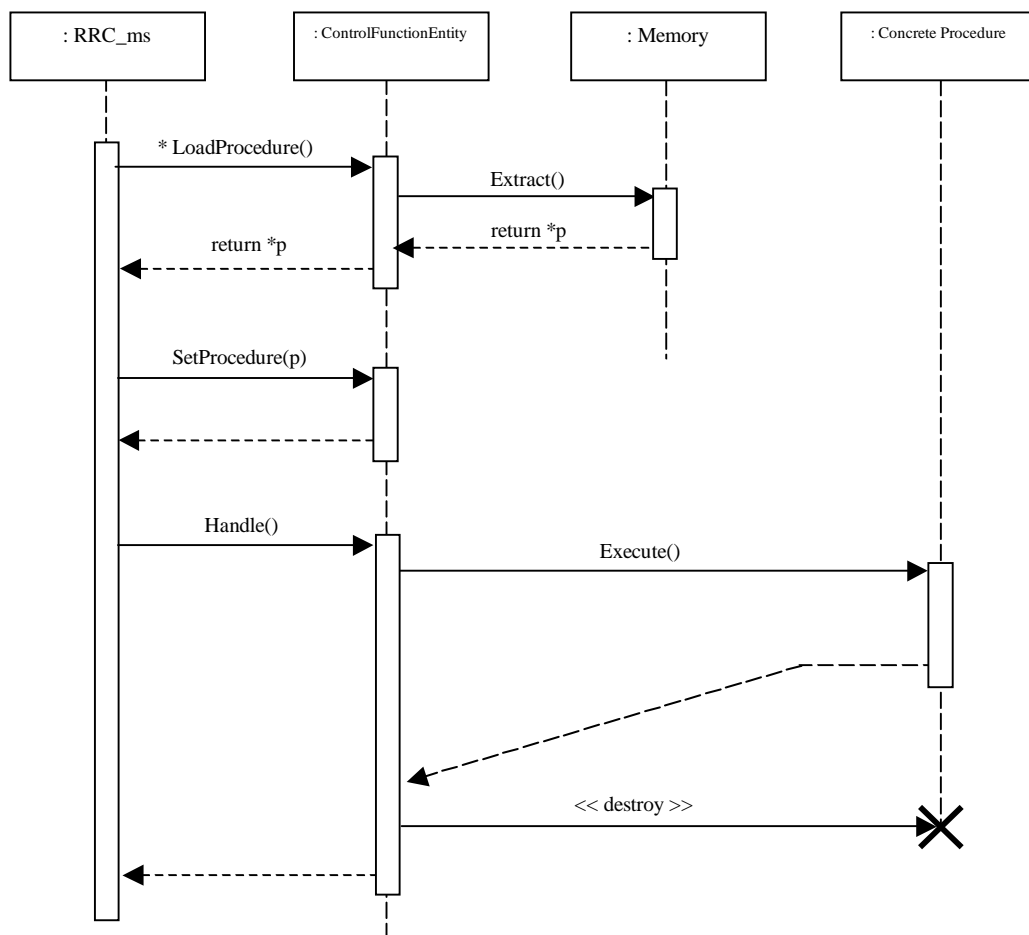


Figura A.24 : Diagramma di sequenza di livello RRC

L'entità RRC estrae dalla memoria propria dell'entità ControlFunctionEntity (CFE) tutte le procedure in essa contenute e ordina alla CFE di gestire ognuna di queste procedure, invocando il metodo `Execute()` sulla procedura concreta.

A.2.2.2 Livello RLC lato UE

Il diagramma A.25 mostra il comportamento del livello RLC nella pila del mobile. Dalla figura si nota come il blocco RLC richiami il metodo *Schedule* per tutte le entità RLC che esso contiene.

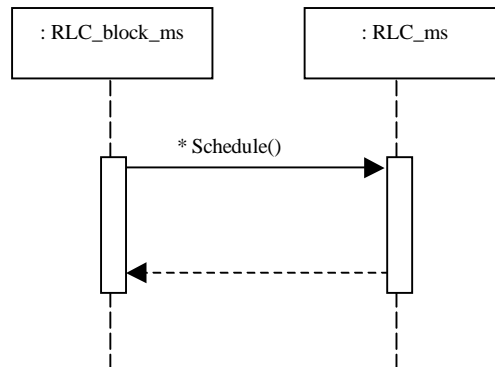


Figura A.25 : Diagramma di sequenza di livello RLC

Poichè esistono tre diverse modalità di funzionamento delle entità RLC, si illustrano in particolare, in figura A.26, le operazioni eseguite dalla sola entità RLC in modalità trasparente.

L'oggetto `RLC_ms` deve gestire sia la trasmissione verso i livelli sottostanti sia verso quelli sovrastanti. Nel primo caso, se verifica la presenza di PDU provenienti dai livelli superiori all'interno del `buffer_tx`, le estrae e le segmenta in RLC PDU di dimensioni opportune, senza aggiungere ulteriori informazioni di controllo. Il livello RLC consegna poi, attraverso il canale logico DTCH, le RLC PDU al livello MAC, sfruttando la primitiva `MAC_data_req`. Nel caso della ricezione, l'entità `RLC_ms` estrae, quando presenti, le PDU giunte al `buffer_rx` dai livelli sottostanti e ne controlla l'intestazione. Solo nel caso in cui siano arrivate senza errori tutte la unità dati della medesima SDU, le riassume e consegna la SDU al livello superiore, mediante la primitiva `RLC_data_ind`.

Il funzionamento del livello RLC nell'UTRAN è identico a quello nell'UE.

A - Progettazione in UML del simulatore

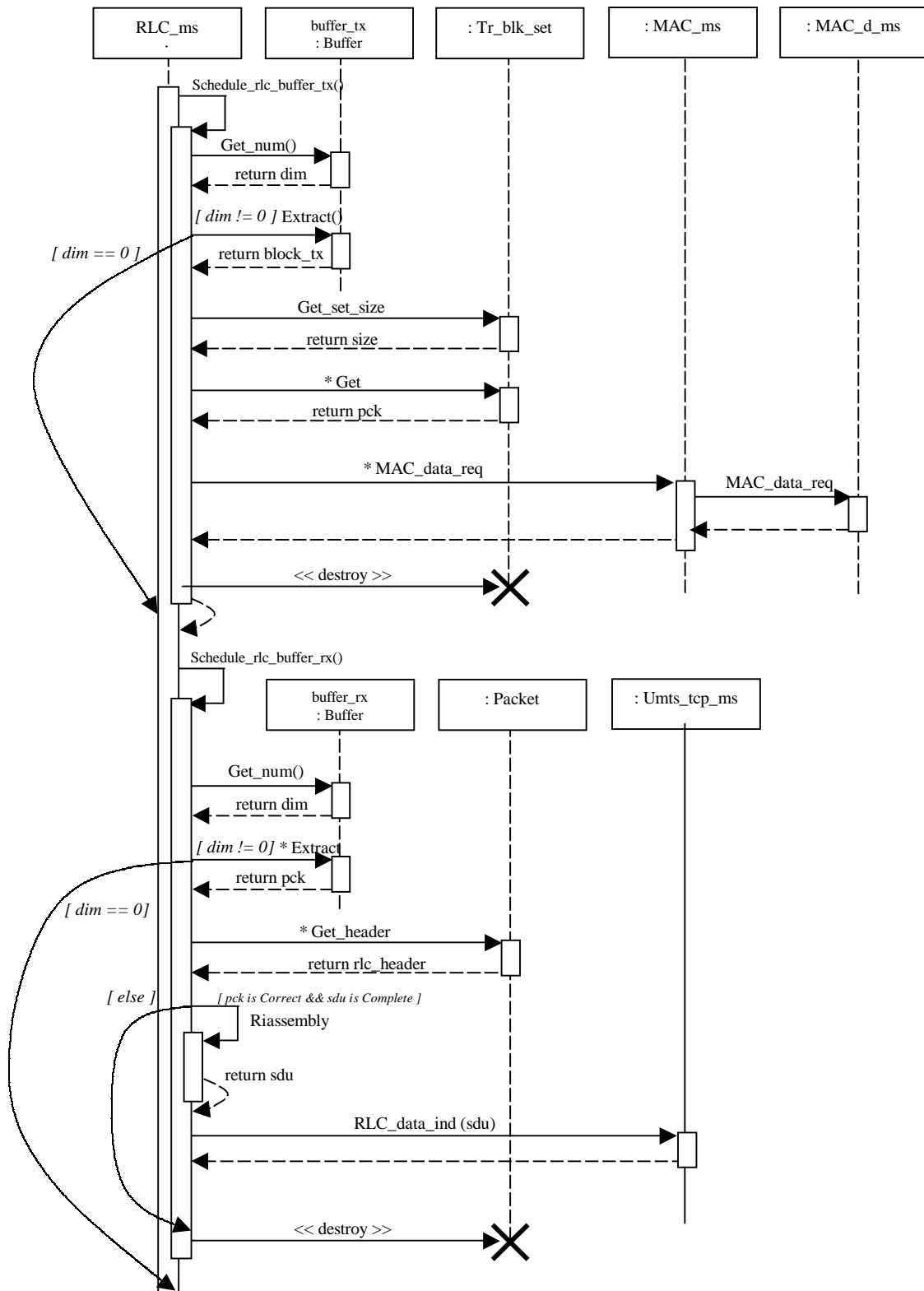


Figura A.26 : Diagramma di sequenza dell'entità RLC in modalità trasparente

A.2.2.3 Livello MAC lato UE

Nelle figure A.27 e A.28 si definisce il comportamento del livello MAC lato UE nel simulatore. La prima figura mostra come il MAC_ms ordini alle due entità che lo compongono, il MAC-d e il MAC-c/sh, di invocare i loro metodi *Schedule*. La seconda illustrazione invece sviluppa in dettaglio quello che accade nel MAC-d.

Il MAC-d lato UE, dopo avere scelto un transport format adeguato, estrae dal *flow_control_buffer* un numero opportuno di pacchetti e li invia, tramite la primitiva *MAC_data_req*, all'entità MAC-c/sh con cui è collegato. Quest'ultima ha il compito di creare l'intestazione di livello MAC e di inserire i pacchetti nel buffer opportuno.

Una volta che il controllo è tornato al MAC-d, questi deve gestire eventuali trasmissioni sul canale DCH, se vi sono elementi nel *DCH_buffer* (come ipotizzato nel diagramma in figura A.28).

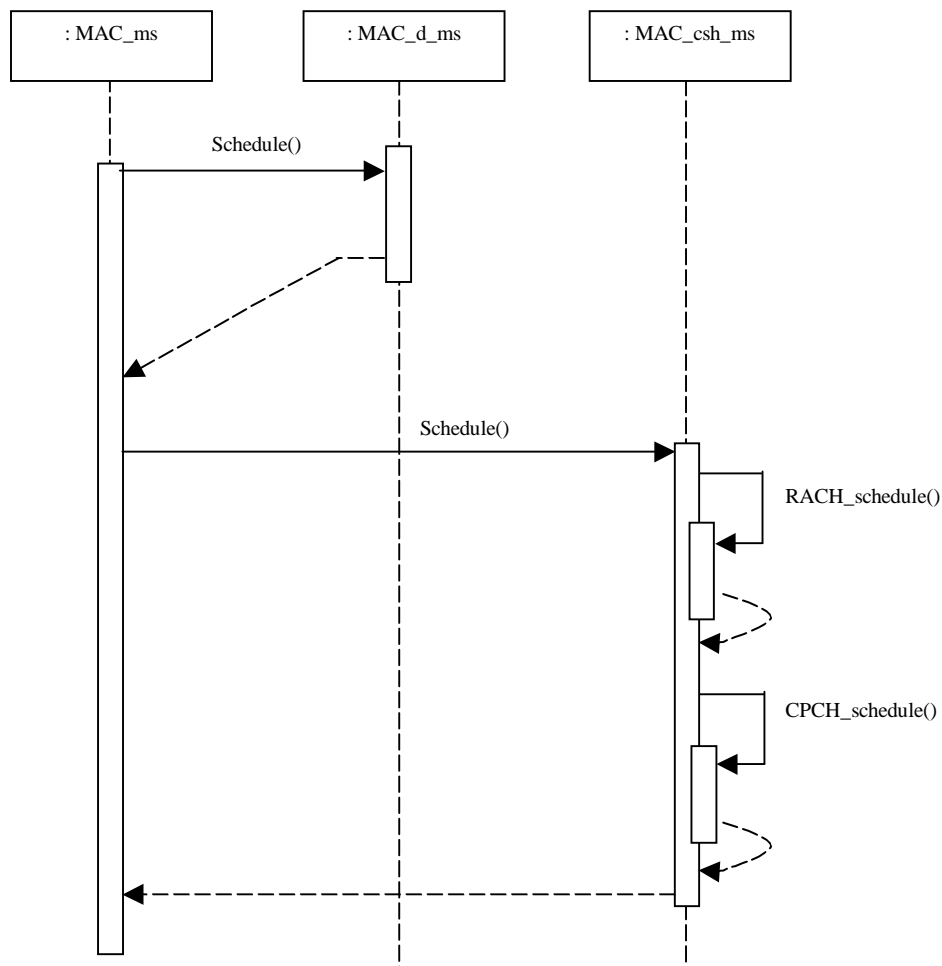


Figura A.27 : Diagramma di sequenza del livello MAC lato UE

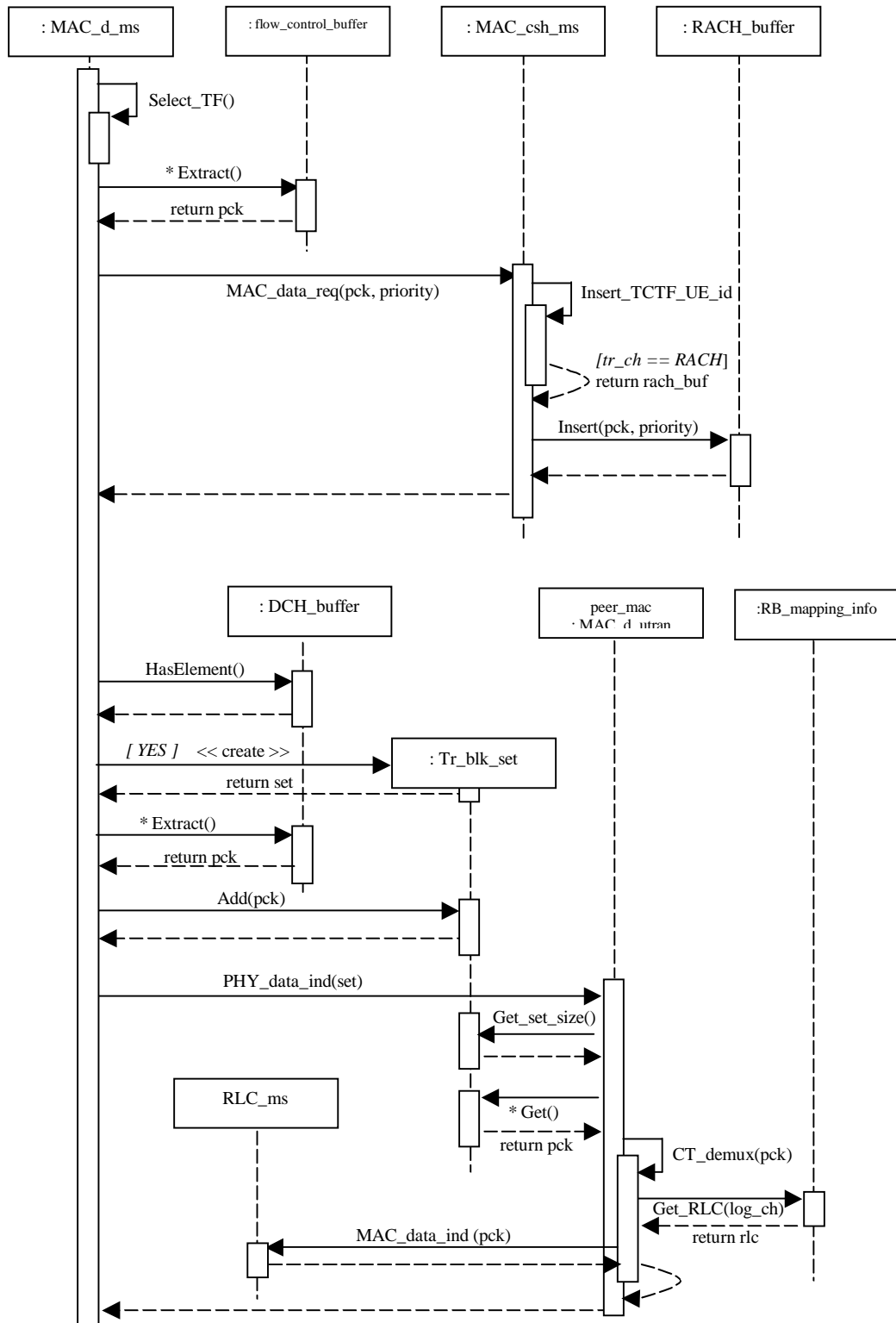


Figura A.28 : Diagramma di sequenza dell'entità MAC-d

A.2.2.4 Livello MAC lato UTRAN

La gestione del livello MAC nell'RNC dell'UTRAN è diversa da quanto visto nei diagrammi di sequenza relativi al MAC lato UE. Questo è dovuto al diverso rapporto di comunicazione tra le varie entità che compongono il MAC nel mobile e nell'UTRAN.

L'oggetto MAC_utran di ogni pila presente nell'RNC gestisce la sola entità MAC-d, mentre l'entità MAC-c/sh, condivisa in genere da più MAC_d dell'UTRAN, è gestita direttamente dall'RNC.

Le figure A.29, A.30 e A.31 mostrano i diagrammi di micro-livello di queste entità.

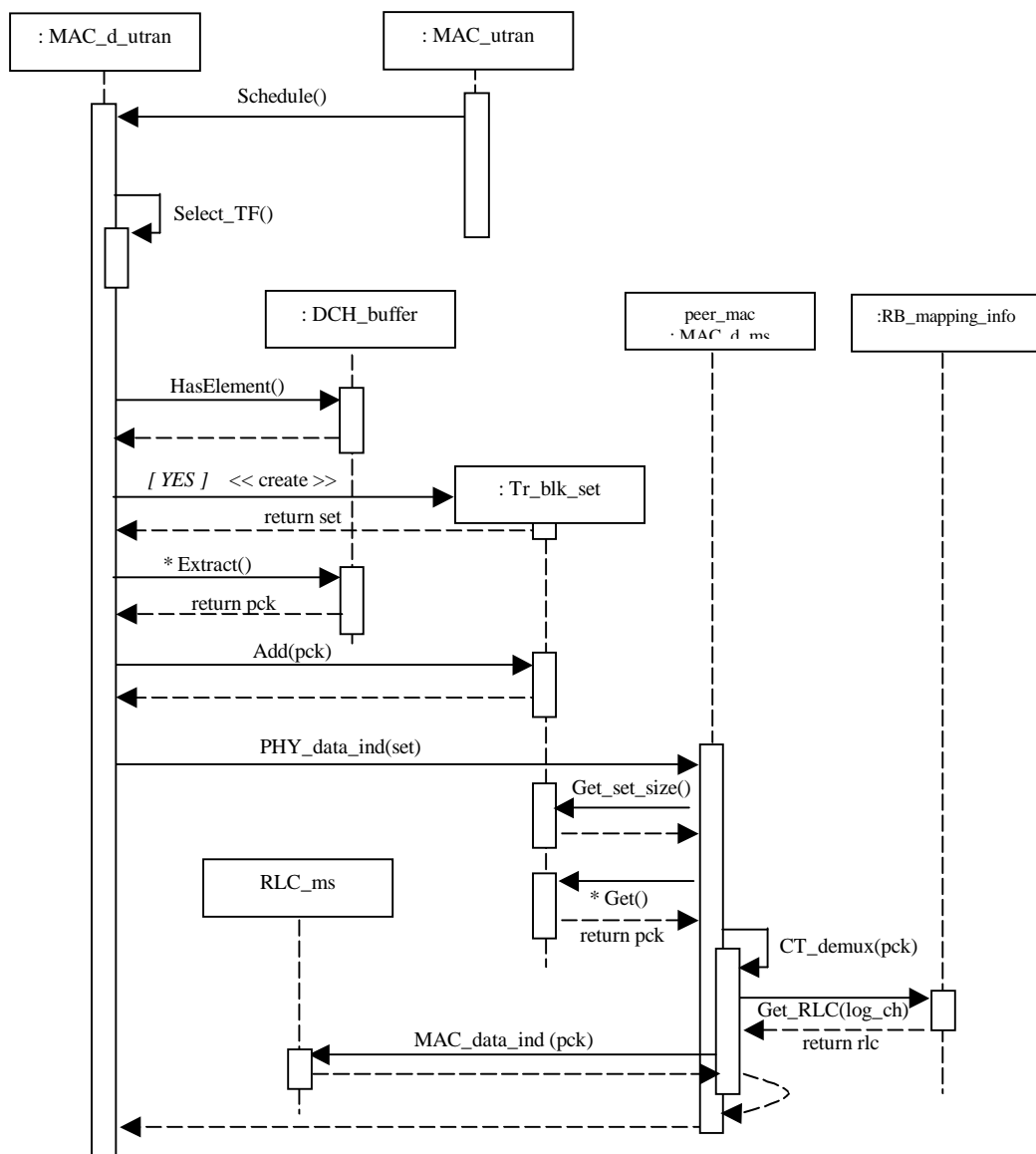


Figura A.29 : Diagramma di sequenza del MAC-d lato UTRAN

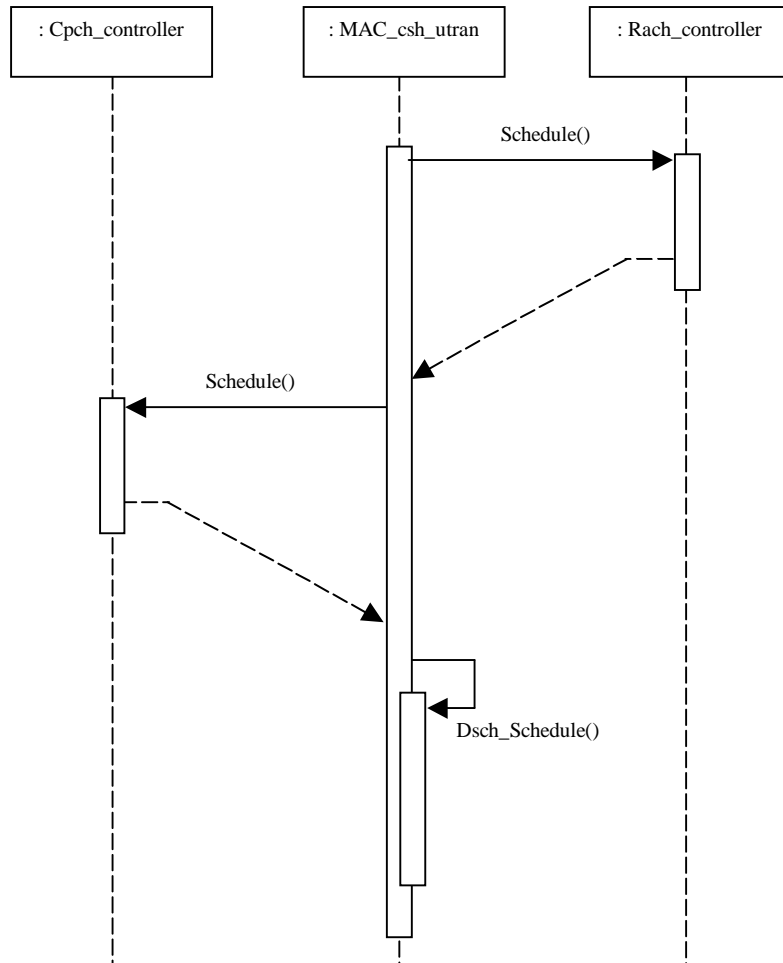


Figura A.30 : Diagramma di sequenza del MAC-c/sh lato UTRAN

In figura A.31 si illustra in dettaglio il ruolo svolto, all'interno del MAC-c/sh di ogni cella nell'UTRAN, dall'oggetto RACH-controller. Il compito del CPCH_controller non è stato rappresentato, ma è in linea di principio, anche se più complesso, analogo a quello del controllore del RACH.

A - Progettazione in UML del simulatore

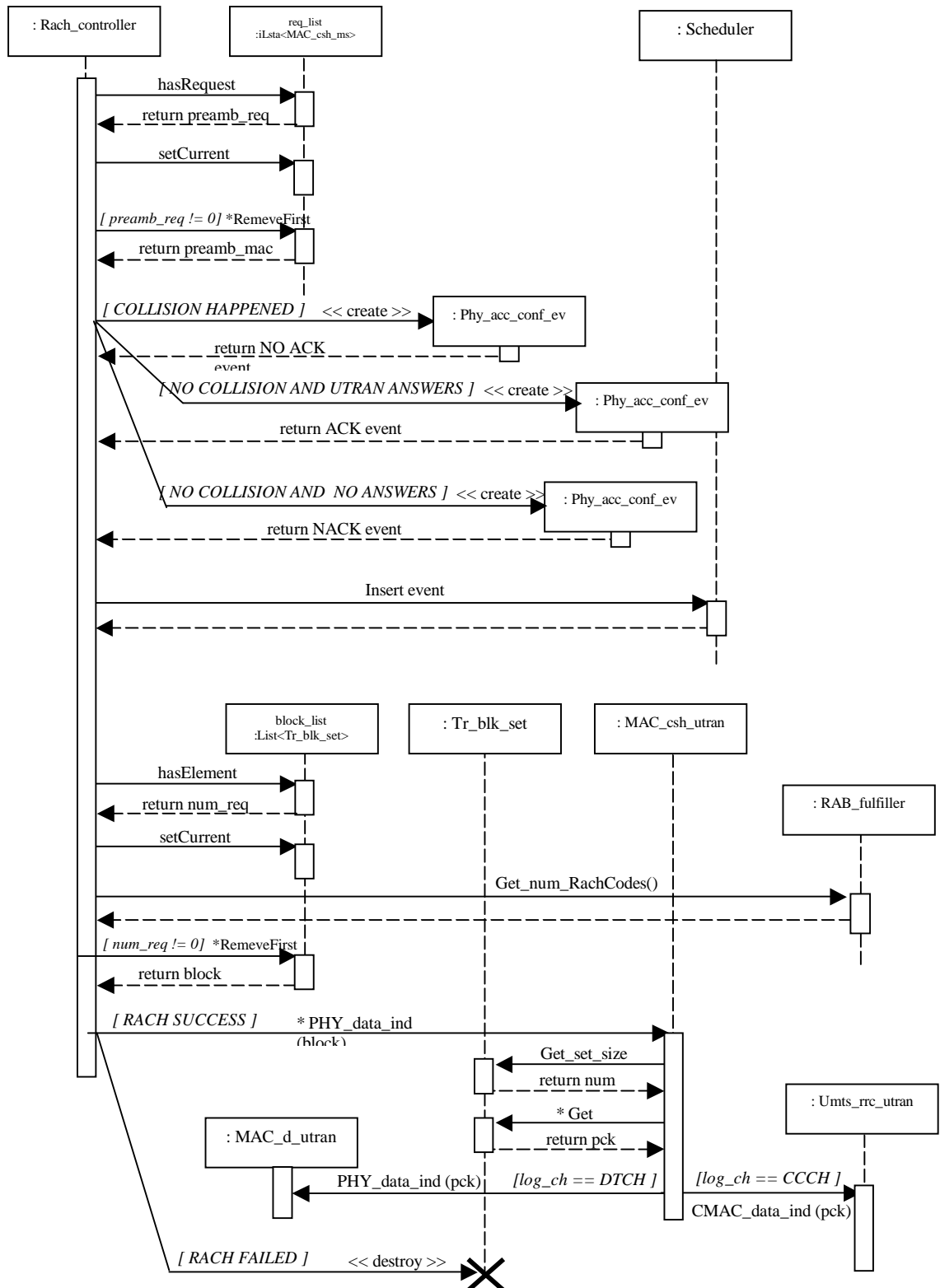


Figura A.31 : Diagramma di sequenza di micro-livello del RACH_controller