Specifica in TRIO delle Unità di Elaborazione Periferica: aspetti relativi alla struttura delle versioni, alla gestione della base dei tempi e al conteggio dei quanti di energia

Dino Mandrioli, Angelo Morzenti,
Politecnico di Milano, Dipartimento di Elettronica,
Novembre 1997.

1. Premesse metodologiche e inquadramento del documento

1.1. Scopo e organizzazione del documento

Questo documento contiene una parziale descrizione e formalizzazione mediante il linguaggio di specifica TRIO delle specifiche di tre diverse unità di elaborazione periferica appartenenti al sistema di telegestione illustrato in [DH020].

Il suo scopo principale è di illustrare alcuni benefici derivanti dall'impiego del metodo di specifica basato su TRIO rispetto ai tradizionali metodi informali. In generale, l'uso di metodi formali di specifica si è dimostrato efficace per raggiungere i seguenti obiettivi:

- Specifiche rigorose e precise;
- Maggior facilità e migliori garanzie nella verifica di correttezza e nel passaggio dalla specifica al progetto e alla realizzazione;
- Possibilità di sfruttare strumenti automatici di supporto allo sviluppo (editor, generatori di codice, generatori di casi di test, ...[1]).

A questi benefici tradizionali TRIO aggiunge una particolare attenzione alla *leggibilità delle specifiche* anche da parte di personale non particolarmente addestrato all'uso dei formalismi e un approccio incrementale alla formalizzazione basato sul principio di *arricchire ma non stravolgere* i metodi di specifica informali.

Per un'analisi generale sulla formalizzazione delle specifiche, sui benefici e costi da essa comportati, sulle caratteristiche peculiari di TRIO rispetto ad altri approcci si rimanda a [CC&94, MPS93, M&S94]. In questo documento l'attenzione viene invece rivolta a un aspetto particolare dell'attività di specifica e dei relativi obiettivi di qualità, soprattutto rilevante nelle applicazioni prese in esame: la *gestione delle specifiche*, ossia la necessità di ottenere con poco sforzo diverse versioni di una famiglia di prodotti (chiameremo questo tipo di gestione "*gestione spaziale* delle versioni") e la necessità di controllarne l'evoluzione nel tempo mettendo in chiaro quali aspetti sono rimasti immutati e quali sono variati nel passaggio da una versione a quella successiva (chiameremo questo tipo di gestione "*gestione temporale* delle versioni").

La seconda parte di questo documento prende in considerazione le unità di elaborazione (UE) UEP, UEPB, UEPM descritte rispettivamente nei documenti [DH023, DH025, DH026] e ne propone una nuova –parziale[2]– formulazione con l'obiettivo di dimostrare come l'impiego di TRIO possa fornire un valido aiuto alla gestione delle specifiche producendo notevoli miglioramenti rispetto alla loro semplice stesura in documenti informali.

TRIO permette di raggiungere lo scopo suddetto grazie alle sue caratteristiche *object-oriented* (OO): infatti alcuni meccanismi tipici dei linguaggi OO (a esempio, ereditarietà e genericità) sono altrettanto utili alla *riusabilità delle specifiche* quanto lo sono per la riusabilità del software, inteso come codice eseguibile. In particolare si mostrerà come ad ogni elemento del documento informale (capitolo, sezione, paragrafo) sia possibile far corrispondere un preciso costrutto linguistico di TRIO (classe); inoltre la descrizione sfrutterà il meccanismo dell'ereditarietà specificando in primo luogo

[1] Precisamente, al momento attuale l'ambiente TRIO include editor di parte testuale e grafica delle specifiche, strumenti di convalida e verifica quali analizzatori di simulazioni, generatori di simulazioni, dimostratori di proprietà, generatori di casi di test.

[2] Le parti dei documenti originari non riprese nel documento presente verranno esplicitamente marcate dall'etichetta "omissis".

le caratteristiche comuni a tutte le UE e successivamente gli aspetti peculiari delle singole UE (UEP, ...) come *eredi* della classe base (vedi figura 1.1).

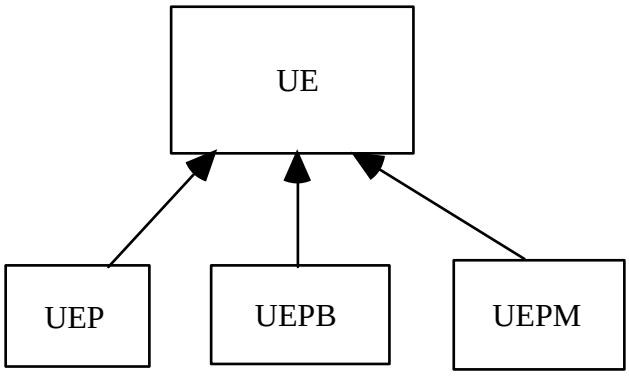


Figura 1.1 Le relazioni di ereditarietà tra le varie classi che specificano le UE.

Ogni classe verrà poi descritta come classe strutturata che include diverse classi componenti secondo lo schema seguente:

1. UE verrà descritta come l'aggregazione di diverse classi componenti (vedi figura 1.2).
2. Ogni erede di UE verrà descritta specificando eventuali ulteriori componenti non comuni alle altre unità ed eventualmente ridefinendo componenti che hanno caratteristiche diverse a seconda dell'unità cui appartengono.
3. Componenti che risultino identiche in ogni UE verranno identificate mediante un unico identificatore; invece componenti con caratteristiche dipendenti dalla particolare unità verranno identificati attraverso un suffisso apposto mediante *underscore* '_' all'identificatore del componente base, come illustrato in figura 1.3 nel caso della UEP (le altre eredi di UE verranno costruite in maniera simile).

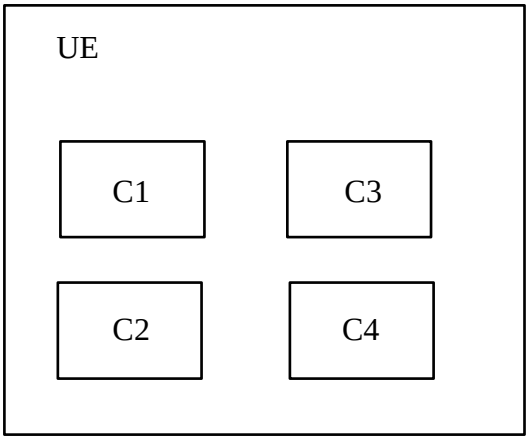


Figura 1.2 La generica struttura di una classe genitore.

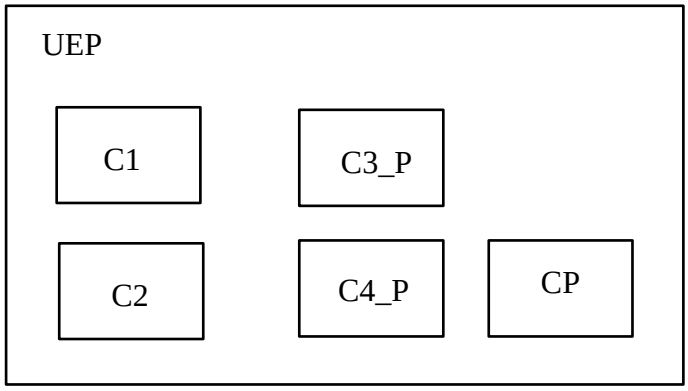


Figura 1.3. La struttura di un erede di UE.

Legenda C1 e C2 denotano componenti comuni a tutte le UE e identici per ognuna di esse; C3 e C4 denotano componenti esistenti in ogni UE ma con caratteristiche diverse nelle singole eredi. C3_P indica la versione di C3 relativa a UEP. Si noti che *C3_P è a sua volta erede di C3*. CP indica un componente di UEP che non esiste nelle altre UE.

Lo schema di cui sopra può essere applicato ricorsivamente: così ad esempio C3 potrà essere ulteriormente decomposta in classi più semplici e le sue eredi C3_P, ... potranno essere definite specificando quali componenti di C3 restano inalterate, quali vanno modificate e quali vanno aggiunte.

Ne deriverà perciò una descrizione estremamente sistematica e rigorosa delle diverse unità che metterà in evidenza, ad ogni livello di astrazione, parti comuni e parti specifiche. Qualora lo si desiderasse, sarà anche possibile, eventualmente in modo automatico, ricavare la specifica di una singola unità (UEP, UEPB, UEPM) a partire dal documento globale.

Infine, la terza parte del documento contiene alcuni commenti esplicativi sulle scelte adottate nel processo di formalizzazione e indica alcuni suggerimenti e alternative di carattere metodologico.

In omaggio al principio che la formalizzazione deve costituire un arricchimento ma non deve sostituire una descrizione informale per non pregiudicarne la leggibilità, la specifica delle UE è fornita mediante la cosiddetta tecnica "pagina-pari/pagina-dispari": nelle pagine pari sono riportate le formule TRIO che specificano determinati requisiti; ad esse affacciati, nelle pagine dispari, sono riportati i commenti informali corrispondenti, in modo che la parte informale del documento sia completamente autocontenuta e leggibile senza alcun prerequisito di conoscenza della notazione formale; la comprensione della parte formale richiede invece una limitata conoscenza di TRIO (si veda ad esempio [M&S94])[3].

In base allo stesso principio si è cercato di restare il più possibile aderenti alla struttura e ai contenuti dei documenti originari. Alcune modifiche si sono tuttavia rese necessarie per correggere lievi incongruenze o incompletezze, come spesso accade nel processo di formalizzazione di specifiche informali. Alcune di queste modifiche sono oggetto di commento nella parte terza.

Si sottolinea infine che, essendo stato il lavoro di formalizzazione delle specifiche solo parziale, è verosimile che proseguendo in tale lavoro possano evidenziarsi altre incompletezze che potrebbero richiedere ulteriori modifiche, anche alle parti già formalizzate. L'esperienza insegna infatti che il processo di formalizzazione non è quasi mai lineare ma richiede diverse “passate” attraverso le quali alcuni dettagli vengono chiariti e, se necessario, modificati.

[3]Anche in questo caso, tuttavia, l'uso attento della logica matematica che viene esercitato in TRIO dovrebbe permettere un certo livello di comprensibilità delle formule di specifica con sforzi di apprendimento particolarmente limitati.

2. Descrizione e formalizzazione delle unità di elaborazione

2.1. Scopo delle prescrizioni

Le presenti prescrizioni hanno lo scopo di descrivere le caratteristiche funzionali e costruttive delle unità elettroniche (UE) da associare a diversi gruppi integrati di misura.

In particolare:

1. La UEP è associata ai gruppi GMY e GTY
2. La UEPB è associata ai gruppi GTWD e GTWS
3. La UEPM è associata ai gruppi GTWM

2.2. Campo di applicazione

Le presenti prescrizioni si applicano alle UE da associare a diversi gruppi integrati. In particolare:

1. La UEP è associata
   - ai gruppi di misura integrati GMY, applicati agli utenti BT monofase con potenza contrattuale compresa tra 1.5 e 6 kW;
   - ai gruppi di misura integrati GTY, applicati agli utenti BT trifase con potenza contrattuale compresa tra 3 e 15 kW.
2. La UEPB è associata
   - ai gruppi di misura integrati GTWD, applicati agli utenti BT trifase con potenza contrattuale compresa tra 20 e 30 kW;
   - ai gruppi di misura integrati GTWS, applicati agli utenti BT trifase con potenza contrattuale compresa tra 35 e 200 kW.
3. La UEPM è associata
   - ai gruppi di misura integrati GTWM1, applicati agli utenti MT trifase con potenza contrattuale compresa tra 100 e 500 kW;
   - ai gruppi di misura integrati GTWM2, applicati agli utenti MT trifase con potenza contrattuale compresa tra 500 e 2.500 kW;
   - ai gruppi di misura integrati GTWM3, applicati agli utenti MT trifase con potenza contrattuale compresa tra 2.500 e 7.500 kW.

2.3. Prescrizioni e norme richiamate nel testo

Come descritto nella sezione 2, le UE fanno parte dell'intero sistema di telegestione del settore della distribuzione. La descrizione contenuta in questo documento fa perciò riferimento ai seguenti altri documenti realtivi all'intero sistema di telegestione e a sue parti:

Documenti di riferimento comuni ad ogni UE:

- TAB. ENEL DH020

  Sistema di telegestione - Architettura, componenti, Funzioni

(omissis)

Documenti di riferimento per la UEP:

(omissis)

Documenti di riferimento per la UEPB:

(omissis)
Documenti di riferimento per la UEPM:
(omissis)

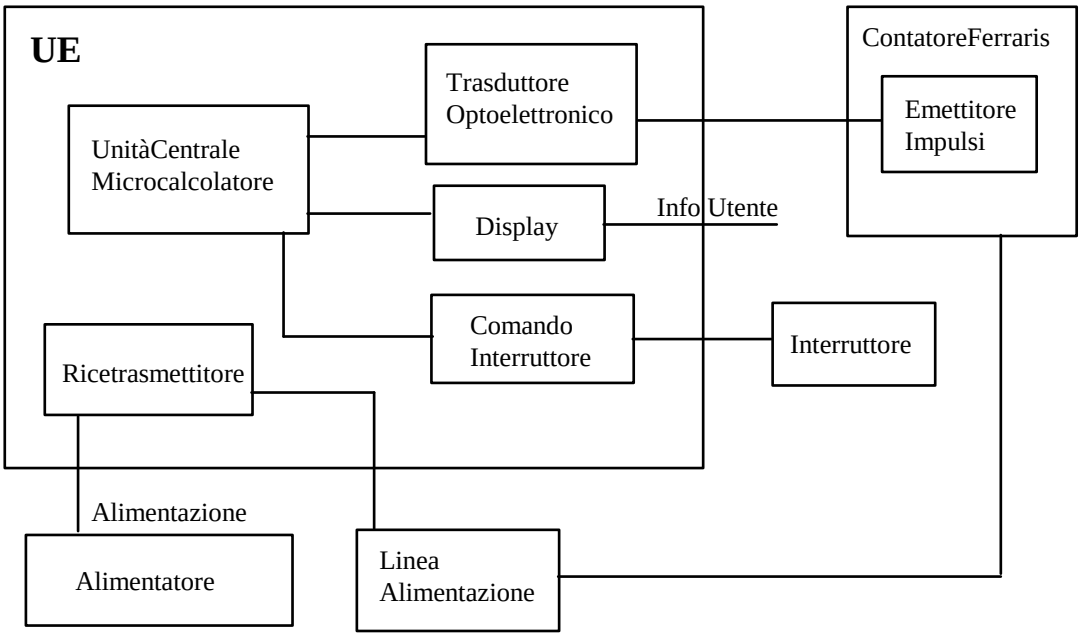


Figura 2.4.1 Struttura fisica della classe UE e sue connessioni fondamentali.

2.4. Architettura fisica delle UE

Ogni UE è costituita da un'Unità Centrale a microcalcolatore cui fanno capo:

- un trasduttore optoelettronico, collegato al dispositivo emettitore di impulsi realizzato all'interno del contatore Ferraris;
- un dispositivo di comando, collegato all'interruttore magnetotermico;
- un dispositivo ricetrasmettitore, collegato, a monte del contatore Ferraris e dell'interruttore, alla stessa linea elettrica di alimentazione dell'utente;
- un display.

Completa l'architettura un alimentatore, anch'esso collegato, a monte del contatore Ferraris e dell'interruttore, alla stessa linea elettrica di alimentazione dell'utente.

Quanto sopra è formalizzato dall'architettura di classi TRIO fornita dalla figura 2.4.1

Specifichiamo ora l'architettura delle singole UE come eredi della UE genitore.

2.4.1 Architettura fisica della UEP

La UEP eredita da UE identica struttura (non ne riportiamo una figura in quanto ha esattamente gli stessi componenti). Tuttavia ne modifica alcuni componenti.

Precisamente, rimangono inalterati:

L'unità centrale a microcalcolatore

Il trasduttore optoelettronico

Il ricetrasmettitore

Lo strumento G. Ferraris

Sono invece ridefiniti:

Il Display, ridefinito come Display_P

Il ComandoInterruttore, ridefinito come ComandoInterruttore_P

L'Alimentatore, ridefinito come Alimentatore_P

L'Interruttore, ridefinito come Interruttore_P

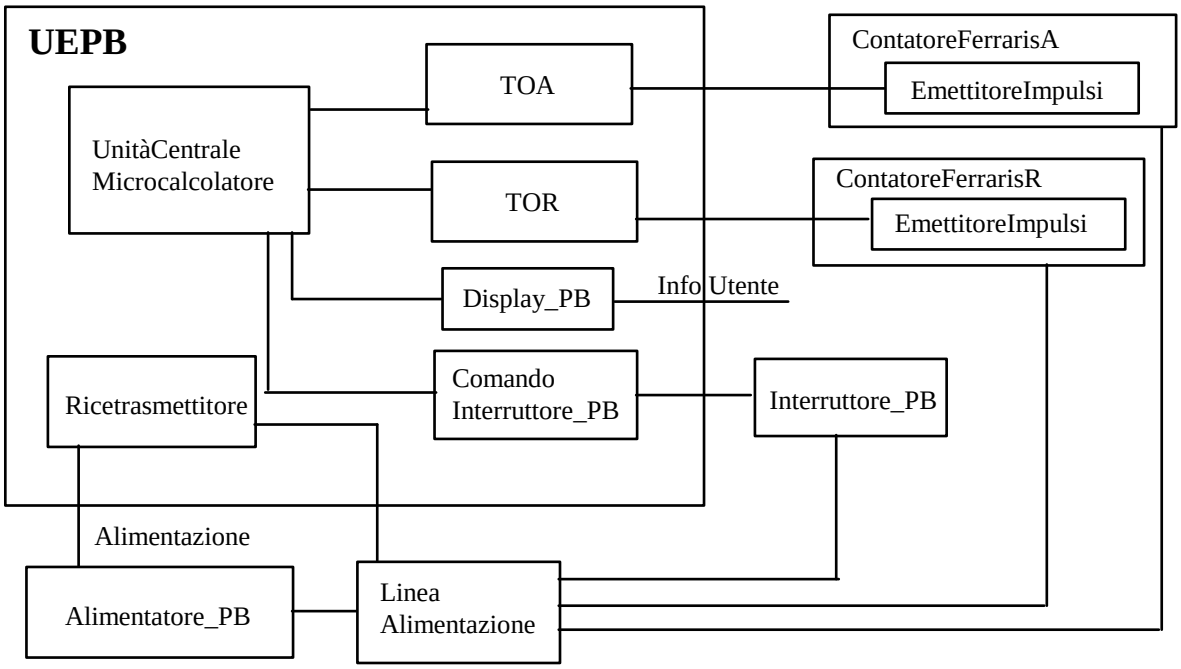


Figura 2.4.2 La struttura della UEPB.

2.4.2 Architettura fisica della UEPB

La UEPB eredita tutti i componenti di UE. Ad essi aggiunge un ulteriore modulo TrasduttoreOptoelettronicoReattivo (TOR) della stessa classe del TrasduttoreOptoelettronico della UE. Esso è usato nella misura di energia reattiva ed è collegato al contatore Ferrsris di energia reattiva. La UEPB ha perciò la struttura di figura 2.4.2.

TOA e TOR sono moduli della classe TrasduttoreOptoelettronico, identici ma con funzioni e collegamenti diversi.

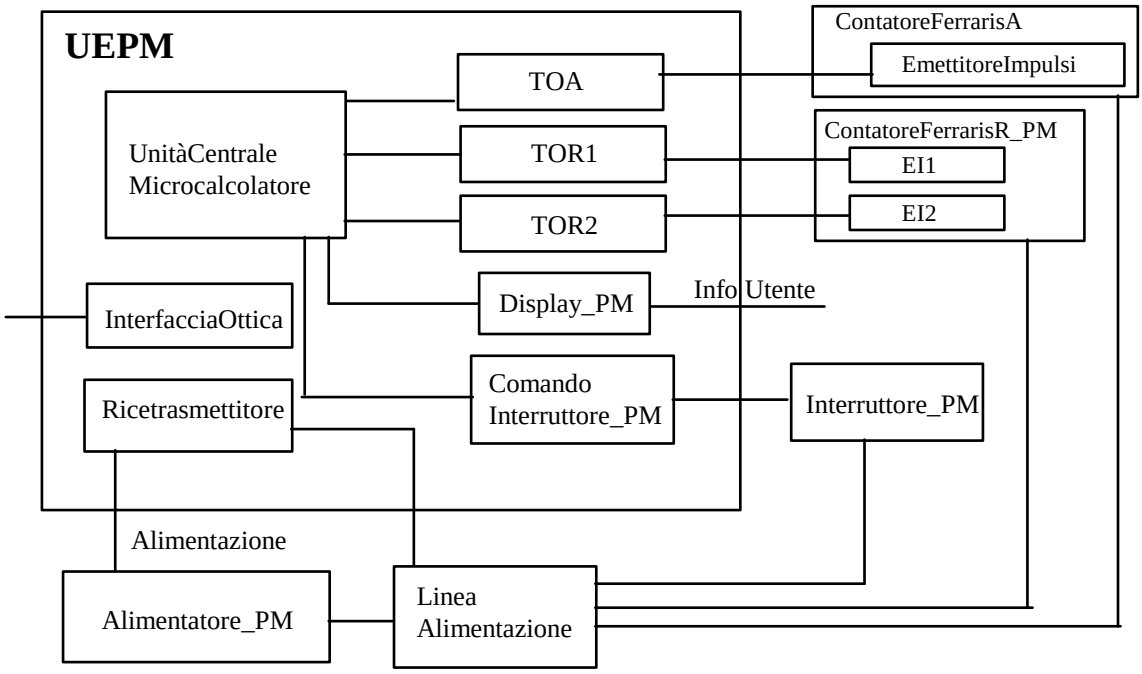
La UEPB ridefinisce i seguenti componenti di UE:

Il Display, ridefinito come Display_PB

Il ComandoInterruttore, ridefinito come ComandoInterruttore_PB

L'Alimentatore, ridefinito come Alimentatore_PB

L'Interruttore, ridefinito come Interruttore_PB



Legenda: EI2: Emettitore di impulsi 1 del Ferraris reattivo

EI2: Emettitore di impulsi 2 del Ferraris reattivo

Figura 2.4.3 La struttura della UEPM.

2.4.3 Architettura fisica della UEPM

La UEPM eredita tutti i componenti di UE. A essi aggiunge due ulteriori moduli, TrasduttoreOptoelettronicoReattivo (TOR1 e TOR2) della stessa classe del TrasduttoreOptoelettronico della UE. Essi sono usati nella misura di energia reattiva e sono collegati al contatore Ferrsris di energia reattiva. La UEPM aggiunge inoltre un'interfaccia ottica per il collegamento in alternativa:

- del terminale utente TU-MT
- del terminale portatile.

La UEPM ha perciò la struttura di figura 2.4.3.

La UEPM ridefinisce i seguenti componenti di UE:

Il Display, ridefinito come Display_PM

Il ComandoInterruttore, ridefinito come ComandoInterruttore_PM

L'Alimentatore, ridefinito come Alimentatore_PM

L'Interruttore, ridefinito come Interruttore_PM

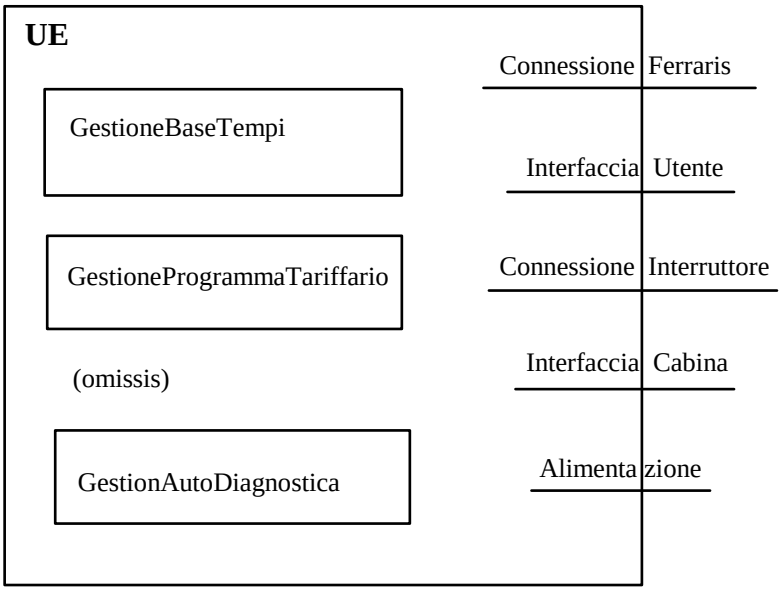


Figura 2.5.1 Struttura della classe "UE".

2.5. Architettura funzionale e compiti delle UE[4]

Nell'ambito dell'architettura fisica descritta nella sezione 2.4, l'UE deve svolgere i seguenti compiti fondamentali.

A. Gestione di una base tempi, con funzioni di orologio/calendario.

B. Gestione di un programma tariffario, atto a permettere l'applicazione di tariffe differenziate con possibilità di modulazione giornaliera e stagionale della potenza impegnata e di tariffe a punta mobile, entrambe con ampio grado di flessibilità sugli orari.

C. Conteggio dei "quanti" di energia misurati dal contatore Ferraris.

D. Elaborazione dei "quanti" di energia, detti al punto precedente, al fine di:

- suddividere l'energia consumata dall'utente in accordo al programma tariffario in vigore;
- determinare i valori di potenza media per il controllo della potenza contrattuale, per l'attuazione di procedure di alleggerimento di carico selettive e per il calcolo sia dei valori massimi assorbiti in accordo con il programma tariffario in vigore che di parametri statistici utili ai fini della determinazione del contributo alla punta dell'utente e della pianificazione dello sviluppo della rete elettrica.

E. Elaborazione di informazioni utili alla conoscenza della qualità del servizio e di eventuali tentativi di frode.

F. Gestione di procedure finalizzate all'inizializzazione dei parametri di funzionamento.

G. Gestione delle procedure di colloquio bidirezionale (rif. TAB. ENEL DH064) con l'Apparato di Cabina Secondaria (ACS) (rif. TAB. ENEL DH036) o con altre UEP.

H. Gestione del comando di apertura dell'interruttore magnetotermico.

I. Gestione di procedure di presentazione dati sul display.

L. Invio di informazioni, sotto il controllo dell'Apparato di Cabina Secondaria (ACS), ad un Terminale di Utente (TU) opzionale (rif. TAB. ENEL DH032).

M. Gestione di un'attività autodiagnostica finalizzata alla disponibilità funzionale dell'apparato e alla salvaguardia dei dati elaborati.

Ogni compito viene formalizzato mediante un'opportuna classe componente della classe “compitiDellaUE” (UE per brevità) secondo lo schema di figura 2.5.1.

Specifichiamo ora l’architettura funzionale delle singole UE come eredi della UE genitore.

[4]Si veda la sezione 1 della parte 3 per alcune spiegazioni e commenti sulla distinzione tra architettura fisica e architettura funzionale.

2.5.1 Architettura funzionale e compiti delle UEP

La UEP eredita da UE identica struttura funzionale (ha esettamente gli stessi compiti). Tuttavia ne modifica alcuni. Precisamente:

Rimangono inalterati

(*** omissis ****)

Sono invece ridefiniti:

GestioneBaseDeiTempi, ridefinita come GestioneBaseDeiTempi.P

(*** omissis ****)

2.5.2 Architettura funzionale e compiti delle UEPB

La UEPB eredita da UE identica struttura funzionale (ha esettamente gli stessi compiti). Tuttavia ne modifica alcuni. Precisamente:

Rimangono inalterati

(*** omissis ****)

Sono invece ridefiniti:

GestioneBaseDeiTempi, ridefinita come GestioneBaseDeiTempi_PB

(*** omissis ****)

2.5.3 Architettura funzionale e compiti delle UEPM

La UEPM eredita da UE identica struttura funzionale (ha esettamente gli stessi compiti). Tuttavia ne modifica alcuni. Precisamente:

Rimangono inalterati

(*** omissis ****)

Sono invece ridefiniti:

GestioneBaseDeiTempi, ridefinita come GestioneBaseDeiTempi_PM

(*** omissis ****)

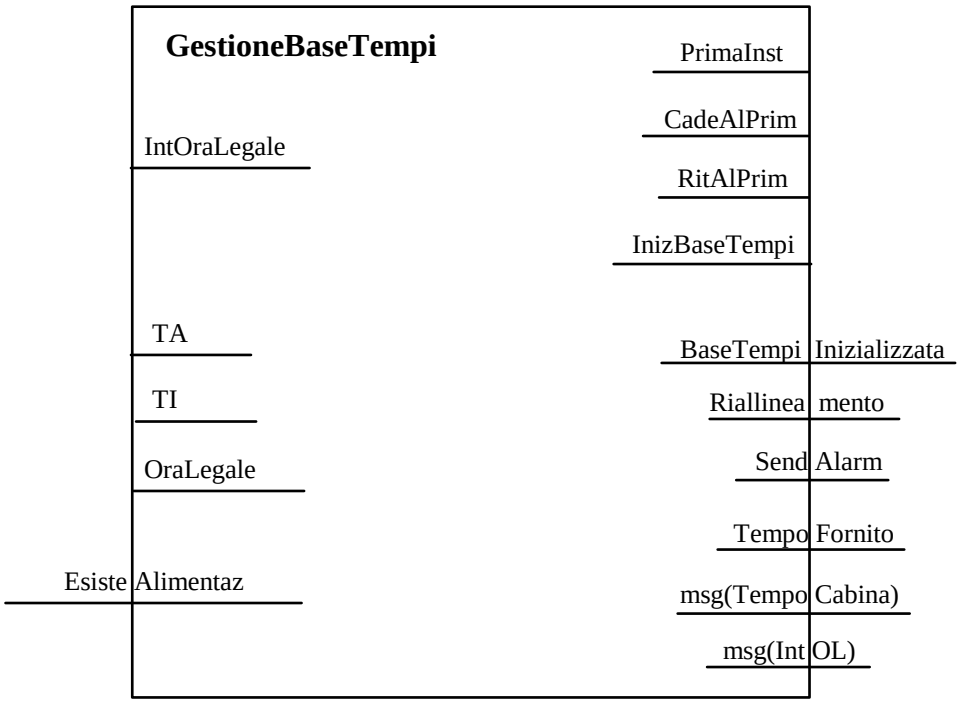


Figura 2.6.1.1 Struttura della classe GestioneBaseTempi

type TempoStrutturato**:**

record secondo, minuto, ora, giorno, mese, anno: integer

endrecord

IntervalloOraLegale:

record inizio, fine: **record**

secondo, minuto, ora, giorno, mese: integer

endrecord

endrecord

class GestioneBaseTempi

Visible: TempoFornito, EsisteAlimentaz, msg(TempoCabina), msg(IntOL), sendAlarm, BaseTempiInzializzata

t**emporal domain** real

2.6 Prescrizioni funzionali

Questa sezione specifica in dettaglio tutti i compiti che costituiscono la classe “UE”. Seguendo lo schema enunciato in precedenza, per ogni compito, definito mediante una classe corrispondente, verranno in primo luogo specificati i requisiti comuni a ogni UE; successivamente, essi verranno eventualemente specializzati tramite ereditarietà alle singole unità.

2.6.1 Gestione della Base dei Tempi

Le UE devono gestire la funzione di orologio atto a conteggiare i seguenti dati temporali: secondi, minuti, ore, giorno, settimana, mese, anno, con ora legale e ciclo quadriennale bisestile.

I requisiti comuni a ogni UE sono definiti precisamente dalla seguente definizione della classe gestioneBaseTempi la cui veste grafica è fornita in figura 2.6.1.1 e la cui formalizzazione testuale in TRIO è riportate di seguito ad essa nelle pagine pari e spiegata informalmente nelle pagine dispari ad esse affacciate. Essa fa uso degli elementi seguenti atti a descrivere le variabili in gioco e il loro trattamento:

TempoStrutturato indica il modo di rappresentare un valore temporale in tutte le sue componenti: secondo, minuto, ora, giorno, mese, anno.

IntervalloOraLegale indica una coppia di valori temporali (senza la componente anno) che marcano il momento di inizio e fine dell'ora legale.

Gli elementi di interfaccia della classe sono *TempoFornito*, *msg*(*TempoCabina*), *msg*(*IntOL*), *sendAlarm*, *EsisteAlimentaz*, *BaseTempiInzializzata*, la cui natura e il cui uso verranno specificati

tra breve.

La definizione della classe adotta un dominio temporale continuo. Ciò non significa che le diverse grandezze usate per misurare e indicare il tempo siano pure continue.

TI items

const:

epsilon, delay: integer

function

conv(TempoStrutturato): Integer

concInv(Integer): TempoStrutturato

TD items

vars TA: real

TI: integer

TempoFornito, TempoCabina: TempoStrutturato

IntOraLegale: IntervalloOraLegale

event items

PrimaInst,
CadeAlPrim,
RitAlPrim,
InzizBaseTempi,
PerditaIniz,
msg(TempoStrutturato)
msg(IntervalloOraLegale)
sendAlarm

state items

predicates

BaseTempiInzializzata, EsisteAlimentaz, OraLegale, Riallineamento

Gli elementi della classe tempo-invarianti sono:

- La costante *delay*, che denota un valore temporale fissato utilizzato per indicare il ritardo per l'esecuzione di alcune procedure;
- La costante *epsilon* che denota l'errore tollerato per certi valori temporali calcolati;
- La funzione *conv* che converte un valore di tempo espresso in secondi, minuti, ... in un valore intero assoluto riferito ad un'unità (più fine dei secondi) e un'origine dei tempi fissata convenzionalmente;
- La funzione *convInv*, inversa di *conv*.

Gli elementi della classe tempo-varianti sono:

Le variabili sono:

- *TA*, che indica un "tempo assoluto": esso è un valore di riferimento coincidente con il dominio temporale di TRIO (questo requisito verrà imposto mediante un opportuno assioma);
- *TI*, che indica un "tempo interno", ossia il valore del clock (del microprocessore) misurato attraverso l'unità di misura minima (secondo o sua frazione, a seconda del livello di risoluzione richiesto);

  Un opportuno assioma imporrà che l'origine dei tempi (fissata convenzionalmente) coincida per TA e per TI; successivamente la loro evoluzione sarà regolata dalla dinamica del sistema;
- *TempoCabina* e *TempoFornito* che indicano, rispettivamente, il valore ricevuto dall'ACS e quello fornito dall'UE, espressi in secondi, minuti, ... Ovvie relazioni stabiliranno –mediante le funzioni conv e convInv– la corrispondenza tra TI e TempoFornito (sono formulazioni diverse della stessa informazione);
- *IntOraLegale*, che indica il valore attuale dell'intervallo di aplicazione dell'ora legale.

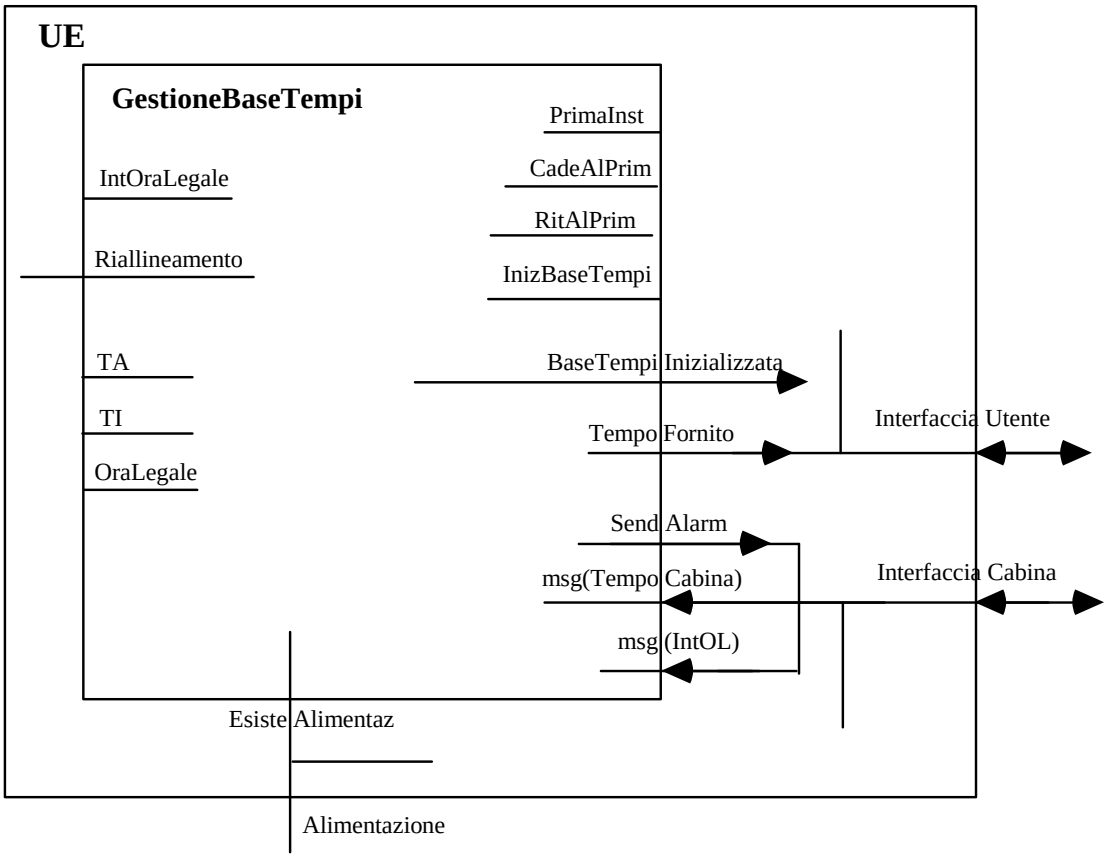
Gli eventi sono:

- *PrimaInst*, che denota l'avvenimento di Prima installazione; questo avvenimento può in realtà accadere più di una volta durante la "vita" dell'apparecchio;
- *CadeAlPrim*, che denota la caduta dell'alimentazione primaria;
- *RitAlPrim*, che denota il ritorno dell'alimentazione primaria;
- *InzizBaseTempi*, che denota l'inizializzazione della base dei tempi;
- *PerditaIniz*, che denota la perdita dell'inizializzazione della base dei tempi;
- *msg*(*TempoCabina*), che denota l'arrivo di un messaggio dall'ACS recante il valore del tempo-cabina, indicato come parametro dell'evento;
- *msg*(*IntervalloOraLegale*), che denota l'arrivo di un messaggio dall'ACS recante il valore da adottare per inizio e fine dell'ora legale;
- *SendAlarm*, che denota l'invio all'ACS di un segnale di allarme riguardante un eccessivo disallineamento tra il TempoCabina e il TempoInterno.

Gli stati sono:

- *BaseTempiInzializzata*, che denota la sussistenza dell'inizializzazione della base dei tempi;
- *EsisteAlimentaz*, che denota l'esistenza dell'alimentazione primaria;
- *OraLegale*, che denota il fatto che l'ora legale sia o meno in vigore;
- *Riallineamento*, che denota il fatto che è in corso una procedura di riallineamento tra

TempoCabina e tempo interno.



Legenda Quando diversi tratti di linea confluiscono in un unico tratto esterno, ciò significa che il tratto esterno rappresenta un'informazione articolata in diverse componenti rappresentate dai tratti confluenti. Tratti uscenti senza etichetta (tratteggiati) indicano l'esistenza di ulteriori componenti che non riguardano la classe GestioneBaseTempi ma potranno intervenire in altre classi. A esempio i messaggi provenienti dalla cabina conterranno il valore del tempo fornito dalla cabina e l'intervallo di applicazione dell'ora legale; vi saranno però altri messaggi che non riguardano la GestioneBaseTempi.

Figura 2.6.1.2 Connessioni tra la classe GestioneBaseTempi e la classe UE

Gli elementi di interfaccia della classe GestioneBaseTempi sono connessi con gli elementi della classe UE come indicato dalla figura 2.6.1.2. Ossia:

- La variabile *TempoFornito* fa parte delle informazioni fornite all'utente;
- I messaggi contenenti il valore del tempo e lo scatto dell'ora legale fanno parte delle informazioni provenienti dalla cabina;
- *SendAlarm* appartiene pure all'interfaccia tra UE e Cabina[5];
- L'esistenza dell'alimentazione è un'informazione legata allo stato complessivo della rete di alimentazione.

[5]Si ricorda che TRIO non distingue formalmente elementi di input da elementi di output nelle interfacce, ma permette un uso informale e opzionale di frecce direzionali.

axioms

/*assiomi preliminari*/

var x, y: real
AllTA**:** **for every** x ((TA = x) ---> AlwF(Dist (TA = x+y, y)))

/*Se necessario/opportuno:
PrimaInst <---->
(TA = 0 **and** TempoFornito = TempoCabina = ValoreIniziale)
*/

(omissis)

Presentazione del tempo:
TempoFornito = conInv (TI)

```
/*La definizione esplicita e completa della funzione conv è omessa*/
```

Gli assiomi della classe specificano completamente il comportamento della classe GestioneBaseTempi.

Per facilitarne la comprensione e la gestione essi sono partizionati in:

- assiomi preliminari, che speciifcano proprietà generali delle variabili utilizzate;
- assiomi specifici della UE.

Si noti che, in un ambiente di specifica completo, gran parte degli assiomi preliminari sarebbero predefiniti in opportune classi di libreria e quindi non dovrebbero essere ripetuti all'interno di definizioni di singole classi.

Alcuni assiomi preliminari sono i seguenti:

AllTA (Allineamento TA): specifica che, a meno di uno spostamento iniziale, TA coincide con il tempo di TRIO. TA serve per avere un riferimento temporale assoluto, visto che i termini temporali di TRIO sono sempre relativi all'istante corrente. Se necessario o opportuno, l'assioma potrebbe essere ulteriormente rinforzato fissando l'origine dei tempi in un convenzionale evento “Origine”, eventualmente coincidente con PrimaInst, se questo potesse avvenire una sola volta all'inizio del funzionamento dell'apparecchio.

Altri assiomi, la cui formalizzazione è omessa per brevità e perché di tipo standard, specificano le relazioni tra eventi e stati: ad esempio, il passaggio da EsisteAlimentaz a **not** EsisteAlimentaz è marcato dall'evento CadeAlPrim; viceversa il passaggio da **not** EsisteAlimentaz a EsisteAlimentaz è marcato dall'evento RitAlPrim (si veda la figura 2.6.1.3); similmente la transizione nello stato BaseTempiInzializzata corrisponde all'evento InizBaseTempi, ...

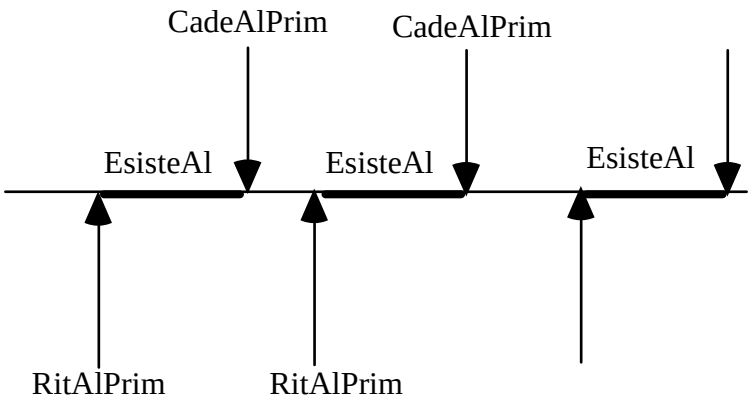


Figura 2.6.1.3 Cambiamenti di stato marcati da eventi

Altri assiomi escludono la contemporaneità tra alcuni eventi che potrebbero generare potenziali conflitti nella specifica: PrimaInst, CadeAlPrim, msg(TempoCabina) (si veda la sezione 2 della parte 3 per un commento esplicativo).

Altri assiomi infine specificano le funzioni di conversione tra tempi indicati da un solo numero e tempi strutturati (indicati da secondi, minuti, ecc.). Esse sono definite in modo da tenere in conto dell'eventuale vigore dell'ora legale.

/*assiomi specifici della UE*/

var x, y, z: real

ReqBase: = **def**

for every x, y

(y ≤ $1_{[mese]}$) **and** (|TI - TA| = x) **and**

Lasts (**not** (Riallineamento **or** InizBaseTempi), y) --->

Lasts (|TI - TA| < x+1", y)

Inizializzazione della base dei tempi:

EsisteAlimentazione **and** msg (TempoCabina) **and**
UpToNow (**not** BaseTempiInizializzata)) <----->
InizBaseTempi **and** TI = conv(TempoCabina)

Mantenimento della base dei tempi:

for every x BaseTempiInizializzata **and** $Lasts_{ie}$(EsisteAlimentazione, x)
----> $Lasts_{ie}$(BaseTempiInizializzata, x)

Funzionamento a base dei tempi inizializzata:

for every z

$Lasts_{ie}$(BaseTempiInizializzata , z) ---> Futr(ReqBase, z)

Funzionamento a base dei tempi non inizializzata:

for every z
not BaseTempiInizializzata **and** RitAlPrim ----> TI = 0
and
$Lasts_{ie}$(**not** BaseTempiInizializzata **and** EsisteAlimentaz, z) ---> Futr(ReqBase, z)

PrimaInstallazione ---> Until (**not** BaseTempiInizializzata, InizBaseTempi)

Gestione dell'ora legale

OraLegale <---->
(BaseTempiInizializzata **and**
IntOraLegale.inizio ≤ TempoFornito.<ora, giorno, mese> ≤ IntOraLegale.fine)

EsisteAlimentazione **and** msg (IntOL) ------> IntOraLegale = IntOL

and until (IntOraLegale = IntOL, EsisteAlimentazione **and** ∃ IntOL msg (IntOL))

PrimaInstallazione ---> IntOraLegale.inizio = /*ultima domenica di marzo*/ **and** IntOraLegale.fine = /*ultima domenica di settembre*/

Gli assiomi specifici della UE sono i seguenti

Preliminarmente, indichiamo con *ReqBase* (Requisito base) la specifica che, nell'arco di un mese, il valore di TI non può variare la propria differenza da TA di più di un secondo, a meno che durante tale periodo non venga effettuata una procedura di riallineamento o un'inizializzazione della base dei tempi che alterano esplicitamente il valore di TI. Questo assioma non deve valere in assoluto, ma essere applicato in modi diversi a seconda delle circostanze. La formula accanto va perciò intesa come *definizione* del Requisito base.

Inizializzazione della base dei tempi:

L'inizializzazione della base dei tempi avviene, sussistendo l'alimentazione e non essendo la base dei tempi già inizializzata, al momento in cui arriva un messaggio dall'ACS; in quel momento il tempo interno viene allineato al tempo cabina, previa opportuna conversione di formato.

Mantenimento della base dei tempi:

La base dei tempi rimane inizializzata finché permane l'alimentazione.

Funzionamento a base dei tempi inizializzata:

Finché perdura l'inizializzazione deve valere il requisito base.

Le varie unità devono poi comportarsi in maniera tra loro diversa nel caso di disallineamento tra TempoCabina e tempo interno. Questi requisiti verranno perciò specificati solo nelle corrispondenti classi eredi.

Funzionamento a base dei tempi non inizializzata:

Quando la base dei tempi non è inizializzata, non appena ritorna l'alimentazione, il tempo interno viene settato a 0; da allora in poi, finché rimane l'alimentazione e la base dei tempi non viene inizializzata, deve valere il requisito base.

Si noti che il requisito base deve sussistere sia che la base dei tempi sia inizializzata, sia che non lo sia; il valore di TI può cambiare in maniera discontinua e al di fuori del requisito base solo in corrispondenza di eventi particolari (caduta o ritorno dell'alimentazione; procedura di riallineamento)

Inizialmente, e finché non giunge un messaggio dall'ACS, la base dei tempi non è inizializzata.

Gestione dell'ora legale

Quando la base dei tempi è inizializzata il predicato di stato OraLegale deve essere vero se e solo se il valore attuale del tempo interno, trascurando la componente “anno” è compreso tra i valori inizio e fine di IntOraLegale. Convenzionalmente, quando la base dei tempi non è inizializzata si assume

che OraLegale sia falso.

Il valore di IntOraLegale è aggiornato ogni volta che giunge un messaggio apposito dalla cabina e mantiene il valore ottenuto fino all'arrivo del prossimo messaggio.

Inizialmente esso viene computato secondo la regola europea: inizio = ultima domenica di marzo, fine = ultima domenica di settembre.

Altre funzionalità (interrupts a tempo)

(Omissis)

/*Le altre funzionalità in oggetto non vengono formalizzate poiché le indicazioni fornite dai documenti originari non sono sufficientemente circostanziate per definirne precisamente scopo e significato. Al più si potrebbero definire dei nuovi eventi “interrupt”, verosimilmente appartenenti all'interfaccia della classe (ossia visibili all'esterno) riservandosi di specificarne la semantica mediante opportuni assiomi in altre classi o in questa stessa qualora venissero fornite specifiche informali più precise*/

endclass GestioneBaseTempi

Altre funzionalità

L'attività di gestione dell'orologio deve emettere opportune indicazioni (interrupt a tempo) verso le attività di elaborazione delle informazioni da parte della stessa UE.

Tra queste indicazioni assumono particolare importanza quelle necessarie per :

- il calcolo dei valori medi della potenza attiva;
- la determinazione della fascia tariffaria in atto;
- la memorizzazione dei dati calcolati al termine del periodo programmato per la fatturazione o in corrispondenza di date intermedie;
- l'esecuzione dei comandi dilazionati contenenti l'informazione di data e ora di attuazione inviati dall' ACS.

class GestioneBaseTempi_P

inherits GestioneBaseTempi

axioms

Perdita di inizializzazione

UpToNow (BaseTempiInizializzata) **and** CadeAlPrim ---> PerditaIniz

Gestione disallineamenti

var TC: integer; y: real

(Lasts(BaseTempiInizializzata, delay) **and** msg(TempoCabina) **and**
TA = y **and** conv(TempoCabina) = TC **and** 1' < |TI - TC| < 5'
<-------->
$Lasts_{ie}$(Riallineamento, delay))
and
($Lasts_{ie}$(Riallineamento, delay) ---->
within (|TI - (TC + TA -y)| < epsilon, delay))

BaseTempiInizializzata **and** msg(TempoCabina) **and**
|TI - conv(TempoCabina)| > 5'
------>
sendAlarm

endclass GestioneBaseTempi_P

2.6.1.1 Gestione Base dei Tempi per la UEP

La UEP mantiene tutti i requisiti della UE per la gestione della base dei tempi.

In aggiunta essa impone:

Perdita di inizializzazione esplicita

Non appena si abbia una caduta di alimentazione si perde l'inizializzazione della base dei tempi. Infatti l'UEP non ha alimentazione interna e la DH023 stabilisce esplicitamente che l'allineamento del tempo venga mantenuto *solo* in presenza di alimentazione.

Gestione disallineamenti

Se, quando la base dei tempi è inizializzata, giunge un valore del tempo di cabina –TC– che differisce dal tempo interno di un valore compreso tra un minuto e 5 minuti, allora –e solo allora– ha inizio una procedura di riallineamento che dura un tempo costante “delay”. Assumendo che durante tale intervallo non si perda l'inizializzazione della base dei tempi, prima della sua fine il tempo interno deve essere riallineato, a meno di un errore tollerato epsilon, al tempo di cabina.

La durata “delay” della procedura di riallineamento non è specificata in questa sezione della DH023, ma viene delimitata dalla definizione precisa della procedura di riallineamento fornita altrove.

Se invece giunge un valore del tempo di cabina che differisce dal tempo interno di un valore maggiore di 5 minuti, occorre mandare un segnale di allarme alla cabina.

class GestioneBaseTempi_PB

inherits GestioneBaseTempi

axioms

Il mantenimento dell'inizializzazione

BaseTempiInizializzata **and** EsisteAlimentaz

----> $Lasts_{ie}$(BaseTempiInizializzata, 48_h)

Gestione disallineamenti

var TC: integer; y: real

(Lasts(BaseTempiInizializzata, delay) **and** msg(TempoCabina) **and** TA = y **and** conv(TempoCabina) = TC **and** 1' < |TI - TC| < 5'

<-------->

$Lasts_{ie}$(Riallineamento, delay))

and

($Lasts_{ie}$(Riallineamento, delay) ---->

within (|TI - (TC + TA -y)| < epsilon, delay))

BaseTempiInizializzata **and** msg(TempoCabina) **and**

|TI - conv(TempoCabina)| > 5'

------>

SendAlarm

endclass GestioneBaseTempi_PB

2.6.1.2 Gestione Base dei Tempi per la UEPB

La UEP mantiene tutti i requisiti della UE per la gestione della base dei tempi.

In aggiunta essa impone:

Il mantenimento dell'inizializzazione :

La base dei tempi deve rimanere inizializzata per almeno 48 ore anche in assenza di alimentazione primaria.

Gestione disallineamenti

La gestione dei disallineamenti da parte della UEPB è identica a quella della UEP.

class GestioneBaseTempi_PM

inherits GestioneBaseTempi

axioms

Il mantenimento dell'inizializzazione

BaseTempiInizializzata **and** EsisteAlimentaz

----> $Lasts_{ie}$(BaseTempiInizializzata, 48_h)

Gestione disallineamenti

BaseTempiInizializzata **and** msg(TempoCabina) **and**

|TI - conv(TempoCabina)| > 1'

------>

SendAlarm

endclass GestioneBaseTempi_PM

2.6.1.3 Gestione Base dei Tempi per la UEPM

La UEPM mantiene tutti i requisiti della UE per la gestione della base dei tempi.

In aggiunta essa impone:

Il mantenimento dell'inizializzazione:

La base dei tempi deve rimanere inizializzata per almeno 48 ore anche in assenza di alimentazione primaria.

Gestione disallineamenti

Nel caso di un disallineamento superiore a 1' occorre inviare immediatamente un segnale di allarme alla cabina.

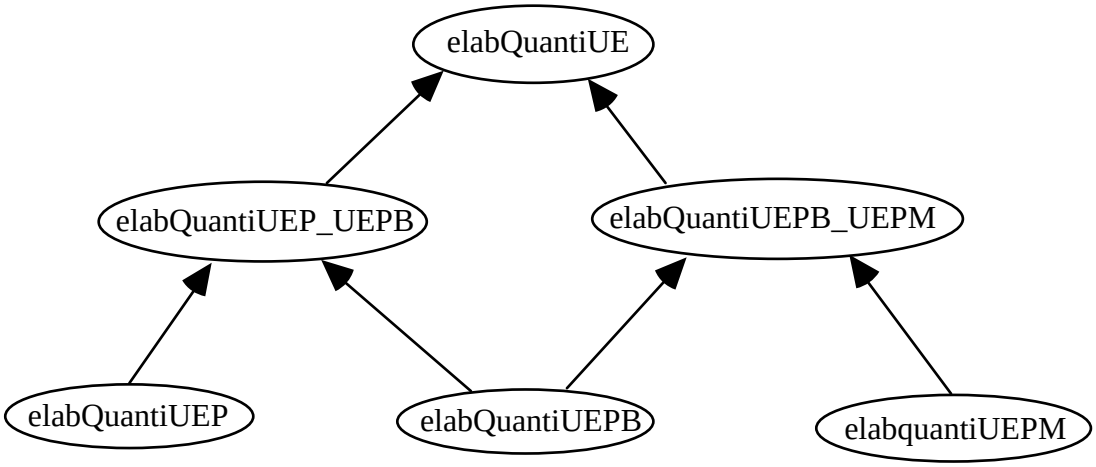


Figura 2.6.2.1 Gerarchia di ereditarietà tra le classi per l'elaborazione dei quanti di energia.

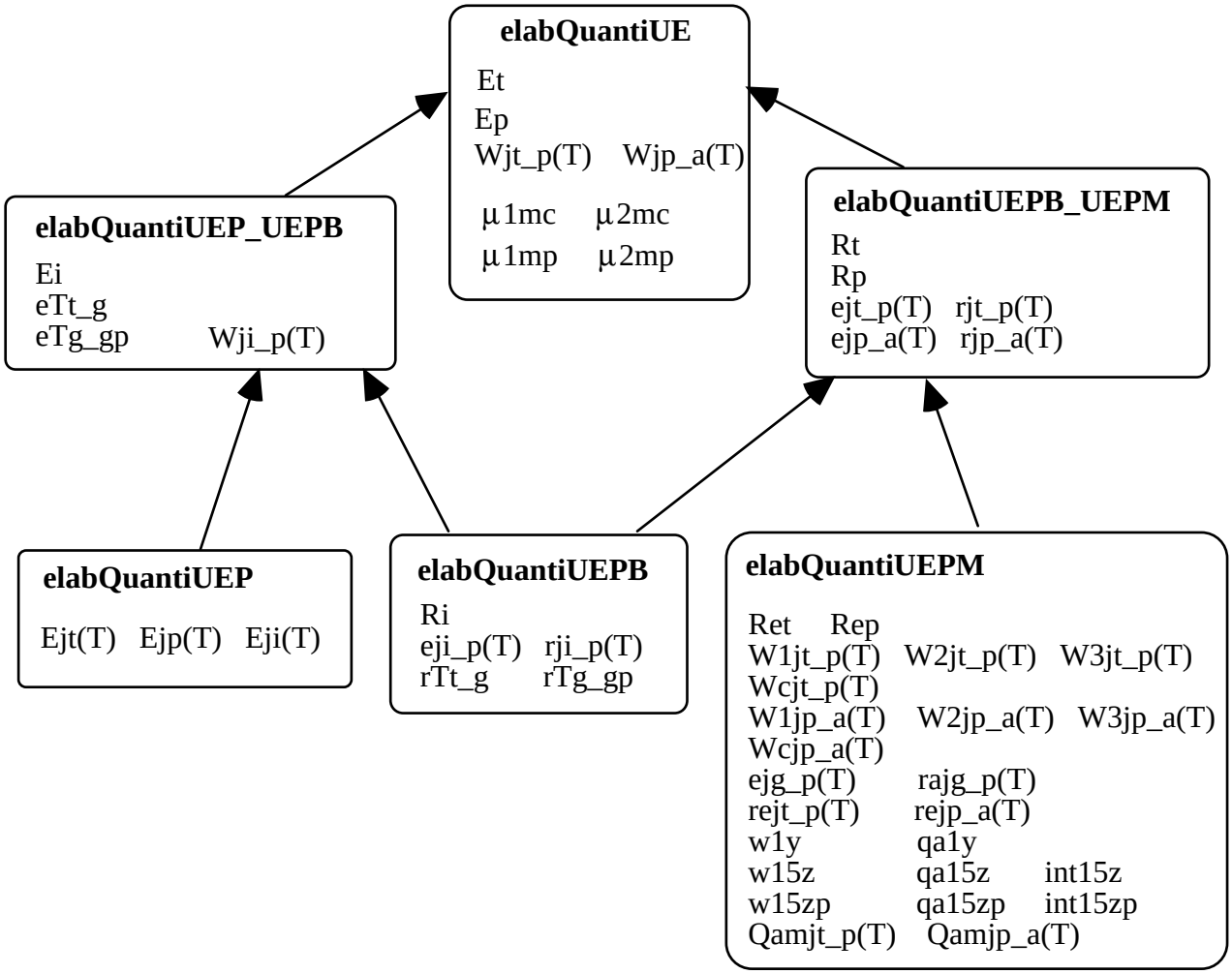


Figura 2.6.2.2 Rappresentazione dettagliata della figura 2.6.2.1, che evidenzia le entità presenti nelle varie classi.

2.6.2 Elaborazione dei Quanti di Energia

La funzione di elaborazione dei quanti di energia calcola un gran numero di quantità (misurazione di valori istantanei e di calcolo della media di varie categorie di energia e di potenze), e presenta importanti similitudini, ma anche significative differenze, nei diversi tipi di unità di elaborazione. Per questo la specifica di tale funzione, rappresentata, in accordo all'architettura funzionale delineata in sezione 2.5, da una modulo di nome *ElaborazioneQuanti*, è stata organizzata mediante una gerarchia di ereditarietà, riportata in figura 2.6.2.1. In figura 2.6.2.2 è riportata una rappresentazione più dettagliata della gerarchia, che evidenzia le entità presenti nelle varie classi.

Mediante tale gerarchia si sono evidenziate tutte le parti comuni a ogni possibile sottoinsiene dei tipi di unità di elaborazione:

- la classe *elabQuantiUE* contiene le entità comuni a tutti i tipi di unità (cioè comuni alla UEP, alla UEPB, e all'UEPM);
- la classe *elabQuantiUEP_UEPB* contiene le entità comuni alla UEP e alla UEPB, ma non presenti nella UEPM;
- la classe *elabQuantiUEPB_UEPM* contiene le entità comuni alla UEPB e alla UEPM, ma non presenti nella UEP;
- infine, le tre classi *elabQuantiUEP*, *elabQuantiUEPB* ed *elabQuantiUEPM* contengono le entità caratteristiche delle tre unità UEP, UEPB e UEPM, cioè presenti in ognuna di esse ma in nessuna delle altre due.

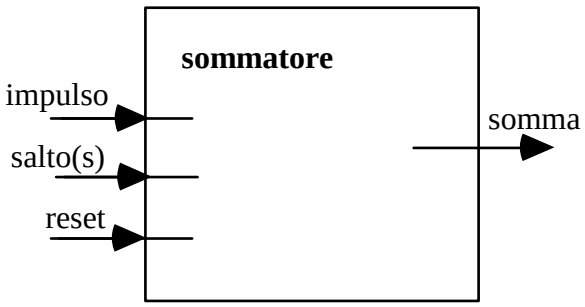


Figura 2.6.2.3 Rappresentazione grafica della classe *sommatore*.

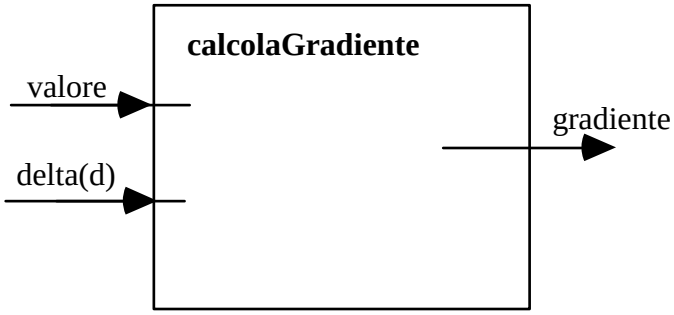


Figura 2.6.2.4 Rappresentazione grafica della classe *calcolaGradiente*.

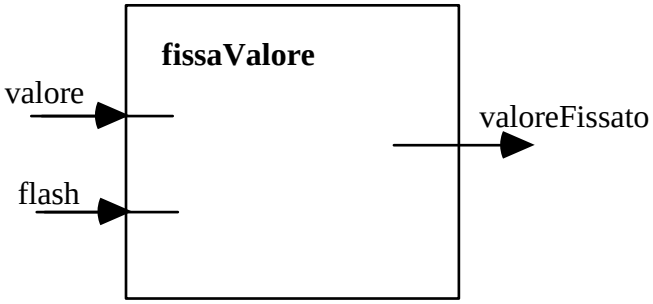


Figura 2.6.2.5 Rappresentazione grafica della classe *fissaValore*.

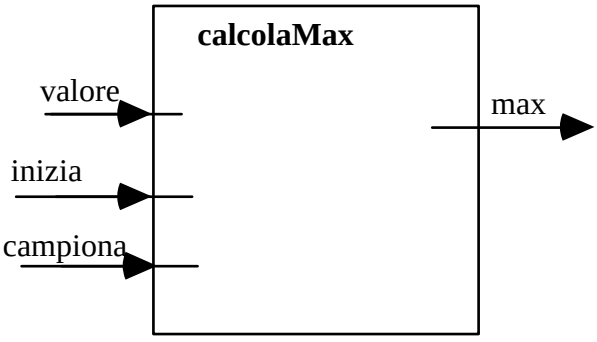


Figura 2.6.2.6 Rappresentazione grafica della classe *calcolaMax*.

2.6.2.1 Individuazione delle componenti base della specifica

La maggior parte delle elaborazioni concernenti la funzione di elaborazione dei quanti può essere ricondotta a un insieme assai limitato di operazioni tipiche, svolte in modo simile su una grande varietà di grandezze diverse.

Allo scopo di massimizzare la riutilizzabilità delle specifiche, e favorire le loro qualità di astrattezza e compattezza, sono quindi state introdotte delle classi ausiliarie che descrivono, *una tantum* e in modo generale e astratto, tale ristretto numero di operazioni fondamentali.

L'operazione fondamentale, sulla quale tutte le altre sono basate, è l'operazione di conteggio: tutte le operazioni di calcolo dell'energia consumata (totale o ripartita sulle varie tariffe, cumulativa a partire dall'installazione o relativa a determinati periodi), si basa sul conteggio del numero dei quanti, di varie categorie, originati dai trasduttori optoelettronici.

Allo scopo di modellare tale operazione è stata introdotta la classe *sommatore,* mostrata in figura 2.6.2.3. ***sommatore*** specifica il calcolo della grandezza *somma* come valore totale del numero di volte in cui accade l'evento *impulso* sommato al valore totale degli argomenti del predicato *salto*(*s*) tutte le volte in cui esso si verifica; l'evento *reset* provoca la reinizializzazione di *somma* al valore nullo. Si noti che l'elemento *impulso* permette di effettuare conteggi di tipo strettamente incrementale, cioè di aggiungere un'unità al valore della somma, mentre *salto* permette di modellare tutte le circostanze in cui il conteggio procede mediante una discontinuità con salti non unitari.

L'altra operazione fondamentale, sulla quale sono basati tutti i calcoli di potenza impegnata, è quella di calcolo di gradiente temporale, o meglio di rapporto incrementale, visto che vengono sempre calcolate potenze medie. L'operazione tipica consiste nel rapportare la variazione di un conteggio di un numero di quanti di energia a un determinato intervallo di tempo, ottenendo un rapporto che, moltiplicato per opportune costanti di proporzionalità, fornisce un valore di potenza media impegnata. A tale scopo si introduce la classe ***calcolaGradiente*** mostrata in figura 2.6.2.4, che, quando il predicato *delta*(*d*) è vero, fornisce la variazione dell'elemento *valore* nel più recente intervallo lungo *d* rapportata al valore di *d*.[6]

Altra operazione fondamentale consiste nel memorizzare i valori istantanei di certe grandezze calcolate in registri non volatili, per essere utilizzati in elaborazioni successive o comunicate ad altri dispositivi. Per modellare tale operazione si introduce la classe ***fissaValore***, mostrata in figura 2.6.2.5, che "fotografa" l'elemento *valore* agli istanti in cui vale il predicato *flash*: l'elemento *valoreFissato* è sempre uguale al *valore* nell'ultimo istante in cui è accaduto *flash*.

Un'ultima operazione di base è relativa al calcolo del valore massimo di una grandezza in un certo

[6] Si noti che *calcolaGradiente* assume che l'intervallo temporale su cui calcolare il rapporto incrementale dia misurato in ore.

intervallo, frequentemente effettuata per ricavare parametri statistici sui consumi di un'utenza. La classe ***valoreMax***, mostrata in figura 2.6.2.6, stabilisce il valore massimo di una serie temporale di *valore* nell'intervallo tra due eventi *inizia* prendendo i valori negli istanti in cui vale *campiona*.

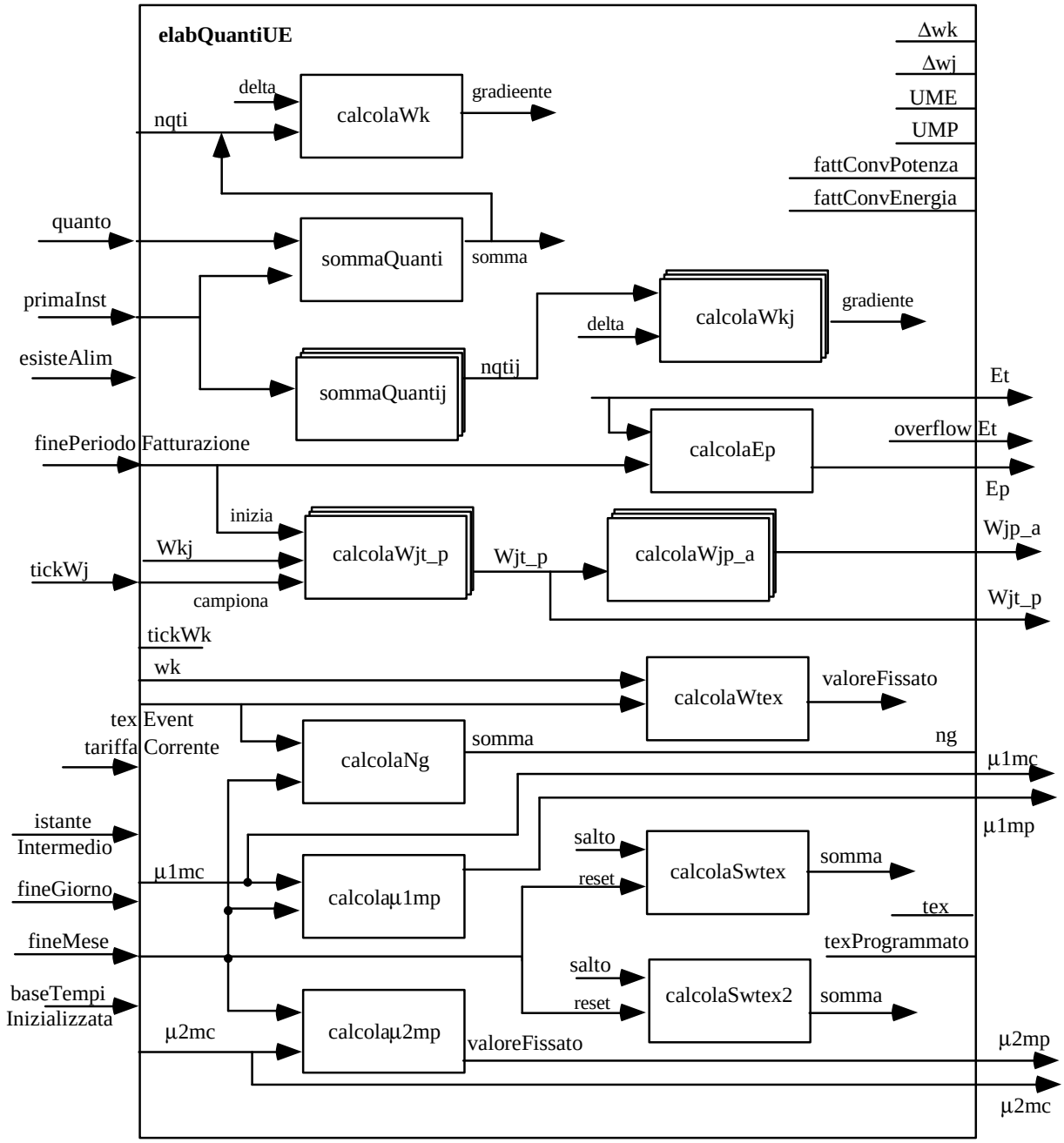


Figura 2.6.2.7 Rappresentazione grafica della classe *elabQuantiUE*.

2.6.2.2 Funzionalità comuni a tutti i tipi di unità di elaborazione

La classe *elabQuantiUE*, la cui rappresentazione grafica è riportata in figura 2.6.2.7, occupa la posizione più elevata nella gerarchia mostrata in figura 2.6.2.1, e specifica le funzionalità comuni a tutti i tipi di UE.

Essa contiene dichiarazioni di elementi e moduli per la specifica del calcolo delle quantità mostrate sulle frecce uscenti dal lato destro della rappresentazione grafica di figura 2.6.2.7, a partire dai dati e segnali mostrati come ingesso sul lato sinistro.

type Tariffa = {T1, T2, T3, T4};

class elabQuantiUE

visible Et, overflowEt, Ep, wjt_p, wjp_a, µ1mc, µ1mp, µ2mc, µ2mp
quanto, primaInst, esisteAlim, finePeriodoFatturazione, tickWk,
tariffaCorrente, istanteIntermedio, fineGiorno, fineMese,
baseTempiInizializzata

TD Items

vars Et: natural;

Ep: natural;

functions Wjt_p(Tariffa): natural;

Wjp_a(Tariffa): natural;

vars µ1mc, µ2mc: natural;

µ1mp, µ2mp: natural;

event Items primaInst;
quanto;
istanteIntermedio;
fineGiorno;
fineMese;
finePeriodoFatturazione;
overflowEt;

Dichiarazione globale del tipo *Tariffa*, che rappresenta le quattro tariffe; è valida per tutte le classi relative all'elaborazione dei quanti e alla gestione delle interruzioni.

Per comodità di lettura, le entità dichiarate nella classe vengono raggruppate a seconda che rappresentino: grandezze da calcolare e memorizzare in registri non volatili (queste a loro volta suddivise in misurazioni di energia, di potenza e parametri statistici), eventi cha cadenzano il funzionamento del contatore, stati del contatore, variabili fisico-matematiche ausiliarie alla definizione dei valori calcolati, costanti e parametri dell'attività.

Grandezze da calcolare e memorizzare in registri non volatili: misurazione dell'energia.

Et rappresenta il valore complessivo dell'energia conteggiata dall'UE a partire dall'istante iniziale fino all'istante corrente.

Ep rappresenta il valore complessivo dell'energia conteggiata dall'UE a partire dall'istante iniziale fino alla fine del periodo di fatturazione precedente quello in corso.

Grandezze da calcolare e memorizzare in registri non volatili: misurazione della potenza.

Wjt_p(*Tariffa*) denota i valori massimi della potenza attiva media in periodi sincroni lunghi *Δwj*, assorbita dall'utente in ciascuna tariffa, a partire dall'istante di fine del periodo di fatturazione precedente fino all'istante corrente.

Wjp_a(*Tariffa*) rappresenta i valori massimi della potenza attiva media in intervalli sincroni lunghi *Δwj*, assorbita dall'utente in ciascuna tariffa nel periodo di fatturazione precedente a quello in corso.

Grandezze da calcolare e memorizzare in registri non volatili: parametri statistici.

μ1mc e *μ2mc* denotano i parametri per i calcoli di *load flow* statistico di rete, relativi al mese solare in corso; essi indicano rispettivamente le medie del primo e del secondo ordine delle potenze consumate in un determinato periodo sincrono di lunghezza *Δwk*, indicato mediante il suo indice *tex* all'interno della giornata (e.g., se *Δwk*=15', l'indice *tex* è un valore tra 1 e 96)

μ1mp e μ2mp sono i valori di μ1 e μ2 relativi al mese solare precedente a quello in corso

Eventi cha cadenzano il funzionamento del contatore

primaInst rappresenta la messa in opera del contatore.

quanto rappresenta la rilevazione del consumo di un quanto di energia.

istanteIntermedio denota il cadere di un istante intermedio nell'ambito di un periodo di fatturazione; NB si assume che *istanteIntermedio* cada sempre a mezzanotte.

fineGiorno denota la fine di un giorno, quindi coincide con la mezzanotte.

fineMese denota la fine di un mese, sincronizzato con la mezzanotte.

finePeriodoFatturazione (autoesplicativo) viene anch'esso assunto sincronizzato alla mezzanotte.

overflowEt denota che il conteggio di *Et* ha raggiunto il valore massimo (ricomincerà da 0 in virtù della gestione circolare dei conteggi). NB: la rilevazione dell'*overflow* di *Et* non è prevista in [DH23], ma sembra utile e ragionevole averla.

tickWk;

tickWj;

texEvent:

state Items

baseTempiInizializzata;
texProgrammato;
tariffaCorrente: Tariffa;

TD Items

vars nqti: natural;

functions nqtij(Tariffa): natural;
vars wk: natural;
wtex: natural;
Swtex, Swtex2: natural;

ng: natural;
functions wkj(Tariffa): natural;

TI Items

consts valoreQuanto: natural;

UME: natural;
UMP: natural;

fattConvEnergia: natural;

fattConvPotenza: natural;

tickWk si verifica alla scadenza di un periodo su cui calcolare *wk*; si assume che *tickWk* sia quello corretto per ogni tipo di contatore, cioè sincrono con la base dei tempi se questa è inizializzata, e asincrono (per UEPB e UEPM) tra Irlm e Iorl, cioè quando la base dei tempi non è inizializzata (in ogni caso l'intervallo su cui calcolare la potenza media è sempre lungo Δwk).

tickWj ha luogo all'istante in cui si calcolano le potenze medie *wkj*; è sincrono con cadenza di 15', quindi deve essere distinto da *tickWk* che può essere asincrono quando la base dei tempi non è inizializzata; si usa un predicato simbolico (*tickWj* invece di usare esplicitamente il predicato *quindiciPrimi*) per maggiore flessibilità e modificabilità.

texEvent si verifica quando è il momento di calcolare wtex

Stati di funzionamento del contatore

baseTempiInizializzata, autoesplicativo, è lo stesso usato nella gestione della base dei tempi.

texProgrammato indica che è correntemente abilitato il calcolo di $\mu 1$ e $\mu 2$.

tariffaCorrente indica la tariffa applicabile all'istante corrente

Variabili fisico-matematiche ausiliarie alla definizione dei valori calcolati

nqti denota il conteggio cumulativo di tutti i quanti di energia consumati dall'installazione fino all'istante corrente.

nqtij(*Tariffa*) indica il conteggio cumulativo dei quanti divisi per tariffa.

wk denota la potenza media consumata nell'ultimo periodo lungo Δwk.

wtex indica il valore di wk nell'intervallo sincrono di indice *tex*.

Swtex e Swtex2 rappresentano la sommatoria dei *wtex* e dei $wtex^2$ a partire da inizio mese fino all'istante attuale.

ng indica il numero dei giorni utilizzati per il calcolo di $\mu 1$ e $\mu 2$.

wkj(*Tariffa*) indica la potenza media, sulle varie tariffe, calcolata nell'ultimo intervallo sincrono di durata Δwj.

Costanti e parametri dell'attività

valoreQuanto indica il valore (in Wh) del quanto di energia misurato dal contatore; NB: non viene riportato, nella presente classe, alcun assioma sul valore di *valoreQuanto*: questo andrà indicato nelle classi foglia della gerarchia, cioè in *elabQuantiUEP*, *elabQuantiUEPB*, *elabQuantiUEPM*.

UME denota l'Unità di Misura dell'Energia, mentre *UMP* l'Unità di Misura della Potenza; anche i valori di *UME* e *UMP* andranno indicati solo nelle classi foglia *elabQuantiUEP*, *elabQuantiUEPB*, *elabQuantiUEPM*.

fattConvEnergia è il fattore che moltiplicato per un numero di quanti dà l'energia corrispondente, tenendo conto del valore del singolo quanto e dell'unità di misura dell'energia.

fattConvPotenza è il fattore che converte la potenza, come definita da moduli di tipo calcolaGradiente in base al numero dei quanti, nell'unità di misura desiderata; il suo valore è determinato dal valore del quanto, dall'unità di misura dell'energia e dall'unità di misura della potenza (NB *calcolaGradiente* assume che l'intervallo temporale su cui calcolare il rapporto

incrementale sia misurato in ore).

maxValEt: natural ;

Δwk: natural;

Δwj: natural;

TD Items

vars tex: natural;

modules sommaQuanti: sommatore;
sommaQuantij: **array** [T1 .. T4] **of** sommatore;

connections { (sommaQuanti.reset, primaInst))
(sommaQuantij.reset, primaInst)
(sommaQuanti.impulso, quanto),
(sommaQuanti.somma, nqti),
(sommaQuantij.somma, nqtij), }

modules calcolaEp: fissaValore;
connections { (calcolaEp.valoreFissato, Ep),
(finePeriodoFatturazione, calcolaEp.flash),
(Et, calcolaEp.valore), }

modules calcolaWk: calcolaGradiente;
connections { (nqti, calcolaWk.valore) }

modules calcolaWtex: fissaValore;
connections { (wk, calcolaWtex.valore),
(texEvent, calcolaWtec.flash),
(calcolaWtec.valoreFissato, wtex) }

modules calcolaNg: sommatore;
connections { (texEvent, calcolaNg.impulso),
(calcolaNg.somma, ng),
calcolaNg.reset, fineMese) }

modules calcolaSwtex: sommatore;
connections { (calcolaSwtex.somma, Swtex),
(calcolaSwtex.reset, fineMese) }

maxValEt indica il valore oltre il quale avviene l'*overflow* del registro che memorizza *Et*; anche *maxValEt* dipende dal tipo di contatore e viene precisato nelle classi foglia

Δwk denota la lunghezza del periodo su cui calcolare *wk*; anch'esso viene precisato nelle classi foglia

Δwj indica la lunghezza del periodo su cui calcolare le potenze medie utili al calcolo dei valori massimi di potenza *Wjt_p*, *Wjp_a* e (nell'UEP) *Wji_p*; il suo valore dipende dal tipo di unità e quindi viene precisato nelle classi foglia. NB *Δwj* viene differenziato da *Δwk* perchè esso è sempre sincrono, al contrario di quanto accade per *wk*, e anche perchè la lunghezza dell'intervallo di calcolo può essere effettivamente diversa per *wk* e per i vari *wkj*.

tex indica l'indice del periodo (di lunghezza *Δwk*) su cui calcolare *wtex*

Il calcolo delle grandezze *nqti* e *nqtij*(*T*) (cioè del numero totale cumulativo dei quanti in qualsiasi tariffa o separati per tariffa) viene specificato sinteticamente mediante moduli di classe *sommatore*, che descrivono l'operazione fondamentale di conteggio. Per *nqtij*(*T*) si utilizza un *array* di quattro moduli perchè si tratta di una funzione, che ha quattro diversi valori in corrispondenza alle quattro tariffe.

Il conteggio dei quanti (complessivo e per ogni tariffa) viene inizializzato a 0 alla prima installazione. NB non viene indicato l'evento corrispondente a *impulso* per *sommaQuantij*, poichè esso dipende dalla tariffa corrente, che viene determinata in maniera diversa per ogni tipo di unità. Questa informazione viene quindi specificata nelle classi eredi nella gerarchia.

Il calcolo della grandezza *Ep* viene specificato sinteticamente utilizzando un modulo di classe *fissaValore* che attribuisce a *Ep* il valore di *Et* al momento in cui si verifica l'evento *finePeriodoFatturazione*

Per la specifica del calcolo della grandezza *wk* si utilizza un modulo di classe *calcolaGradiente* che valuta il rapporto incrementale del conteggio *nqti*.

Il calcolo della grandezza *Wtex* viene specificato sinteticamente utilizzando un modulo di classe *fissaValore*, che attribuisce a *Wtex* il valore di *wk* al momento in cui si verifica l'evento *texEvent*

La determinazione di *ng* (il numero dei giorni utilizzato per valutare $\mu 1mc$ e $\mu 2mc$) viene specificata sinteticamente mediante un modulo di classe *sommatore*; si considera il numero di volte in cui si verifica *texEvent* a partire dalla fine del mese precedente

Il calcolo di *Swtex* (la sommatoria dei *wtex* a partire da inizio mese fino all'istante attuale) viene specificato mediante un modulo di classe *sommatore*. *Swtex* viene posto a zero a fine mese.

modules calcolaSwtex2: sommatore;
connections { (calcolaSwtex2.somma, Swtex2)
(calcolaSwtex2.reset, fineMese) }

modules calcolaμ1mp, calcolaμ2mp: fissaValore;
connections { (calcolaμ1mp.valoreFissato, μ1mp),
(calcolaμ2mp.valoreFissato, μ2mp),
(fineMese, calcolaμ1mp.flash),
(fineMese, calcolaμ2mp.flash),
(μ1mc, calcolaμ1mp.valore),
(μ2mc, calcolaμ2mp.valore)}

modules calcolaWkj: **array** [T1 .. T4] **of** calcolaGradiente;
connections { (nqtij, calcolaWkj.valore) }

modules calcolaWjt_p: **array** [T1 .. T4] **of** calcolaMax;
connections { (calcolaWjt_p.max Wjt_p),
(wkj, calcolaWjt_p.valore),
(tickWj, calcolaWjt_p.campiona),
(finePeriodoFatturazione, calcolaWjt_p.inizia) }

modules calcolaWjp_a: **array** [T1 ..T4] **of** fissaValore;
connections { (Wjt_p, calcolaWjp_a.valore),
(finePeriodoFatturazione, calcolaWjp_a.flash),
(calcolaWjpa.valoreFissato, Wjp_a) }

axioms **TI var** T: Tariffa; n: natural;

fattConvEnergia = valoreQuanto / UME

fattConvPotenza = valoreQuanto / (UME * UMP)

Et = (nqti * fattConvEnergia) MOD (maxValEt + 1)

$\text{overflowEt} \leftrightarrow (\text{UpToNow}(\text{Et} = \text{maxValEt}) \wedge \text{NowOn}(\text{Et} = 0))$

Il calcolo di *Swtex2* (la sommatoria dei quadrati dei *wtex* a partire da inizio mese fino all'istante attuale) viene specificato mediante un modulo di classe *sommatore*. Swtex2 viene posto a zero a fine mese.

Il calcolo di $\mu 1mp$ e $\mu 2mp$ viene specificato mediante due moduli di classe *fissaValore*. $\mu 1mp$ e $\mu 2mp$ sono uguali al valore assunto rispettivamente da $\mu 1mc$ e $\mu 2mc$ alla fine del mese.

Il calcolo delle varie potenze *Wkj* sulle diverse tariffe viene specificato mediante un array di moduli di classe *calcolaGradiente*, un modulo per ogni tariffa.

Il calcolo delle varie potenze *Wjt_p* sulle diverse tariffe viene specificato mediante un array di moduli di classe *calcolaMax*, un modulo per ogni tariffa. I valori della potenza *wkj* vengono campionati all'istante rappresentato dall'evento *tickWkj*, e il valore massimo viene reinizializzato alla fine di ogni periodo di fatturazione.

Il calcolo delle potenze *Wjp_a* viene specificato mediante un array di moduli di classe *fissaValore*, un modulo per ogni tariffa. I vari *Wjp_a* sono uguali al valore assunto dai rispettivi *Wjt_p* alla fine del periodo di fatturazione.

Il fattore di conversione dell'energia (che permette di ottenere l'energia equivalente a un dato numero di quanti) è uguale al rapporto tra il valore del quanto di energia (in Wh) e l'unità di misura dell'energia.

Il fattore di conversione della potenza (che converte la potenza, come definita da moduli di tipo *calcolaGradiente* in base al numero dei quanti, nell'unità di misura desiderata) è uguale al rapporto tra il valore del quanto (in Wh) e il prodotto dell'unità di misura dell'energia e l'unità di misura della potenza.

La potenza Et è proporzionale a *nqti* e cresce con esso, salvo overflow.

Si noti che i contatori nell'energia consumata sono nell'unità di misura UME quindi sono legati al numero dei quanti (eventualmente delle diverse tariffe) diviso per il numero UME. In tal modo è possibile soddisfare il requisito (cfr §5.4.3, DH23, ultimo paragrafo) di tener conto dei resti decimali in corrispondena a discontinuità nelle tariffe di fatturazione. La conseguenza naturale degli arrotondamenti di decimali, in caso di cambio di contratto, è che il consumo viene arrotondato per difetto (sugli ultimi consumi) per il cliente uscente, e per eccesso (sui primi consumi) per chi

subentra.

$(tex = 0) \times \neg texProgrammato$

$$texEvent \leftrightarrow \begin{pmatrix} tickWk \wedge \\ Past(WithinP(fineGiorno, \Delta wk), \Delta wk * (tex - 1)) \wedge \\ Lasted(esisteA\,lim, \Delta wk) \end{pmatrix}$$

$calcolaWk.delta(n) \times (tickWk \quad n = \Delta wk)$

$wk = calcolaWk.gradiente * fattConvPotenza$

$\neg calcolaNg.salto(n)$

$calcolaSwtex.salto(n) \times (texEvent \quad n = wtex)$

$\neg calcolaSwtex.impulso$

$calcolaSwtex2.salto(n) \times (texEvent \quad n = wtex^2)$

$\neg calcolaSwtex2.impulso$

$$(ng \neq 0) \rightarrow \left(\mu 1mc = \frac{Swtex}{ng} \wedge \mu 2mc = \frac{Swtex2}{ng} \right)$$

$calcolaWkj[T].delta(n) \times (n = \Delta wj \quad tickWj)$

$wkj(T) = calcoloWkj[T].gradiente * fattConvPotenza$

end elabQuantiUE.

Si assume che *tex* valga 0 se e solo se non devono essere calcolati $\mu 1$ e $\mu 2$.

L'evento *texEvent* che indica lo scoccare del periodo su cui calcolare la potenza wtex, si verifica nell'istante, tra quelli in cui si calcola *Wk*, per il quale sono passati *tex* periodi di lunghezza Δwk, e tale che nell'ultimo periodo lungo Δwk l'alimentazione è sempre stata presente. Si noti che la sottoformula $\mathrm{Past}(\mathrm{WithinP}(\mathrm{fineGiorno}, \Delta\mathrm{wk}), \Delta\mathrm{wk} * (\mathrm{tex} - 1))$ tiene conto della possibilità di un segnale di *fineGiorno* arrivato in ritardo a causa di un'interruzione. La condizione Lasted(EsisteAlim, Δwk) fa sì che nel calcolo di $\mu 1$ e $\mu 2$ si contino solo le giornate in cui l'alimentazione è stata disponibile per tutto il periodo di lunghezza Δwk e di indice *tex*.

Il calcolo di *wk* viene svolto all'istante denotato dall'evento *tickWk* su un periodo di tempo di lunghezza Δwk. Il valore definito dal modulo *calcolaGradiente* viene convertito nell'unità di misura appropriata per la potenza mediante moltiplicazione per *fattConvPotenza*.

Il conteggio dei giorni su cui calcolare $\mu 1mc$ e $\mu 2mc$ viene incrementato di uno alla volta.

Nel calcolo di *Swtex*, questo viene incrementato del valore corrente di *wtex* quando si verifica *texEvent*; l'incremento unitario di *Swtex* (in generale specificabile usando l'item *impulso* del modulo *sommatore*) non viene mai effettuato.

Nel calcolo di *Swtex2*, similmente a quanto detto per *Swtex*, *Swtex2* viene incrementato del valore corrente di *wtex* quando si verifica *texEvent*; l'incremento unitario di *Swtex* (in generale specificabile usando l'item *impulso* del modulo *sommatore*) non viene mai effettuato.

Il valore di $\mu 1mc$ (e rispettivamente di $\mu 2mc$) è definito, quando il conteggio dei giorni *ng* è positivo, come il rapporto di *Swtex* (rispettivamente, di *Swtex2*) e *ng*.

Il calcolo delle varie potenze *wkj* è del tutto analogo a quello di *wk*.

Si noti che qui si ignora il problema di come le interruzioni di alimentazione possano influire sul calcolo delle varie potenze medie: a seconda dei casi un'interruzione può accrescere o diminuire i valori di potenza media calcolata su un intervallo durante il quale è caduta l'alimentazione.

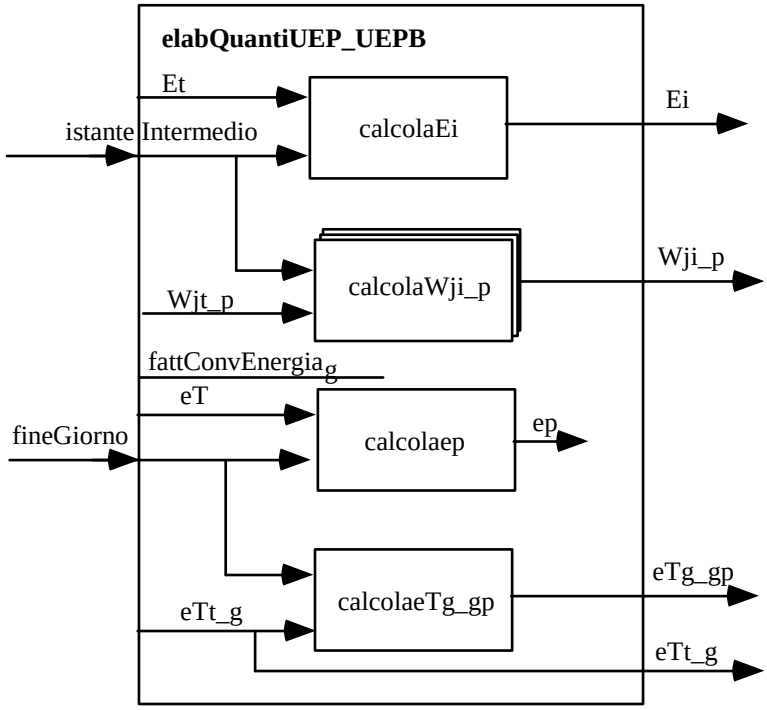


Figura 2.6.2.8 Rappresentazione grafica della classe *elabQuantiUEP_UEPB*.

class elabQuantiUEP_UEPB

inherit elabQuantiUE;

visible Ei, eTt_g, eTg_gp, Wji_p;

TD Items

vars Ei: natural;

eTt_g: natural;

eTg_gp: natural;

functions Wji_p(Tariffa): natural;

vars eT: natural;

ep: natural;

TI Items

consts UMEg;

fattConvEnergia$_g$: real;

2.6.2.3 Funzionalità comuni alle unità di elaborazione UEP e UEPB

La classe *elabQuantiUEP_UEPB*, definita come erede di *elabQuantiUE*, specifica le funzionalità di elaborazione dei quanti di energia comuni alla UEP e UEPB, ma non alla UEPM. La rappresentazione grafica, riportata in figura 2.6.2.8, riporta solo gli elementi aggiuntivi rispetto alla classe antenato. All'interfaccia di questa aggiunge quattro valori che misurano energia o potenza, secondo quanto precisato nel seguito.

Ei denota il valore complessivo dell'energia conteggiata dall'UE a partire dall'istante iniziale fino a un eventuale istante intermedio del periodo di fatturazione in corso.

eTt_g rappresenta l'energia consumata dall'inizio del giorno corrente fino all'istante attuale.

eTg_gp è l'energia totale assorbita nel giorno precedente a quello attuale.

Wji_p indica i valori massimi della potenza attiva media in un periodo sincrono di Δwj minuti, assorbita dall'utente in ciascuna tariffa *Tj* alla fine di un eventuale periodo intermedio.

eT indica il consumo totale di energia fino all'istante attuale, in unità di misura giornaliera per l'energia UME_g.

ep denota il consumo totale in unità di misura giornaliera alla fine del giorno precedente.

UMEg unità di misura dell’energia consumata giornalmente.

$fattConvEnergia_g$ è il fattore moltiplicativo che permette di ottenere il valore di energia, consumata giornalmente, corrispondente a un numero dato di quanti.

modules calcolaEi: fissaValore;
connections { (calcolaEi.valoreFissato, Ei),
(Et, calcolaEi.valore),
(istanteIntermedio, calcolaEi.flash) }

modules calcolaWji_p: **array** [T1 .. T4] **of** fissaValore;
connections { (istanteIntermedio, calcolaWji_p.flash),
(wjt_p, calcolaWji_p.valore),
(calcolaWji_p.valoreFissato, wji_p} };

modules calcolaep: fissaValore;
connections { (fineGiorno, calcolaep.flash),
(eT, calcolaep.valore),
(calcolaep.valoreFissato, ep) };

modules calcolaeTg_gp: fissaValore;
connections { (calcolaeTg_gp.valoreFissato, eTg_gp),
(eTt_g, calcolaeTg_gp.valore),
(fineGiorno, calcolaeTg_gp.flash) };

axioms eT = nqti * $fattConvEnergia_g$

eTt_g = eT - ep

$fattConvEnergia_g$ = valoreQuanto / UMEg

UMEg = 10;

end elabQuantiUEP_UEPB.

Il calcolo della grandezza *Ei* viene specificato sinteticamente utilizzando un modulo di classe *fissaValore* che attribuisce a *Ei* il valore di *Et* al momento in cui si verifica l'evento *istanteIntermedio*.

Il valore dei vari *Wji_p*(*T*) viene specificato sinteticamente utilizzando un array di moduli di classe *fissaValore* (un modulo per ogni tariffa). Esso prescrive che *Wji_p*(*T*) sia uguale al valore di *Wjt_p*(*T*) fissato all'istante intermedio.

Il valore dell'energia *ep* viene specificato sinteticamente utilizzando un modulo di classe *fissaValore*. *ep* è uguale al valore di *eT* fissato all'istante di fine del giorno precedente.

Il valore dell'energia *eTg_gp* viene specificato sinteticamente utilizzando un modulo di classe *fissaValore*. *eTg_gp* è uguale al valore di *eTt_g* fissato all'istante di fine del giorno precedente.

L'energia *eT* è proporzionale al numero dei quanti consumati, moltiplicati per il fattore di conversione nell'unità di misura giornaliera.

L'energia consumata a partire dalla fine del giorno precedente è uguale alla differenza tra quella consumata fino all'istante attuale e quella consumata fino alla fine del giorno precedente.

Il fattore di conversione dal numero di quanti all'energia (giornaliera) è dato dal rapporto tra il valore del quanto e l'unità di misura (giornaliera).

L'energia consumata giornalmente si misura in 10 Wh; ciò vale sia per l'UEP sia per l'UEPB.

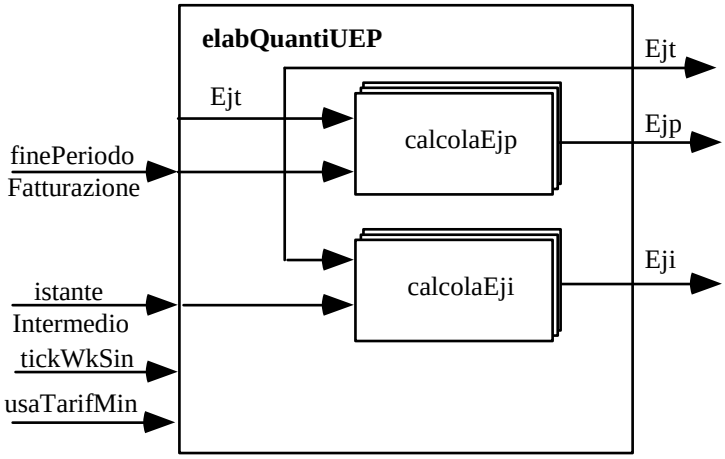


Figura 2.6.2.9 Rappresentazione grafica della classe *elabQuantiUEP*.

class elabQuantiUEP

inherit elabQuantiUEP_UEPB

visible Ejt, Ejp, Eji,
tickWkSin, usaTarifMinx

TD Items

functions

Ejt(Tariffa): natural;

Ejp(Tariffa): natural;

Eji(Tariffa): natural;

event Items tickWkSin;

TI Items

consts maxValEjt;

modules calcolaEjp, calcolaEji: **array** [T1 .. T4] **of** fissaValore;
connections { (calcolaEjp.valoreFissato, Ejp),
(Ejt, calcolaEjp.valore),
(finePeriodoFatturazione, calcolaEjp.flash),
(calcolaEji.valoreFissato, Eji),
(Ejt, calcolaEji.valore),
(istanteIntermedio, calcolaEji.flash) }

axioms $(T \neq T3) \rightarrow \left(\text{sommaQuantij}[T].\text{impulso} \leftrightarrow \left(\begin{array}{c} \text{quanto} \wedge \\ \text{baseTempiInizalizzata} \wedge \\ \text{tariffaCorrente} = T \end{array} \right) \right)$

$$\text{sommaQuantij}[T3].\text{impulso} \leftrightarrow \left(\begin{array}{c} \text{quanto} \wedge \\ \left(\begin{array}{c} \neg\text{baseTempiInizalizzata} \wedge \text{usaTarifMin} \\ \vee \\ \text{baseTempiInizializzata} \wedge \text{tariffaCorrente} = T3 \end{array} \right) \end{array} \right)$$

¬sommaQuantij[T}.salto(n)

2.6.2.4 Funzionalità peculiari dell'unità di elaborazione UEP

La classe *elabQuantiUEP*, rappresentata graficamente in figura 2.6.2.9, è erede della classe *elabQuantiUEP_UEPB*, e occupa quindi una posizione terminale nella gerarchia delle classi mostrata in figura 2.6.2.1. Essa specifica le funzionalità possedute dalla UEP e soltanto da essa. Tutte le funzionalità in comune con UEPB e con UEPM vengono ereditate dalle due classi antenato *elabQuantiUEP_UEPB* e *elabQuantiUE*.

Ejt(*Tariffa*) denota i valori dell'energia attiva assorbita in ciascuna tariffa a partire dall'istante di installazione del contatore, fino all'istante corrente.

Ejp(*Tariffa*) rappresenta i valori dell'energia attiva assorbita in ciascuna tariffa a partire dall'istante iniziale, fino alla fine del periodo di fatturazione precedente a quello in corso.

Eji(*Tariffa*) indica i valori dell'energia attiva assorbita in ciascuna tariffa a partire dall'istante iniziale fino ad un istante intermedio.

tickWkSin denota l'istante di calcolo della potenza media *wk*, effettuato in modo sincrono e specificato dalla classe di gestione della base dei tempi.

maxValEjt rappresenta il valore massimo ammissibile per i vari *Ejt*(*T*), che si assume non possano crescere indefinitamente nel tempo.

Il calcolo di *Ejp* e di *Eji* viene specificato sinteticamente mediante due array di moduli di classe *fissaValore* (l'array contiene un modulo per ogni tariffa). Per ogni tariffa *T*, *Ejp*(*T*) e *Eji*(*T*) sono uguali al valore dell'energia *Ejt*(*T*) all'istante di fine del periodo di fatturazione e, rispettivamente, all'istante intermedio.

La tariffa per il conteggio dei quanti è determinata da *tariffaCorrente* se *baseTempiInizializzata*, altrimenti è uguale alla tariffa minima *T3*.

Nell’unità UEP i quanti di energia sono sempre conteggiati uno alla volta (non accade mai che più quanti vengano conteggiati simultaneamente per una tariffa, perchè, al contrario che nell’UEPB e nell’UEPM, durante le interruzioni si applica senz’altro la tariffa più economica T3.

valoreQuanto = 50

UME = 1000

UMP = 100

maxValEt = 999.999;
maxValEjt = 999.000;

Δwk = 15';

Δwj = 15';

Ejt(T) = (nqtij(T) * fattConvEnergia) MOD (maxValEjt + 1)

tickWk = tickWkSin

end elabQuantiUEP.

Per l'UEP il valore del quanto di energia è di 50 Wh.

L'energia si misura in kWh.

La potenza si misura in decimi di kW, cioè in 100 W.

(per l'energia misurata su basse giornaliera, UMEg = 10 Wh è specificato nella classe *elabQuantiUEP_UEPB*, e quindi viene ereditato da *elabQuantiUEP*.

Si noti che in base ai valori precedenti, cioè *UMEg* = 10Wh e *valoreQuanto* = 50 Wh, risulta che $\text{fattConvEnergia}_g = 5$, cioè la precisione con cui si misurano i valori *eTtg* e *eTg_gp* è eccessiva rispetto al valore del quanto: i valori di questi conteggi saranno infatti sempre multipli di 5.

maxValEt e *maxValEjt* indicano i valori limite per le energie *Et* e *Ejt*, raggiunti i quali avviene l'overflow.

Per la UEP la potenza *wk* si calcola su intervalli di 15 minuti.

Per la UEP le potenze *Wjt_p*, *Wip_a* e *Wji_p* si calcolano su intervalli di 15 minuti.

Le potenze *Ejt*(*T*) crescono in modo proporzionale ai conteggi con *nqtij*(*T*) salvo overflow e ricicli.

Il calcolo di *wk* avviene sempre rispetto a un intervallo sincrono.

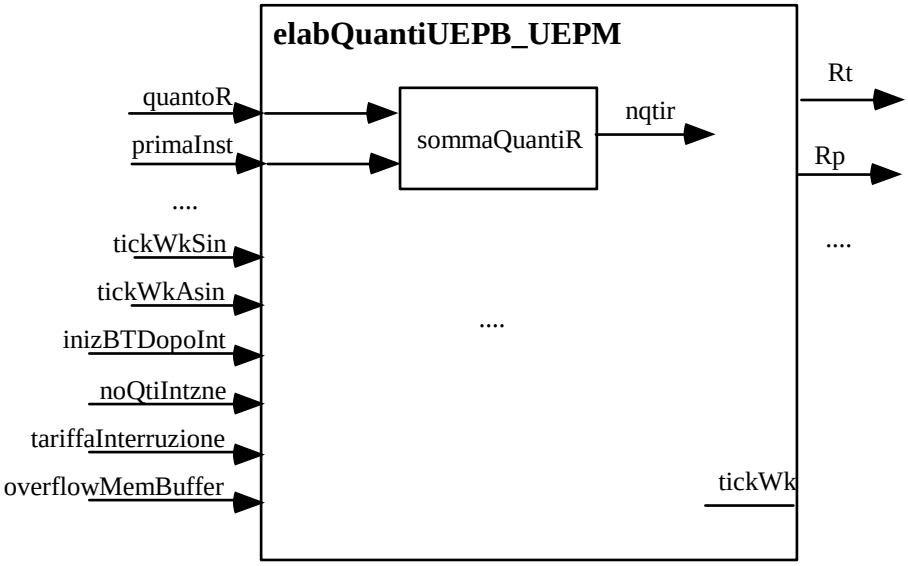


Figura 2.6.2.10 Rappresentazione grafica della classe *elabQuantiUEPB_UEPM*.

class elabQuantiUEPB_UEPM

inherit elabQuantiUE

visible ... quantoR, tickWkSin, tickWkAsin, ..., Rt, Rp, ...

..... omissis .....

event Items quantoR;

tickWkSin, tickWkAsin;

inizBTDopoInt;

overflowMemBuffer(natural);

TD Items

vars tariffaInterruzione: Tariffa;

noQtiIntzne: natural;

..... omissis .....

2.6.2.5 Funzionalità comuni alle unità di elaborazione UEPB e UEPM

La classe *elabQuantiUEPB_UEPM* è erede della classe *elabQuantiUE*, occupa quindi una posizione intermedia nella gerarchia delle classi in figura 2.6.2.1, e specifica tutte le funzionalità comuni a UEPB e UEPM, e assenti nell'UEP. La rappresentazione grafica in figura 2.6.2.10, così come quella testuale nelle prossime pagine, riporta al solito solo gli elementi aggiuntivi rispetto alla classe antenato.

La classe contiene la dichiarazione di *Rt*, *Rp*, *ejt_p*(*Tariffa*), *rjt_p*(*Tariffa*), *ejp_a*(*Tariffa*), *rjp_a*(*Tariffa*) .... non riportate qui per brevità.

quantoR denota l'acquisizione di un quanto di energia reattiva.

tickWkSin e *tickWkAsin* sono gli eventi che indicano l'istante in cui calcolare la potenza media *wk*, nei due casi (mutuamente esclusivi) in cui il periodo di calcolo sia sincrono oppure asincrono rispetto alla base dei tempi. Essi vengono rispettivamente dalle classi *gestioneBaseTempi* e *gestioneInterruzioni*.

L'evento *inizBTDopoInt* indica che la base dei tempi viene di nuovo inizializzata dopo un'interruzione.

L'evento *overflowMemBuffer* indica l'avvenuto overflow nel conteggio quanti energia attiva durante il periodo di assenza di base dei tempi a seguito di un'interruzione; l'argomento dell'evento indica il numero dei quanti conteggiati, che vanno attribuiti alla tariffa minima. A questo scopo l'evento *overflowMemBuffer* viene connesso al modulo che specifica l'elaborazione dei quanti.

tariffaInterruzione rappresenta la tariffa da applicare ai consumi di energia effettuati durante il periodo di assenza di base dei tempi a seguito di un'interruzione. Viene determinata dalla funzione di gestione delle interruzioni.

noQtiIntzne denota il numero di quanti consumati durante il periodo di assenza di base dei tempi a

seguito di un’interruzione. Viene determinata dalla funzione di gestione delle interruzioni.

La classe *elabQuantiUEPB_UEPM* specifica gli item *Rt*, *Rp*, *ejt_p*(*Tariffa*), *rjt_p*(*Tariffa*), *ejp_a*(*Tariffa*), *rjp_a*(*Tariffa*) in modo del tutto analogo a quanto fatto in *elabQuantiUE* e in *elabQuantiUEP_UEPB* per item analoghi.

axioms

$$(\text{baseTempiInizializzata} \rightarrow \text{tickWk} = \text{tickWkSin}) \wedge (\neg\text{baseTempiInizializzata} \rightarrow \text{tickWk} = \text{tickWkAsin})$$

sommaQuanti[T].impulso $\times$ quanto baseTempiInizializzata tariffaCorrente = T

$$\text{sommaQuanti}[T].\text{salto}(n) \leftrightarrow \left(\begin{array}{l}(\text{inizBTDopoInt} \wedge n = \text{noQtiIntzne} \wedge T = \text{tariffaInterruzione}) \vee \\ (\text{overflowMemBuffer}(n) \wedge T = T3)\end{array}\right)$$

end elabQuantiUEPB_UEPM.

Se la base tempi è presente *wk* viene calcolata su un periodo di tempo sincrono; in assenza di base tempi *wk* viene calcolata su un periodo di tempo asincrono.

In condizioni di funzionamento normale, in presenza cioè della base dei tempi, il conteggio dei quanti di una determinata tariffa viene incrementato di un'unità se e solo se all'istante corrente viene consumato un quanto di energia e la tariffa è quella valida correntemente.

Il conteggio dei quanti per una tariffa viene aumentato di una quantità *n* nei seguenti due casi, e solo in questi.

i. Quando ritorna la base dei tempi a seguito di un'interruzione, e allora l'aumento è uguale al numero dei quanti consumati durante il periodo di assenza della base dei tempi immediatamente successivo al ritorno dell'alimentazione, per la tariffa minima valida in quell'intervallo, come determinata dalla funzione di gestione delle interruzioni tramite il valore di *tariffaInterruzione*.

ii. Durante un periodo particolarmente prolungato di assenza della base dei tempi, quando il conteggio dei quanti consumati supera la capacità del registro di conteggio; allora il conteggio viene ripreso da zero e il numero dei quanti conteggiati finora, argomento dell'evento *overfloMemBuffer*, viene attribuito alla tariffa più economica *T3*.

2.6.2.6 Funzionalità peculiari dell'unità di elaborazione UEPB

La classe *elabQuantiUEPB* specifica le funzionalità della UEPB e assente negli altri due tipi. Essa occupa perciò una posizione terminale nella gerarchia delle classi di figura 2.6.2.1.. La specifica di *elabQuantiUEPB* viene omessa in quanto non presenta alcun elemento di novità rispetto a quanto già presentato nei paragrafi precedenti.

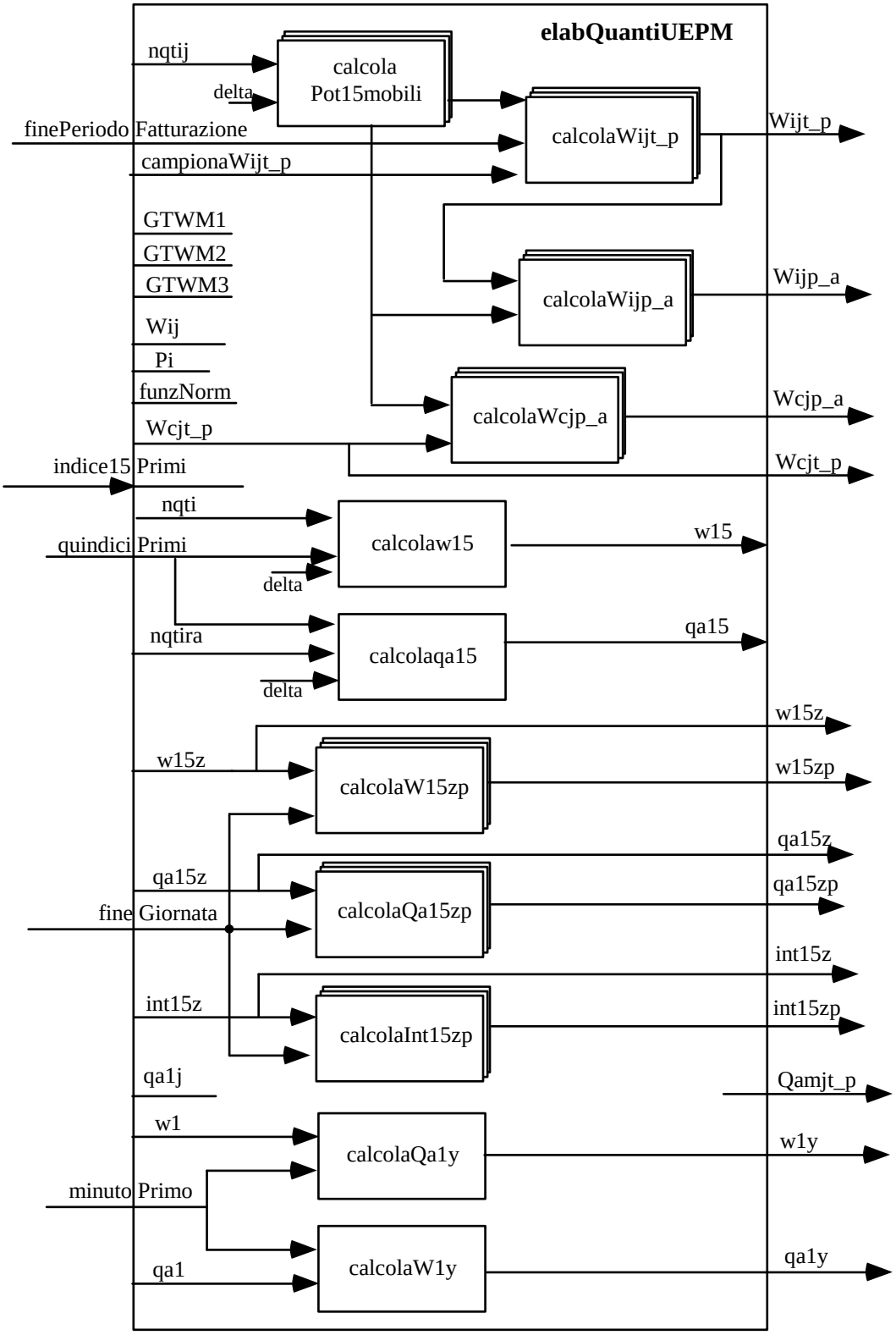


Figura 2.6.2.11 Rappresentazione grafica della classe *elabQuantiUEPM*.

class elabQuantiUEPM
inherit elabQuantiUEPB_UEPM [**rename** rjt_p **as** rajt_p, rjp_a **as** rajp_a];
visible .... omissis ....
TD Items .... omissis ....
axiom .... omissis ....

end elabQuantiUEPM.

2.6.2.7 Funzionalità peculiari dell'unità di elaborazione UEPM

La classe *elabQuantiUEPM*, rappresentata graficamente in figura 2.6.2.11 specifica tutte le entità presenti nell'UEPM e assenti negli altri tipi di UE. Si trova perciò in posizione terminale nella gerarchia di ereditarietà di figura 2.6.2.1.

La classe *elabQuantiUEPM*, erede della *elabQuantiUEPB_UEPM*, aggiunge a essa numerose grandezze calcolate e memorizzate in registri non volatili. Tra queste, un certo numero (precisamente *Ret*, *Rep*, *ejg_p*(*T*), *rajg_p*(*T*), *rejt_p*(*T*), *rejp_a*(*T*), *Qamjp_a*(*T*) dichiarati e specificati nel modo usuale ....) ha natura e caratteristiche del tutto simili ad altre grandezze presenti nelle altre classi, e la loro specifica può utilizzare moduli delle classi componenti base (come *sommatore*, *fissaValore*, *calcolaGradiente*, etc.).

Altre grandezze (precisamente *Wcjt_p*, *w15z*, *qa15z*, *Int15z*, *w15zp*, *qa15zp*, *Int15zp*, *Qamjt_p*) hanno natura sostanzialmente diversa da quelle calcolate negli altri tipi di unità di elaborazione, e quindi la loro specifica richiede l'introduzione di un certo numero di assiomi *ad hoc*, oltre all'usuale utilizzo di moduli delle classi componenti base, come tratteggiato nella rappresentazione grafica della classe *elabQuantiUEPM*, in cui si evidenzia un insieme di moduli e di connessioni tra i loro item esportati.

... omissis ....

... omissis ....

... omissis ....

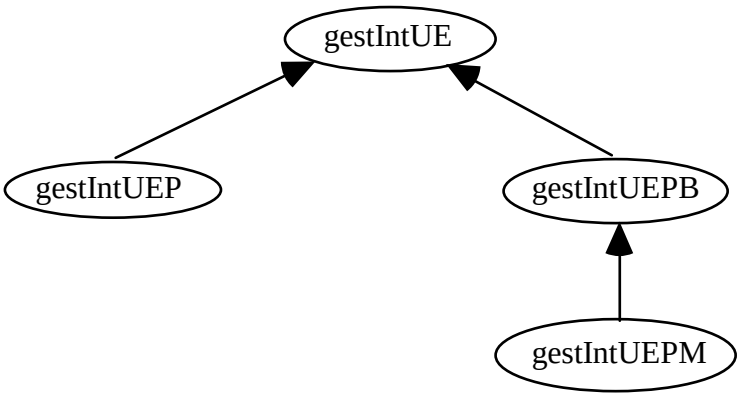


Figura 2.6.3.1 Gerarchia di classi di specifica della funzione di gestione delle interruzioni.

2.6.3 Gestione delle Interruzioni

In modo analogo a quanto fatto nel paragrafo 2.6.2 per la funzione di elaborazione dei quanti di energia, anche per la funzione di gestione delle interruzioni strutturiamo la specifica in una gerarchia ereditaria di classi, che evidenzia le funzionalità comuni ai diversi tipi di unità di elaborazione o peculiari di alcune di esse.

Tuttavia, come mostrato in figura 2.6.3.1, la gerarchia è qui più semplice, in quanto non ci sono funzionalità comuni a UEP e UEPB che non siano comuni anche a UEPM (non esiste quindi ragione di introdurre una classe *gestIntUEP_UEPB* analoga a *elabQuantiUEP_UEPB*); inoltre le funzionalità di gestione delle interruzioni per UEPM sono semplicemente un superinsieme di quelle per UEPB.

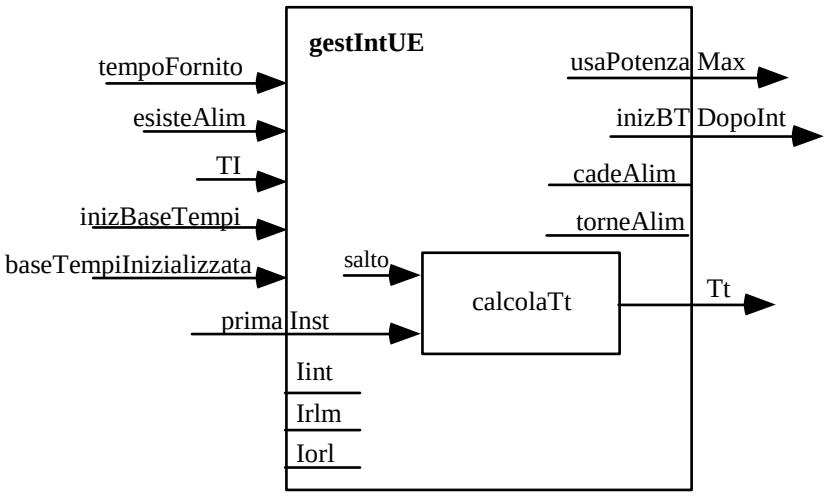


Figura 2.6.3.2 Rappresentazione grafica della classe *gestIntiUE* che modella le funzionalità comuni a tutte le UE.

class gestIntUe

visible primaInst, inizBaseTempi, esisteAlim, TI, tempoFornito,
usaPotenzaMax, inizBTDopoInt, Tt

event Items cadeAlim;

tornaAlim;

inizBaseTempi;

inizBTDopoInt;

state Items usaPotenzaMax;

baseTempiInizializzata;

TD Items

vars

tempoFornito: tempoStrutturato;

Iint,

Irlm,

Iorl: tempoStrutturato;

Tt: natural;

2.6.3.1 Funzionalità comuni a tutti i tipi di unità di elaborazione

La classe *gestIntUE*, la cui rappresentazione grafica è riportata in figura 2.6.3.2, occupa la posizione più alta nella gerarchia di ereditarietà di figura 2.6.3.1, in quanto raggruppa le funzionalità comuni a tutti i tipi di unità di elaborazione.

In estrema sintesi, tali funzionalità comprendono:

- la valutazione della durata di ogni interruzione e il calcolo della durata complessiva di tutte le interruzioni;
- l'indicazione di utilizzare la potenza massima per lo stacco dell'interruttore magnetotermico durante il periodo tra il ritorno dell'alimentazione, quando la base dei tempi è stata persa, e la reinizializzazione della base dei tempi.

Anche nelle classi per la specifica della gestione delle interruzioni verrà utilizzata la classe ausiliaria *sommatore* per la specifica dei conteggi cumulativi, introdotta nel paragrafo 2.6.2.1.

L'evento *cadeAlim* corrisponde all'istante in cui cade l'alimentazione (inizio di un'interruzione).

L'evento *tornaAlim* corrisponde all'istante in cui torna l'alimentazione (fine di un'interruzione).

L'evento *inizBaseTempi* corrisponde all'istante in cui viene inizializzata la base dei tempi; *inizBaseTempi* viene determinato dalla funzione di gestione della base dei tempi.

L'evento *inizBTDopoInt* corrisponde all'istante in cui la base dei tempi viene di nuovo inizializzata dopo un'interruzione.

Lo stato *usaPotenzaMax* indica di usare la potenza massima per l'apertura dell"interruttore magnetotermico.

Lo stato *baseTempiInizializzata* indica che la base dei tempi è presente.

tempoFornito indica l'ora corrente, come determinata dalla funzione di gestione della base dei tempi.

Iint denota l'istante di inizio di un'interruzione.

Irlm denota l'istante di ritorno dell'alimentazione a seguito di un'interruzione (con eventuale perdita della base dei tempi).

Iorl denota l’istante di reinizializzazione della base dei tempi a seguito di un’interruzione in cui la base dei tempi è stata persa.

Tt indica il conteggio cumulativo della lunghezza dei periodi di assenza di alimentazione.

modules calcolaTt: sommatore;

connections { (calcolaTt.reset, primaInst)),
(Tt, calcolaTt.somma) }

axioms ¬calcolaTt.impulso

$$\text{inizBTDopoInt} \leftrightarrow \left(\text{inizBaseTempi} \wedge \text{Since}_{ee}\left(\neg\text{inizBaseTempi}, \left(\begin{array}{l} \text{tornaA lim} \\ \wedge \\ \neg\text{baseTempiInizializzata} \end{array} \right) \right) \right)$$

usaPotenzaMax × Since(¬inizBaseTempi, tornaAlim ¬baseTempiInizializzata)

var t: tempoStrutturato;

(cadeAlim tempoFornito = t) ∅ Until(Iint = t, cadeAlim)

cadeAlim × UpToNow(esisteAlim) NowOn(¬esisteAlim)
tornaAlim × UpToNow(¬esisteAlim) NowOn(esisteAlim)

tornaAlim baseTempiInizializzata ∅ calcolaTt.salto(tempoFornito - Iint)

$$\text{inizBTDopoInt} \rightarrow \left(\begin{array}{l} \text{Iorl} = \text{tempoFornito} \\ \wedge \\ \text{Irlm} = \text{tempoFornito} - \text{TI} \\ \wedge \\ \text{calcolaTt.salto}(\text{tempoFornito} - \text{Iint} - \text{TI}) \end{array} \right)$$

$$\text{calcolaTt.salto}(\text{n}) \leftrightarrow \left(\begin{array}{l} \text{tornaA lim} \wedge \text{baseTempiInizializzata} \wedge (\text{n} = \text{tempoFornito} - \text{Iint}) \\ \vee \\ \text{inizBTDopoInt} \wedge (\text{n} = \text{tempoFornito} - \text{Iint} - \text{TI}) \end{array} \right)$$

end gestIntUE.

Il conteggio del tempo totale di interruzione viene specificato sinteticamente mediante un modulo di tipo *sommatore*. Il conteggio viene azzerato all'installazione del contatore.

Il conteggio di *Tt* avviene per salti, non per incrementi unitari.

Si noti che non è previsto il caso di overflow del registro che memorizza *Tt*, anche se ciò è in realtà possibile dopo un periodo di funzionamento del contatore sufficientemente lungo.

L'evento *inizBTDopoInt* si verifica esattamente quando la base dei tempi viene inizializzata dopo che era stata persa a seguito di un'interruzione.

Si usa la potenza massima per staccare l'interruttore magnetotermico quando, dopo un'interruzione di alimentazione, la base dei tempi è stata persa e non è ancora tornata.

Quando cade l'alimentazione si memorizza l'istante corrente in *Iint*.

Gli eventi *cadeAlim* e *tornaAlim* sono definiti dal passaggio dalla presenza all'assenza (e rispettivamente dall'assenza alla presenza) dell'alimentazione.

Quando torna l'alimentazione e la base dei tempi è ancora inizializzata (ciò può accadere in UEPB e UEPM, che hanno l'alimentazione ausiliaria) bisogna solo calcolare la durata dell'interruzione e aggiungerla a *Tt*.

Se quando torna l'alimentazione la base dei tempi non è inizializzata, solo al momento della reinizializzazione si calcola la durata dell'interruzione e la si aggiunge a *Tt*.

(NB si sta assumendo che TI parta da zero quando torna l'alimentazione dopo una perdita della base dei tempi)

I precedenti assiomi forniscono delle condizioni sufficienti per l'aggiornamento del valore di *Tt*. Nel seguito si danno le condizioni necessarie.

Tt viene aggiornato solo quando torna l'alimentazione e la base tempi è ancora inizializzata, oppure quando la base dei tempi viene inizializzata dopo un'interruzione in cui era stata persa.

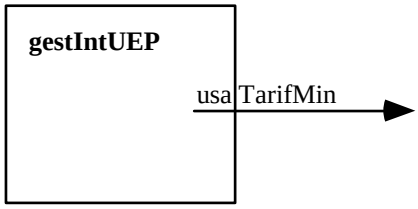


Figura 2.6.3.3 Rappresentazione grafica della classe *gestIntUEP*.

class gestIntUEP

inherit gestIntUE

visible usaTarifMin

state Items usaTarifMin;

axioms usaTarifMin × Since(–inizBaseTempi, tornaAlim ¬baseTempiInizializzata)

end gestIntUEP.

2.6.3.2 Funzionalità peculiari dell'unità di elaborazione UEP

La classe *gestIntUEP*, la cui rappresentazione grafica è riportata in figura 2.6.3.3, specifica le funzionalità caratteristiche della sola UEP, e occupa perciò la posizione terminale nella gerarchia di classe di figura 2.6.3.1.

Essa specifica che nel periodo di assenza di base dei tempi che segue a un'interruzione (la UEP perde sempre la base dei tempi in caso interruzione, essendo sprovvista di alimentazione ausiliaria) l'energia consumata viene sempre conteggiata alla tariffa minima.

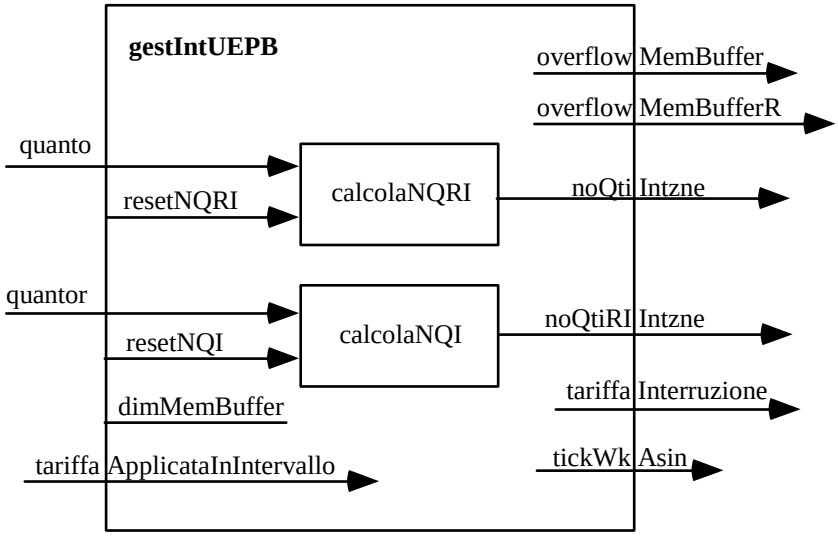


Figura 2.6.3.4 Rappresentazione grafica della classe *gestIntUEPB*.

class gestIntUEPB

inherit gestIntUE

visible quanto, quantor, tariffaApplicataInIntervallo,
overflowMemBuffer, overflowMemBufferR, noQtiIntzne, noQtiRIntzne,
tariffaInterruzione

event Items overflowMemBuffer(natural);
overflowMemBufferR(natural);

quanto;
quantor;
resetNQI, resetNQRI;

tickWkAsin;

TD Items

vars tariffaInterruzione: Tariffa;

noQtiIntzne, noQtiRIntzne: natural;

TI Items

consts dimMemBuffer;

TI Items

predicates tariffeApplicateInIntervallo(tempoStrutturato, tempoStrutturato, Tariffa);

2.6.3.3 Funzionalità dell'unità di elaborazione UEPB

La classe *gestIntUEPB,* rappresentata graficamente in figura 2.6.3.4, specifica le funzionalità di gestione delle interruzioni per la classe UEPB, che sono un superinsieme di quelle comune a tutte le unità di elaborazione e un sottoinsieme di quelle di UEPM. Essa occupa perciò una posizione intermedia nella gerarchia di ereditarietà di figura 2.6.3.1, essendo erede della classe *gestIntUE,* e antenato della classe *gestIntUEPM*.

In estrema sintesi, essa specifica il comportamento della UE durante il periodo che segue il ritorno dell'alimentazione, nel caso di perdita della base dei tempi (che non si verifica sempre, essendo le UEPB e UEPM fornite di alimentazione ausiliaria), fino alla successiva risincronizzazione. Durante questo periodo la UE deve, oltre quanto già specificato in *gestIntUE*,

- conteggiare il numero dei quanti di energia attiva e reattiva consumati, senza sapere ancora a quale tariffa essi verranno contabilizzati, e quindi memorizzando tali conteggi in registri ausiliari;
- per tali registri ausiliari deve essere gestito l'eventuale overflow, improbabile ma possibile in caso di eccezionale ritardo della risincronizzazione;
- durante il periodo di assenza della base dei tempi, deve generare, a intervalli di tempo regolari, sincronizzati all'istante di ritorno dell'alimentazione, il segnale che causa il calcolo della potenza media *wk*.
- alla risincronizzazione deve infine stabilire a quale tariffa devono essere attribuiti i quanti di energia consumati durante il periodo di assenza della base dei tempi.

L'evento *overflowMemBuffer* denota un overflow nel conteggio quanti di energia attiva consumati durante il periodo di assenza della base dei tempi a seguito di un'interruzione. Similmente, *overflowMemBufferR* denota un overflow nel conteggio quanti di energia reattiva consumati durante il periodo di assenza della base dei tempi a seguito di un'interruzione.

Gli eventi *quanto* e *quantor* denotano l'avvenuto consumo di un quanto di energia attiva o, rispettivamente, reattiva.

Gli eventi *resetNQI* e *resetNQRI* denotano che il valore della memoria buffer, per l'energia attiva o reattiva, viene posto a zero.

l'evento *tickWkAsin* denota il segnale per il calcolo della potenza media *wk* su intervalli asincroni, come prescritto per i periodi di assenza di alimentazione della base dei tempi.

tariffaInterruzione denota la tariffa da applicare nel periodo tra il ritorno dell'alimentazione e il ritorno della base dei tempi, quando questa è stata persa a seguito di un'interruzione.

noQtiIntzne e *noQtiRIntzne* denotano il numero dei quanti di energia attiva e reattiva consumati dopo un'interruzione, in caso di perdita della base dei tempi, dal ritorno dell'alimentazione fino alla reinizializzazione della base dei tempi.

Il parametro costante*dimMemBuffer* indica il valore massimo del registro di memoria per *noQtiIntzne* e *noQtiRIntzne*; raggiunto tale valore avviene l'overflow del registro.

tariffeApplicateInIntervallo è un predicato che, date due ore che definiscono un intervallo (*Irlm* e *Iorl* nel caso di suo uso in gestioneInterruzioni) dà l'insieme di tutte le tariffe valide in almeno una parte di tale intervallo.

modules calcolaNQI, calcolaNQRI: sommatore; /* sommatori per noQtiIntzne e noQtiRIntzne */

connections { (resetNQI, calcolaNQI.reset),
(resetNQRI, calcolaNQRI.reset),
(quanto, calcolaNQI.impulso),
(quantor, calcolaNQRI.impulso),
(calcolaNQI.somma, noQtiIntzne),
(calcolaNQRI.somma, noQtiRIntzne) }

axioms $\text{resetNQI} \leftrightarrow \left(\begin{array}{l} \text{primaInst} \vee \\ \text{inizBTDopoInt} \vee \\ (\text{calcolaNQI.somma} = \dim \text{MemBuffer} \wedge \text{quanto}) \end{array} \right)$

$\text{resetNQRI} \leftrightarrow \left(\begin{array}{l} \text{primaInst} \vee \\ \text{inizBTDopoInt} \vee \\ (\text{calcolaNQRI.somma} = \dim \text{MemBuffer} \wedge \text{quantor}) \end{array} \right)$

$\neg$calcolaNQI.salto(n) $\neg$calcolaNQRI.salto(n)

$\text{overflowMemBuffer}(n) \leftrightarrow \left(\begin{array}{l} n = \dim \text{MemBuffer} \\ \wedge \\ \text{UpToNow}(\text{noQtiInterzne} = \dim \text{MemBuffer}) \\ \wedge \\ \text{NowOn}(\text{noQtiInterzne} = 0) \end{array} \right)$

$\text{overflowMemBufferR}(n) \leftrightarrow \left(\begin{array}{l} n = \dim \text{MemBuffer} \\ \wedge \\ \text{UpToNow}(\text{noQtiRInterzne} = \dim \text{MemBuffer}) \\ \wedge \\ \text{NowOn}(\text{noQtiRInterzne} = 0) \end{array} \right)$

$\text{tickWkA sin} \leftrightarrow \left(\begin{array}{l} \text{Since}(\neg \text{inizBaseTempi}, \text{tornaA lim} \wedge \neg \text{baseTempiInizializzata}) \\ \wedge \\ \exists n \ (n > 0 \wedge \text{LastTime}(\text{tornaA lim} \wedge \neg \text{baseTempiInizializzata}, n \cdot \Delta\text{wk})) \end{array} \right)$

inizBTDopoInt $\varnothing$
if tariffaApplicataInIntervallo(Irlm, Iorl, T3) **then** tariffaInterruzione = T3
elsif tariffaApplicataInIntervallo(Irlm, Iorl, T2) **then** tariffaInterruzione = T2
elsif tariffaApplicataInIntervallo(Irlm, Iorl, T1) **then** tariffaInterruzione = T1
else tariffaInterruzione = T4
fi

end gestIntUEPB.

NB se al ritorno dell'alimentazione dopo un'interruzione la base dei tempi non è stata persa, il conteggio dei quanti e la loro attribuzione alle varie tariffe riprende normalmente, e l'unità di elaborazione non deve fare nulla (oltre quanto già specificato in *gestIntUE*).

Il calcolo del numero dei quanti di energia attiva e reattiva, consumati al ritorno dell'alimentazione in assenza di base dei tempi, viene specificato nel modo usuale mediante due moduli di classe *sommatore*.

I conteggi di *noQtiIntzne* e *noQtiRIntzne* vengono azzerati:

- all'installazione del contatore,
- alla reinizializzazione della base dei tempi dopo un'interruzione in cui era stata persa,
- quando avviene un overflow.

Il conteggio di *noQtiIntzne* e *noQtiRIntzne* avviene per incrementi unitari e non per salti.

L'overflow nel conteggio di *noQtiIntzne* e di *noQtiRIntzne* corrisponde al passaggio dal valore massimo al valore nullo; nel contesto dell'overflow il valore massimo viene comunicato (tramite l'argomento degli eventi *overflowMemBuffer* e *overflowMemBufferR*) alla funzione di elaborazione dei quanti.

L'evento *tickWkAsin* ha luogo durante il periodo di assenza di base dei tempi dopo il ritorno dell'alimentazione a seguito di un'interruzione, e si verifica a intervalli di lunghezza Δwk a partire dal ritorno dell'alimentazione.

Alla reinizializzazione della base dei tempi, i quanti consumati vanno attribuiti alla più economica tra tutte le tariffe in vigore durante il periodo tra *Irlm* e *Iorl*, secondo l'ordine decrescente *T4-T1-T2-T3*.

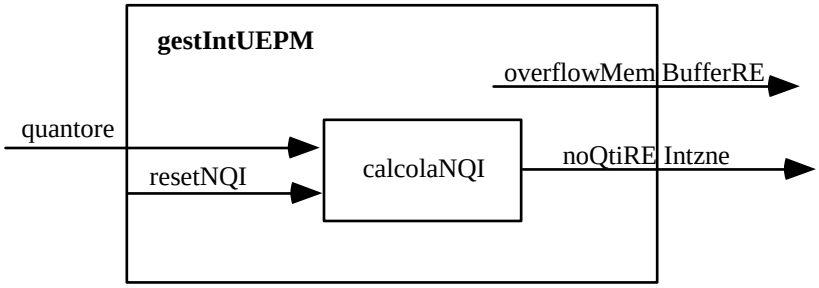


Figura 2.6.3.5 Rappresentazione grafica della classe *gestIntUEPM*.

class gestIntUEPM

inherit gestIntUEPB

visible quantore, overflowMemBufferRE, noQtiREIntzne

... omissis ....

.... omissis....

end gestIntUEPM.

2.6.3.4 Funzionalità peculiari dell'unità di elaborazione UEPM

La classe *gestIntUEPM*, rappresentata graficamente in figura 2.6.3.5, specifica le funzionalità di gestione delle interruzioni peculiari della classe UEPM, essa è perciò erede della classe *gestIntUEPB*, e occupa una posizione terminale nella gerarchia di ereditarietà di figura 2.6.3.1.

La gestione delle interruzioni nell'UEPM deve svolgere operazioni di conteggio del numero dei quanti di energia con memorizzazione in registri ausiliari e di gestione dell'overflow di tali registri, per l'energia reattiva erogata. Tali operazioni vengono svolte dall'UEPM in modo del tutto analogo alle azioni svolte dall'UEPB per l'energia attiva e reattiva assorbite.

La classe include dichiarazioni degli item relativi alla gestione dell'energia reattiva erogata: *overflowMemBufferRE*, *noQtiREIntzne*, *quantore*. Questi vengono omessi per brevità.

Anche gli assiomi che specificano il comportamento dei componenti della classe vengono omessi, essendo del tutto simili a quelli analoghi di *gestIntUEPB*, relativi all'energia attiva assorbita.

3. Alcuni commenti di carattere esplicativo e metodologico

3.1. Architettura fisica e architettura funzionale delle UE

Rispetto ai documenti originari si è ritenuto di separare la descrizione dell'architettura fisica delle UE da quella funzionale mediante due distinte definizioni della classe UE. Questa scelta non è dovuta al linguaggio TRIO, che avrebbe permesso di mantenere unite le descrizioni dei componenti fisici e dei compiti di UE in un'unica classe. Essa è piuttosto motivata da considerazioni di carattere metodologico. Se da un lato è evidente che entrambi i tipi di descrizione sono utili e necessari alla completa definizione della UE e alla sua realizzazione, è anche vero che essi hanno obiettivi diversi. Infatti, i documenti di specifica originari, dopo aver brevemente descritto la struttura fisica delle UE, si concentrano sulla definizione dei loro compiti prescindendo dall'eventuale loro attribuzione ai singoli componenti (a esempio: presentazione dei dati all'utente fornita dal Display).

In un quadro metodologico ideale si raccomanderebbe di separare ancor più nettamente le due diverse “visioni” della UE. In primo luogo dovrebbe essere fornita l'architettura funzionale e la sua descrizione di dettaglio secondo lo schema adottato nei documenti originari e sostanzialmente mantenuto in questo. Successivamente, l'architettura fisica dovrebbe essere presentata come un ulteriore livello di raffinamento delle specifiche. Procedendo nel dettaglio i singoli compiti esaminati in precedenza a livello globale potrebbero essere raffinati ed attribuiti a singoli componenti (ad esempio la gestione della base dei tempi dovrebbe essere verosimilmente attribuita all'Unità Centrale Microcalcolatore ed eventualmente ad un suo ulteriore sotto componente “Clock”). Si avrebbero quindi i Compiti dell'Unità Centrale Microcalcolatore, i compiti del Display, ..., ed eventualmente anche compiti ancora definiti a livello globale.

Un tale approccio aiuterebbe inoltre la gestione temporale delle specifiche: ad esempio potrebbe accadere che, pur rimanendo inalterate le funzioni dell'intera unità alcuni suoi componenti vengano rimpiazzati da altri forniti di tecnologie più moderne. In tal caso alcuni compiti potrebbero essere riallocati da un componente a un altro. Sarebbe in tal caso facile aggiornare con la massima precisione la specifica dell'unità mantenendo inalterata la parte relativa all'architettura funzionale, ma modificandone l'architettura fisica e riesaminando la definizione di dettaglio dell'attribuzione dei compiti ai singoli componenti (ottenendo anche il beneficio di una più facile riprogettazione e verifica di coerenza della nuova versione).

3.2. Il dominio temporale adottato e le sue proprietà

Per mantenere la specifica ad un elevato livello di astrazione e restare aderenti il più possibile alla lettera dei documenti originari si è adottato come dominio temporale l'insieme dei numeri reali (tempo continuo). Ovviamente diverso dominio hanno le *variabili usate per misurare il tempo* (TI, TempoCabina, ...). L'uso di un dominio temporale continuo permette di ipotizzare eventi assolutamente istantanei e conseguenze immediate a determinati eventi (ad esempio: l'arrivo di un segnale di cabina determina l'*allineamento immediato* del tempo interno). Esso ha anche permesso di assumere senza inconvenienti l'ipotesi di esclusione di contemporaneità tra alcuni eventi, che ha portato ad una formulazione semplificata di alcuni assiomi.

È ipotizzabile che ad un più basso livello di astrazione convenga usare un dominio temporale discreto (non necessariamente coincidente con TI), non escludere la contemporaneità di eventi e prevedere un certo ritardo tra eventi-causa ed eventi-effetto. Ciò comporterebbe però specifiche più dettagliate dei documenti originari.

3.3. Gestione dei disallineamenti temporali nelle diverse unità

La formalizzazione dei requisiti riguardanti la gestione dei disallineamenti temporali ha richiesto qualche lieve modifica rispetto ai contenuti dei documenti originari: le modifiche si sono rese necessarie per alcune incompletezze e lievi contraddizioni (a esempio le condizioni di riallineamento espresse nel documento DH025 sono determinate in base al tipo di messaggio proveniente dall'ACS e in base all'entità del disallineamento in modo che potrebbe però generare conflitti). Ciò è probabilmente dovuto in gran parte al fatto che le informazioni significative si trovavano in sezioni diverse del documento DH025: Sezione 5.1 e Sezione 5.7. Infatti, in Sezione 5.1, quinto paragrafo, si prescrive che l'aggiornamento della base dei tempi debba aver luogo nei due seguenti casi (NB: le sottolineature sono state aggiunte per maggiore evidenza)

Quando la base dei tempi dell'UEPB risulta inizializzata (informazioni di data e ora presenti), eventuali messaggi provenienti dall'ACS e contenenti le informazioni di anno, mese, giorno, ore e minuti debbono provocare l'aggiornamento della base tempi dell'UEPB:

- solo se i due riferimenti temporali (quello dell'UEPB e quello inviato dall'ACS) non differiscono tra loro per più di 5 minuti, in caso di messaggio broadcast da parte dell'ACS;
- in ogni caso, in caso di messaggio singolo da parte dell'ACS.

Invece in Sezione 5.7 (sottosezione D sui parametri temporali di sincronizzazione) si specifica che

> Se, al contrario, la base dei tempi dell'UEPB risulta già inizializzata, la sua reinizializzazione può avvenire solo tramite messaggi singoli inviati dall'ACS (Vedi paragrafo 5.1).

Evidentemente la prescrizione del paragrafo 5.7 contraddice quella del paragrafo 5.1, primo punto, là dove impone che l'inizializzazione avvenga solo a fronte di un messaggio singolo, mentre in Sezione 5.1 si ammette anche il caso di messaggio broadcast, con differenza tra i riferimento temporali di non più di 5 minuti.

La formalizzazione qui proposta è stata ricavata nello spirito di modificare al minimo struttura e contenuti dei documenti originari; è però probabile che una riflessione più approfondita e completa suggerisca una revisione un po' più incisiva dei medesimi e porti ad una formulazione meglio strutturata (con minori ridondanze e quindi minori rischi di incongruenze).

3.4 Strutturazione delle specifiche in gerarchie di ereditarietà

Come evidenziato all’inizio dei paragrafi 2.6.1, 2.6.2 e 2.6.3, la specifica delle funzioni di gestione della base dei tempi, di elaborazione dei quanti di energia e di gestione delle interruzioni, che possiedono importanti parti comuni ma pure differiscono in misura significativa tra i vari tipi di unità di elaborazione, è stata organizzata mediante una gerarchia di classi legate dalla relazione di ereditarietà.

Tale organizzazione gerarchica permette di fattorizzare le parti comuni della funzione per i diversi tipi di unità di elaborazione. Ciò a sua volta favorisce la redazione di specifiche più compatte (vengono evitate tutte le inutili duplicazioni), modificabili (le eventuali modifiche a funzionalità comuni a più tipi di unità di elaborazione devono essere formalizzate una sola volta, per la sola classe che contiene tali funzionalità) e consistenti (l’assenza di duplicazioni evita di creare inconsistenze tra copie multiple degli stessi requisiti, che si generano specialmente in fase di modifica e aggiornamento della documentazione, in presenza di ridondanze).

L'organizzazione gerarchica, accoppiata a un'adeguata rappresentazione grafica e a un uso sistematico delle convenzioni di nomenclatura degli elementi, permette inoltre di ottenere un efficace "colpo d'occhio" sull'organizzazione della famiglia dei contatori per le diverse categorie, evidenziando i criteri adottati per la ripartizione della caratteristiche della funzione tra i diversi tipi di unità di elaborazione.

A esempio, un semplice esame della figura 2.6.2.2 permette di riconoscere le seguenti linee guida, evidentemente adottate nella differenziazione della funzione di elaborazione quanti all'interno della famiglia di contatori ma mai espressamente evidenziate nella documentazione disponibile [DH020, DH023, DH025, DH026]. Riportiamo una lista, non esaustiva, dei criteri che traspaiono dalla gerarchia mostrata in figura 2.6.2.2:

- tutti i tipi di UE calcolano, essenzialmente nello stesso modo (salvo per il valore eventualmente diverso delle unità di misura e di alcuni parametri temporali: si vedano in proposito le osservazioni che seguono in questo paragrafo sulle "costanti differite"), il *load flow* statistico (valori dei momenti $\mu 1$ e $\mu 2$ del primo e del second'ordine della potenza consumata in un determinato intervallo all'interno della giornata);
- tutti i tipi di UE calcolano il valore corrente, e quello alla fine del precedente periodo di fatturazione, dell'energia consumata; le diverse UE differiscono però in relazione alle categorie di energia che vengono conteggiate: tutte considerano l'energia attiva complessiva, UEPB e UEPM considerano anche l'energia reattiva, e la sola UEPM distingue tra energia attiva assorbita ed erogata;
- solo UEP e UEPB registrano i valori all'istante intermedio delle grandezze misurate;
- solo UEP e UEPB calcolano e memorizzano l'energia consumata giornalmente;

Riteniamo che la strutturazione delle funzionalità dei diversi tipi di UE mediante l'uso della gerarchia di ereditarietà sia un importante strumento di modellizzazione e concettualizzazione, che può trovare un importante impiego anche al di fuori dello stretto ambito di specifica dei requisiti di progetto: esso potrebbe essere un utile ausilio anche nelle fasi preliminari di studio di fattibilità e analisi dei requisiti dell'utente.

Nella gerarchia ereditaria delle classi si è fatto uso sistematico di un tipo di definizione che nei linguaggi di programmazione orientati agli oggetti viene chiamato delle "entità differite". Si tratta di elementi che vengono dichiarati, in classi "alte" della gerarchia di ereditarietà, in modo incompleto ma sufficiente a permetterne fin da subito un uso appropriato. L'informazione mancante, relativa alle entità differite, viene poi fornita all'interno di ulteriori classi eredi, attribuendo pieno significato ai costrutti in cui tali entità compaiono.

Troviamo numerosi esempi di applicazione di questa idea nella specifica della funzione di elaborazione dei quanti di energia. La maggior parte delle costanti e dei parametri dell'attività (e.g., il valore del quanto di energia, le unità di misura dell'energia e della potenza), che sono presenti, se pur con valori diversi, in tutti i tipi di unità di elaborazione, sono dichiarate nella classe *elebQuantiUE*, la più alta nella gerarchia di ereditarietà. Tali costanti vengono utilizzate all'interno degli assiomi che specificano proprietà fondamentali e generali (e perciò comuni a tutte le unità di elaborazione) della funzione di elaborazione quanti. Il valore di tali costanti varia però da un tipo di unità di elaborazione all'altro e viene perciò precisato (mediante opportuni semplici assiomi) soltanto nelle classi foglia della gerarchia, che corrispondono ai singoli tipi di unità di elaborazione. Per esempio, la costante *valoreQuanto* viene dichiarata nella classe *elabQuantiUE* e in questa classe compare negli assiomi che specificano la quantità di energia consumata in funzione del conteggio dei quanti. Il valore preciso della costante è inessenziale nella specifica della relazione che lega il conteggio dei quanti alla quantità di energia consumata, e può quindi senza inconvenienti, anzi

aumentando la flessibilità e riutilizzabilità delle specifiche, essere specificato nelle classi foglia: infatti la classe *elabquantiUEP* contiene un assioma che specifica che *valoreQuanto* è uguale a 50 Wh, mentre *elabquantiUEPM* ne contiene uno che pone *valoreQuanto* uguale a 500 Wh.

L’uso di costanti differite permette di aumentare vieppiù la modificabilità delle specifiche: qualsiasi modifica dei parametri di una funzione per un determinato tipo di unità di elaborazione può essere formalizzata in modo estremamente semplice e circoscritto cambiando il singolo assioma che nella classe corrispondente ne specifica il valore.

Ai fini della produzione di una documentazione “esterna” al gruppo di progetto, puramente informativa e illustrativa, una gerarchia di classi per specificare una singola funzione potrebbe essere considerata eccessivamente complessa e forse difficilmente leggibile e fruibile. A tale proposito osserviamo che a partire da una gerarchia di classi, si può ottenere in modo sistematico (cioè mediante operazioni di tipo puramente editoriale e senza bisogno di alcun approfondimento concettuale) o addirittura automatico la produzione di documenti di specifica strutturalmente più semplici, risultanti da un’operazione che potremmo chiamare di “appiattimento” della gerarchia di ereditarietà, in cui tutte le informazioni relative a un particolare tipo di unità di elaborazione vengono raggruppate in una singola classe. A tale scopo è sufficiente considerare, lungo il grafo della gerarchia di ereditarietà, tutti i cammini che partono dalla classe al livello più alto per arrivare alla classe corrispondente al singolo tipo di unità do elaborazione. Con riferimento ancora all’esempio della funzione di elaborazione dei quanti, il documento unico relativo alla UEPB potrebbe essere ottenuto componendo le dichiarazioni delle classi *elabQuantiUE*, *elabQuantiUEP__UEPB*, *elabQuantiUEPB_UEPM* ed *elabQuantiUEPM*.

3.5 Fattori di criticità temporale

Nella descrizione della funzione di elaborazione dei quanti di energia, si è notato che molti requisiti informali prescrivono di effettuare, per una serie di grandezze, diverse operazioni nello stesso istante di tempo. Il caso emblematico di questa categoria di requisito è quello delle quantità *w15z*(*n*) e *qa15z*(*n*), che rappresentano i valori medi (in 15 minuti fissi, cioè sincroni rispetto all’inizio della giornata: *n* varia perciò da 1 a 96) della potenza attiva e reattiva assorbite nel giorno in corso. Tali valori devono essere memorizzati in due insiemi di registri, da azzerare all'inizio di ciascuna giornata. Inoltre, sempre a inizio giornata, i valori di *w15z*(*n*) e *qa15z*(*n*) relativi alla giornata precedente devono essere copiati in registri analoghi, detti *w15zp*(*n*) e *qa15zp*(*n*).

Occorre quindi assumere che a mezzanotte venga calcolato “istantaneamente” (in pratica, in un tempo trascurabile) la potenza media degli ultimi 15' della giornata, e assegnata “istantaneamente” a *w15z*(96) e a *qa15z*(96) e poi “istantaneamente” *w15z* e *qa15z* vengano assegnati a (cioè copiati nei registri) *w15zp* e *qa15zp*, che memorizzano i valori del giorno precedente, e poi diventino immediatamente nulli. Ciò non è impossibile a patto di interpretare in modo opportuno le nozioni di “instante” e “istantaneità”, ma introduce (inutilmente) una notevole criticità temporale e fa dipendere la correttezza dell’implementazione rispetto a questa specifica “ideale” da una tolleranza che non è quantificata esplicitamente.

Si può intuire che il vero significato di tali specifiche informali sia che le tre operazioni distinte (calcolo dell’ultimo valore, assegnamento di tutta la serie dei valori del giorno precedente e azzeramento dei registri per il giorno che sta iniziando) vengano eseguite non simultaneamente ma in sequenza, in un tempo non nullo ma sufficientemente piccolo da essere considerato ”trascurabile agli effetti esterni”. Questo tipo di criticità temporale può avere due tipi di conseguenze:

- nella fase di progetto e implementazione, se le specifiche non sono sufficientemente chiare ed esplicite può portare alla costruzione di sistemi che contengono gravi errori di temporizzazione;
- nella fase di specifica dei requisiti di progetto, e nelle successive collegate fasi di convalida e verifica, può portare a situazioni di contraddizione e inconsistenza.

La presenza di azioni a tempo nullo e simultanee ma sequenzializzate (se viste a un certo livello di astrazione) è stata in passato argomento di ricerca e oggetto di lavori pubblicati da parte degli autori

del presente rapporto. L’argomento presenta comunque degli aspetti di approfondimento semantico e di formalizzazione matematica che vanno ben al di là delle possibilità di approfondimento fornite da in questa sede; si rimanda quindi a questo proposito ai lavori [FMM94, GMM96].

Bibliografia

[CC&94] E.Ciapessoni, E. Corsetti, E. Crivelli, D. Mandrioli, E. Ratto, “Introduzione a TRIO: un ambiente per la specifica e l'analisi di sistemi in tempo reale”, Rapporto ENEL/CRA, 1994.

[DH020] ENEL SpA, “Sistema di Telegestione”, Direzione Divisione della Distribuzione, Rapporto DH020, Giugno 1990.

[DH023] ENEL SpA, “Sistema di Telegestione: Unità di elaborazione periferica UEP per GMY e GTY singoli”, Direzione Divisione della Distribuzione, Rapporto DH023, Settembre 1992.

[DH025] ENEL SpA, “Sistema di Telegestione: Unità di elaborazione periferica UEP per GTWD e DTWS”, Direzione Divisione della Distribuzione, Rapporto DH025, Giugno 1990.

[DH026] ENEL SpA, “Sistema di Telegestione: Unità di elaborazione periferica UEPM per GTWM”, Direzione Divisione della Distribuzione, Rapporto DH026, Giugno 1990.

[FMM94] M.Felder, D.Mandrioli, A.Morzenti, “Proving properties of real-time systems through logical specifications and Petri net models”, IEEE TSE-Transactions of Software Engineering, vol.20, no.2, Feb.1994, pp.127-141.

[GMM96] A.Gargantini, D.Mandrioli, A.Morzenti, “Dealing with zero-time transitions in axiom systems”, Rapporto interno del Dipartimento di Elettronica e Informazione del Politecnico di Milano, sottomesso per pubblicazione.

[M&S94] A. Morzenti, P. San Pietro, “Object-Oriented Logic Specifications of Time Critical Systems”, ACM TOSEM - Transactions on Software Engineering and Methodologies, vol.3, n.1, January 1994, pp. 56-98.

[M&S94] A.Morzenti, P. San Pietro, “Introduzione al linguaggio di specifica TRIO”, ENEL/CRA, 1994.

[MPS93] A. Morzenti, M. Pezzè, P. San Pietro, “Confronto di modelli di sistemi in tempo reale”, Rapporto ENEL/CRA, 1993.