Indice dei contenuti

Indice Analitico

Indice Diagrammi UML e non

Indice di estratti di codice

0. Premessa

0.1 Target – A chi è rivolto

Il presente lavoro (codice sorgente e documentazione) è rivolto principalmente a programmatori in linguaggio C di microcontrollori ad 8bit.

0.2 Target – Tipologia di applicazioni

Il ***firmware*** a cui la presente documentazione si riferisce è stato sviluppato pensando ad applicazioni PLC, sia per la parte funzionale che per quella interfaccia; le applicazioni PLC sono quelle applicazioni in cui devono essere gestite le logiche che legano gli ***ingressi*** alle ***uscite***; gli ingressi e le uscite possono essere sia ***digitali*** che ***analogici***.

0.3 Avvertenze

Nella presente documentazione ci sono ancora molte inesattezze (dovuti in parte anche alla difficoltosa sincronizzazione con il codice) e nel firmware aspetti da completare ma il codice sviluppato con le sue regole e i suoi modelli implementati, e qui descritti, già permette uno sviluppo modulare di una applicazione per microcontrollori 8-bit ed era questo l'obiettivo base di questo lavoro.

Alcuni concetti o termini di informatica specie nell'ambito della programmazione ad oggetti nel tentativo di portare questa tecnica allo sviluppo di applicazioni per microcontrollori 8-bit usando il linguaggio C sono stati, forse, un po' abusati, quindi se qualcuno ritene che siano stati usati impropriamente è pregato di farmelo notare (la mia mail è “md120308 at gmail dot com”), ringrazio in anticipo tutti coloro che mi faranno questa cortesia.

Nel caso qualche lettore gradisca un maggiore dettaglio di alcune parti della presente documentazione sarò ben lieto di esaudirne le richieste, stessa cosa per il firmware.

In ultimo desidero evidenziare che tutte le critiche positive e negative sulla presente documentazione e in generale sull'intero lavoro (firmware e documentazione) saranno bene accette.

1. Introduzione

1.1 Caratteristiche

Il codice è stato sviluppato pensando alle seguenti caratteristiche

- ***Ottimizzazione del codice***: minimizzare l'utilizzo di memoria ***ROM*** e ***RAM***
- ***Modularità***: sviluppare nuove funzionalità senza dover modificare il codice di altri moduli.

1.2 Modularità e suoi vantaggi

La seconda caratteristica ha numerosi vantaggi, ben noti a coloro che usano il paradigma della ***programmazione orientata agli oggetti***:

- Facilita il ***debug*** e quindi in generale anche la manutenzione del codice
- Facilita l'introduzione di nuove funzionalità minimizzando problemi di interferenza con altre parti del codice
- Permette un più veloce e “pulito” riutilizzo del codice in altri progetti anche con differenti ***piattaforme*** HW e/o SW
- Permette di identificare chiaramente le parti del codice indipendenti dalla piattaforma su cui far girare il fw (notare che la piattaforma può essere HW o SW)
- Possibilità di sviluppare progetti coinvolgendo più persone o gruppi e in ultima istanza creare in rete una repository di moduli e configurazioni fw/hw

1.3 Schemi e moduli ricorrenti in applicazioni PLC

In applicazioni PLC (ma non solo!) ci sono ***schemi*** e ***moduli*** spesso ricorrenti, di seguito i principali che sono stati implementati nel codice sorgente a cui la presente documentazione si riferisce:

- ***Schemi***
    - ***produttore-consumatore***: controllo del verificarsi di particolari situazioni o condizioni, chiamate anche eventi (Produzione evento), e successiva esecuzione di azioni associate a tali eventi (Consumazione evento)
    - ***timers di sistema***: quasi sempre i moduli hanno bisogno di eseguire delle procedure a particolari intervalli di tempo; ad esempio gestione dell'anti-rimbalzo nella lettura dei tasti oppure la gestione del lampeggio nei moduli interfaccia utente
    - ***funzioni di init***: quasi sempre i moduli hanno bisogno di una inizializzazione che deve essere eseguita alla partenza del microcontrollore

- ***Moduli***
  - ***gestione interfaccia utente*** (componenti base: lista di voci, modifica valori)
    - ***gestione parametri***
    - ***multilingua***

1.3 Descrizione piattaforme usate

Il codice sorgente a cui è associata la presente documentazione è stato implementato su due diverse piattaforme una HW ed una SW

- piattaforma HW: EasyPicV7 di MikroElektronica
- piattaforma SW: Windows7 Microsoft

Si vuole far notare che ci sono dei vantaggi nel poter eseguire il fw su piattaforme SW come Windows, Linux oppure OS X

- allestire debug automatici del fw (naturalmente sono esclusi i ***driver***, questi possono essere testati solo sulle piattaforme per cui sono scritti)
- rendere il progetto FW (ad esclusione dei driver) indipendente dal progetto HW a cui è associato

1.4 Progetti / Sviluppi futuri

Possibili sviluppi futuri del presente lavoro vertono su tre linee, una rivolta all'ambiente di sviluppo l'altra rivolta alla riutilizzabilità e condivisione del codice e l'ultima relativa ad un progetto PLC openSource:

- permettere lo sviluppo rapido di appicazioni per microcontrollori 8-bit, allo stesso modo delle applicazioni Windows/OS X/Linux
  - creare interfaccia utente con tool grafici senza modificare manualmente il codice nello stile degli ambienti di sviluppo come Delphi, QtCreator / QtDesign, VisualStudio oppure degli ambienti di sviluppo PLC per la programmazione delle interfacce utente
- creare in rete una repository per la condivisione di moduli relativi alle più disparate applicazioni e necessità, e driver relativi alle più svariate configurazioni di “microcontrollore + piattaforma”; per quest'ultimo aspetto già si può trovare qualcosa in rete
- sviluppare applicazioni nello stile del linguaggio LADDER ma sfruttando anche moduli più complessi che permettano di uscire dal paradigma procedurale che quando l'applicazione è leggermente più articolata crea difficoltà di manutenzione e debug; si veda in rete il progetto OpenPLC (progetto open) e CodeFlow (progetto proprietario)

Notare che i primi due punti sono tra loro strettamente legati.

1.5 Progetti simili o complementari

- FlowCode
- FemtoOS
- OpenPLC
- Librerie per vari microcontrollori per la gestione dei registri
- EICASLAB

2. Descrizione del Codice

2.1 Files e loro nomenclatura

Tutti i nomi dei file eccetto il file **main.c** rispettano la seguente codifica:

[hw_|ui_][kr|ui|fn|rs2|ms]<Nome del modulo>[Def|Tbl|Glo|Edt].[c|h]

2.1.1 Prefissi

Notare che ci sono due livelli di prefissi la prima termina con un underscore "_"; di seguito il significato e l'utilizzo di questi prefissi

2.1.1.1 Prefissi di primo livello

- **hw_**: evidenziare un file in cui il codice gestisce istruzioni dipendenti dalla piattaforma, sono i file che costituiscono i driver del sistema, tipicamente fanno parte dei moduli delle risorse (moduli rs2)
- **ui_**: evidenziare il file header che contengono variabili globali di un modulo che si desidera siano presenti nella lista dei parametri che si possono modificare via interfaccia utente; i file con questo primo livello di prefisso sono sempre i file con postfisso “Edt” e quindi sono sempre file header

2.1.1.1 Prefissi di secondo livello – Identificatore del gruppo di moduli

Questo livello di prefisso identifica anche il gruppo di moduli. Di seguito la loro descrizione:

- **kr**: (sta per **ke**r**r**nel) sono i moduli base di un progetto che non possono mai mancare; ci sono due moduli principali e ciascuno offre dei “servizi” agli altri moduli del firmware:
  - krProcess: è lo schedulatore dei processi; nel codice i processi sono rappresentati dalle righe della matrice dei processi e sono caratterizzati da una funzione che genera eventi (Produce un evento) e da una funzione che processa gli eventi (Consuma un evento)
  - krTimer: gestisce le funzioni evento associate alle variabili timer di cui necessitano i vari moduli del firmware
- **ui**: (sta per **u**ser **i**nterface) sono i moduli che si occupano di gestire l'interfaccia utente;
- **fn**: (sta per **f**u**n**zionale) sono i moduli funzionali dove è presente il codice che gestisce la logica tra gli ingressi e le uscite;
- **rs2**: (sta per RiSorse, togliendo le vocali si ha rsrs da cui rs2) sono i moduli che forniscono agli altri moduli l'interfaccia con l'hardware, tipicamente al proprio interno contengo i driver del sistema
- **ms**: (sta per **m**i**s**cellaneous), sono moduli generici che non apparetngono a nessuna delle precedenti categorie

2.1.2 Postfissi

- **Def**: (è un postfisso e sta per **Def**inition) sono quei file in cui sono “fisicamente” definite le variabili globali dei moduli; questi file sono sempre con estensione “.c” non hanno il nome dei moduli ma solo il prefisso di secondo livello.
- **Tbl**: (è un postfisso e sta per **T**a**bl**e) sono quei file in cui sono defnite Tabelle statiche necessarie per la gestione delle dipendenze indirette come ad esempio le funzioni timer dei moduli o gli eventi provenienti dai tasti, per maggiori dettagli si veda il paragrafo <2.4.1.1 Dipendenze Dirette e Indirette>;
- **Glo**: (è un postfisso e sta per **Glo**bal) è un file header in cui sono presenti strutture e variabili condivisi da più moduli, di solito quando venfgono condivise le strutture e/o le variabili c'è sempre un modulo di riferimento da cui il file prende anche il nome
- **Edt**: (è un postfisso e sta per **Ed**i**t**able) sono quei file in cui sono definite le variabili globali di un modulo con le loro proprietà di visualizzazione e di modifica che si vuole poter modificare via interfaccia utente.

2.2 Moduli / Componenti e Gruppi di Moduli / Componenti

I file con estensione “.c” ed “.h” possono essere suddivisi in base ai seguenti due criteri, formando rispettivamente moduli/componenti e gruppi di moduli/componenti:

- <prefisso di secondo livello> e <nome> in comune
- solo <prefisso di secondo livello> in comune

Di seguito alcuni esempi:

2.2.1 Esempio di Moduli / Componenti

- Modulo **krProcess** è costituito dai seguenti files: krProcess.c, krProcess.h, krProcessGlo.h, krProcessTbl.c;
- Modulo **krTimer** e costituito dai seguenti files: krTimer.c, krTimer.h, krTimerGlo.h, krTimerTbl.c;
- Modulo **uiPage** è costituito dai seguenti files: uiPage.c, uiPage.h, uiPageGlo.h

2.2.2 Esempio di Gruppo di Moduli / Componenti

- moduli del gruppo **kernel** identificati dal prefisso **kr**: i moduli sono krEvent, krProcess, krTimer, hw_krInterrupt;
- moduli del gruppo **interfaccia utente** identificati dal prefisso **ui**: i moduli sono uiPage, uiLbox, uiPar, uiVar, uiStr, uiDef

2.2.3 Driver

I driver non sono moduli ma sono semplicemente dei file identificati dal prefisso di primo livello “hw_” e in cui sono presenti istruzioni dipendenti dalla piattaforma; notare poi che questi file tipicamente dfanno parte dei moduli di risorse quindi con prefisso di secondo livello “rs2”; un caso particolare è il file “hw_krInterrupt.c” che è anche modulo non essendoci altri file con lo stesso prefisso **kr** e nome **Interrupt**.

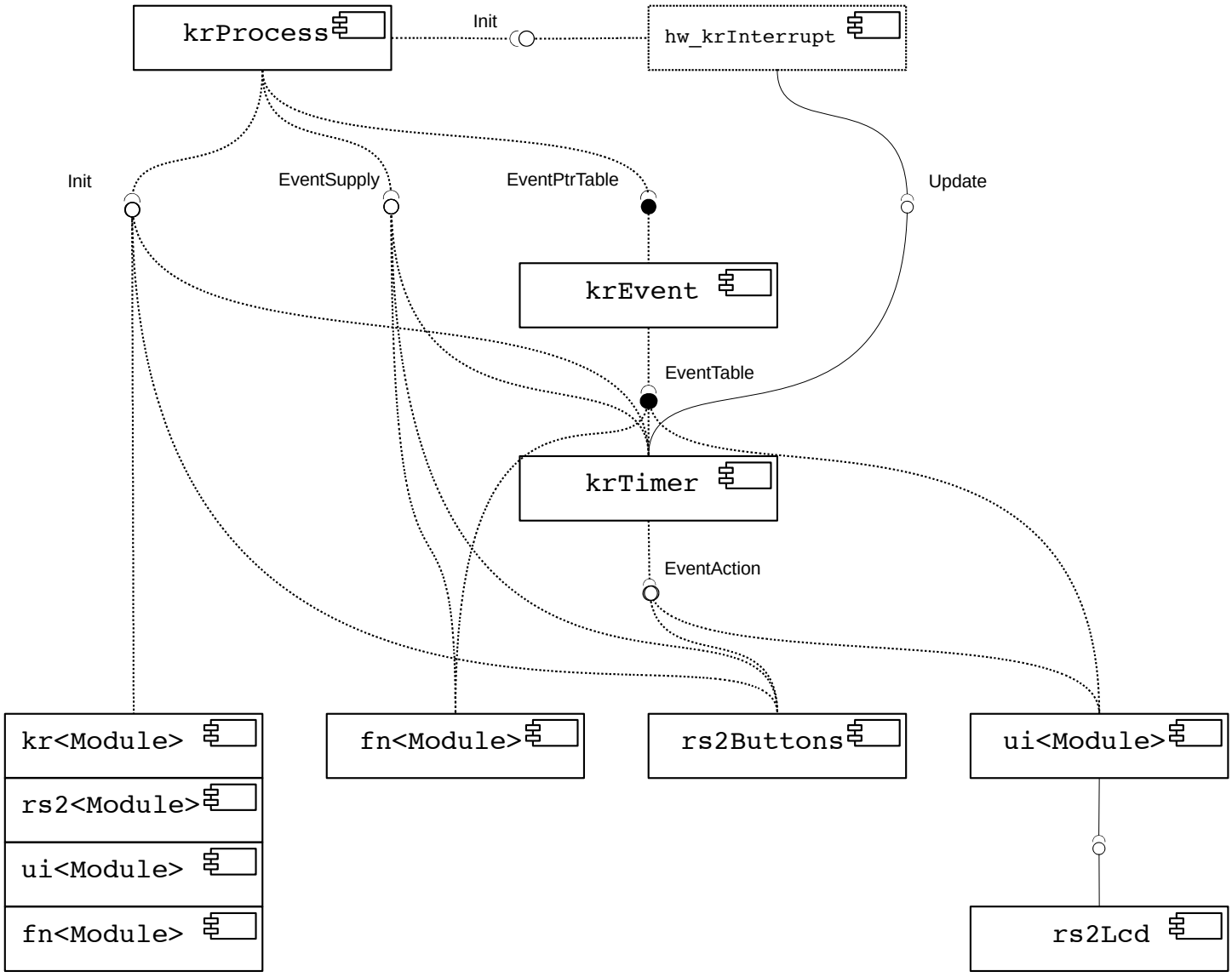
2.3 Descrizione del codice e Diagrammi UML

Per descrivere la struttura del fw ed il suo flusso di esecuzione ci si è avvalsi dell'uso dei diagrammi UML nella maggior parte delle situazioni; l'unica eccezione è il diagramma per la descrizione delle relazioni tra le tabelle dei componenti interfaccia definite nel file ***uiApplication.c*** ***uiApp_XXX_Tlb.c***

2.4 Struttura del codice

In questo paragrafo si descriverà la parte statica del codice attraverso i diagrammi dei componenti (visione globale della struttura del codice) e delle classi (visione più specifica e riguardante solo i moduli “ui” Interfaccia Utente):

2.4.1 Diagramma dei Componenti - Dipendenze tra i moduli



Diag. 1: Diagramma dei componenti – Descrizione delle dipendenze dei moduli

Legenda:

Dipendenza indiretta – Puntatore a Tabella di Puntatori a Funzione

Dipendenza indiretta – Puntatore a Funzione

Dipendenza diretta

2.4.1.1 Dipendenze Dirette e Indirette

Nella relazione di dipendenza tra due moduli si dice che il modulo A dipende dal Modulo B se A esegue una funzione X del modulo B, questo è il caso delle dipendenze dirette rappresentate da linee continue nel diagramma <Diag. 1: Diagramma dei componenti – Descrizione delle dipendenze dei moduli>; ci sono però situazioni in cui sebbene il modulo A esegua una funzione X del modulo B i ruoli sono invertiti, questo accade quando la funzione X viene eseguita in modo anonimo tramite un puntatore a funzione o come si dice in altri contesti (linguaggio C#, Java) tramite funzione anonima o funzione Lambda, questo è il caso delle dipendenze indirette rappresentate da linee tratteggiate sempre nel diagramma <Diag. 1: Diagramma dei componenti – Descrizione delle dipendenze dei moduli>; affinché una funzione X del modulo B venga eseguita in determinate situazioni il modulo B lo deve comunicare al Modulo A che, per esempio, è il modulo che fornisce questo "servizio", a sua volta per permettere ciò il modulo A mette a disposizione una funzione di inizializzazione H la quale prevede come parametri anche ed in particolare il puntatore alla funzione X. Quindi come si vede, in questo caso, di fatto è il Modulo B che dipende dal modulo A dovendo chiamare la funzione di inizializzazione H del modulo A.

Osservazione:

Nelle dipendenze indirette la funzione X del modulo B da eseguire diventa parametro della funzione di inizializzazione H del modulo A e chi la esegue, il modulo A, non ha il ruolo di dipendente ma di reggente; concludendo, l'uso di puntatori a funzione (o funzioni anonime, funzioni lambda) permette l'inversione dei ruoli nelle relazioni di dipendenza tra moduli e questo è un aspetto molto usato per lo sviluppo di codice modulare.

Nota 1:

Nel caso specifico del presente firmware la fase di inizializzazione del "servizio" viene eseguita a design time inserendo direttamente il puntatore alla funzione X in un apposito array quindi senza la chiamata della funzione di inizializzazione H; gli array di cui si è appena fatta menzione sono definiti nei file Tbl; è stata scelta questa impostazione per ottimizzare il codice.

I servizi o dipendenze indirette presenti nel firmware sono:

- **Init**: funzione che viene eseguita alla partenza del micro prima che inizi il loop;
- **Timer / Event Action**: funzione che viene eseguita allo scadere della variabile timer ad esso associata;
- **EventSupply / Controllo**: funzione eseguita ad ogni ciclo di loop che ha il compito di controllare il verificarsi di specifici eventi come ad esempio la pressione di un tasto oppure la ricezione di messaggi provenienti dalla seriale;
- **EventPtrTable**: Array statici di puntatori a tabelle di funzione i cui elementi non possono essere cambiati run time; contengono le tabelle che a loro volta contengono le funzioni disposte in base alla codifica degli eventi vengono eseguite al verificarsi di eventi segnalati dalle funzioni di cui al punto precedente;

- **EventTable**: Array dinamici di puntatori a tabelle di funzioni contenuti nella tabella di cui al punto precedente; i suoi elementi possono essere cambiati durante l'esecuzione (run time) e contengono i puntatori è il caso delle tabelle eventi dei componenti interfaccia.

Nota 2:

I due ultimi servizi potrebbero essere inglobati in uno unico, il sottoscritto però ha avuto problemi nel farlo. E' un aspetto da approfondire per semplificare il codice e ottimizzarlo ulteriormente.

Nel diagramma dei componenti le dipendenze indirette vengono rappresentate dal simbolo del lecca-lecca (lollypop) tratteggiato, accanto al punto di aggancio è riportato il nome del tipo di funzione da eseguire

2.4.1.2 Fasi nello sviluppo del codice

Nella realizzazione di un codice modulare ci sono due fasi:

- **Fase I**: implementazione dei moduli; si concretizza nella scrittura dei file con il nome dei moduli ad eccezione dei file Tlb;
- **Fase II**: assemblaggio dei moduli; i moduli devono essere messi insieme, praticamente questo avviene nei file Tbl. Nel caso specifico del presente firmware l'assemblaggio dei moduli avviene a **design-time**, appunto scrivendo i file Tlb; si è scelta questa impostazione nell'ottica dell'ottimizzazione del codice tuttavia a seconda dei casi e delle esigenze può essere provata anche la via del **run-time**;

2.4.1.3 Diagramma e Codice

Il diagramma fornisce le seguenti informazioni esplicite ovvero direttamente legate al codice:

- i moduli nelle dipendenze indirette che hanno il ruolo di dipendente implementeranno funzioni di init , timer, controllo, oppure tabelle eventi

Il diagramma fornisce le seguenti informazioni implicite:

- in base alla *nota 1* del paragrafo precedente; i moduli nelle dipendenze indirette che forniscono il servizio e che quindi hanno il ruolo di reggenti saranno caratterizzati da un file con postfisso Tlb, questo file verrà implementato nella fase di assemblaggio dei moduli.

Esempi:

Modulo “rs2Buttons”

il modulo implementa funzioni delle seguenti tipologie:

- ***Init***
- ***Timer****: funzione che contiene l'implementazione della macchina a stati relativa alla lettura dello stato dei tasti*
- ***EventSupply / Controllo****: nel caso specifico del presente firmware questa funzione verrà*

inserita, nella fase di assemblaggio , nella matrice dei processi e accoppiata con la tabella degli eventi dei moduli interfaccia; si evidenzia che la tabella degli eventi a cui è associata può cambiare nel corso dell'esecuzione del fw ed essere a seconda delle situazioni quella del modulo uiPage, uiLbox o del mdoulo uiPar

Modulo "krTimer"

Il modulo usufruisce di alcuni servizi (Init, Controllo, Update) ma fornisce anche un servizio (timer di sistema) quindi può assumere sia un ruolo di dipendente sia di reggente nelle relazioni di dipendenza indiretta.

Il modulo nelle relazioni di dipendenza indiretta con ruolo di "dipendente" implementa i seguenti tipi di funzioni

- ***Init***
- ***EventSupply / Controllo***
- ***Update****: necessaria per gestire il decremento dei timer all'interno delle routine di interrupt*

Il modulo, nel ruolo di "reggente", nella relazione di dipendenza indiretta è caratterizzata dal seguente file:

- *file con postfisso **Tlb**: in questo file è implementato l'array delle funzioni Evento Timer; notare che ad ogni funzione è associato una variabile timer è associato un array delle variabili timer*

Il presente diagramma essendo un diagramma statico non entra nei dettagli di come siano implementate le funzioni di cui dichiara l'esistenza, per questi aspetti ci si deve rivolgere principalmente a tre diagrammi: il diagramma delle sequenze, diagramma della macchina a stati, diagramma delle attività; nella presente documentazione si farà uso più avanti solo dei primi due diagrammi, vedere paragrafo <2.5 Flusso di esecuzione del codice>.

2.4.1.4 Tabelle di aggregazione dei Moduli

Le tabelle di aggregazione dei moduli sono presenti nei file con postfisso "Tlb", i moduli del kernel forniscono le seguenti tabelle (o anche "servizi") di aggregazione:

- **Tabella Init**: è un semplice array tuttavia è importante tener presente che l'ordine di inserimento delle funzioni è anche quello di esecuzione quindi si deve stabilire a priori l'ordine di inserimento delle funzioni di init;
- **Tabella Timer**: è un semplice array di puntatori a funzione; tuttavia la posizione di questi puntatori all'interno della tabella deve corrispondere a quella delle variabili timer ad essi associate presenti nell'array dei Timer; di seguito vedere riquadri <Codice 1: Tabella EventTimer – dal file "krTimerTbl.c"> e <Codice 2: Definizione Identificativi delle variabili Timer – dal file "krTimerGlo.h"> in cui sono riportate rispettivamente la tabella degli eventi timer e la definizione degli identificativi delle variabili timer;

- **Tabella/Matrice dei Processi**: ogni riga di questa matrice rappresenta un processo, eseguito ad ognio ciclo di loop, in cui è presente una funzione generatore di evento e la sua corrispettiva funzione consumatore evento (più precisamente è una tabella di funzioni il cui ordine rappresenta la codifica dell'evento); per maggiori dettagli vedere il paragrafo <2.5.1.1 Schema Produttore/Consumatore> e relativo riquadro <Codice 6: Matrice dei processi – dal file “krProcessTbl.c”>

Osservazione:

Alcune caratteristiche dei moduli a livello globale possono essere dedotte dal diagramma e sono le seguenti:

- *i moduli kernel non dipendono da nessun altro modulo, e rappresentano la radice delle dipenendenze, tutti i moduli dipendono dai moduli kernel*
- *i moduli funzionali e quelli dell'interfaccia utente sono tra loro indipendenti*
- *il modulo rs2Buttons e i moduli interfaccia sono tra loro indipendenti ma comunicano attraverso lo scambio di messaggi rappresentati da eventi; questo scambio è permesso dall'accoppiamento tra la funzione di generazione degli eventi da tasti (modulo “rs2Buttons”) con le tabelle eventi dei moduli interfaccia (moduli “ui”)*
- *i moduli che usufruiscono di “servizi” devono definire appropriate funzioni e/o tabelle di funzioni .*

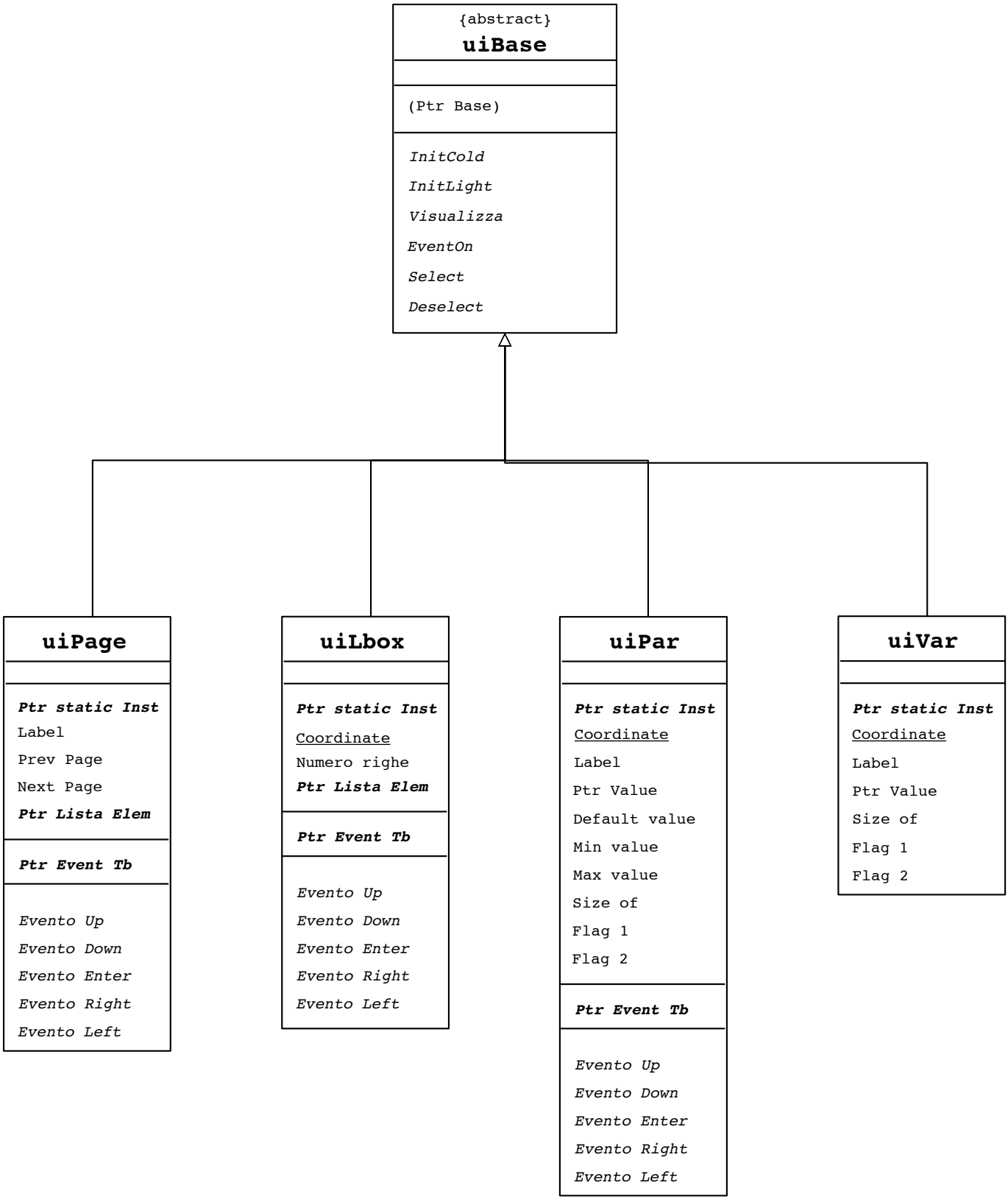
```
_ROM EventTableDef EventTimer[TMR_NUMBER] = {
                                TestTmrSecond,    // TMR_SECOND        0  //  modulo Test
                                TestTmrGLCD,      // TMR_TEST_GLCD     1  //  modulo Test
                                EventNull,        // TMR_TOUCH         2  //  modulo Touch
                                EventNull,        // TMR_LCD_INIT      3  //  non fare niente
                                ButtonsTimer,     // TMR_BUTTONS       4  //  modulo Buttons
                                uiParTimer,       // TMR_UI_PAR        5  //  modulo Parameter
                                uiLboxTimer       // TMR_UI_LBOX       6  //  modulo ListBox
};                                                // TMR_NUMBER        7
```

Codice 1: Tabella EventTimer – dal file “krTimerTbl.c”

```
#define TMR_SECOND      0  //  modulo Test
#define TMR_TEST_GLCD   1  //  modulo Test
#define TMR_TOUCH       2  //  modulo Touch
#define TMR_LCD_INIT    3  //  non fare niente
#define TMR_BUTTONS     4  //  modulo Buttons
#define TMR_UI_PAR      5  //  modulo Parameter
#define TMR_UI_LBOX     6  //  modulo ListBox
#define TMR_NUMBER      7
```

Codice 2: Definizione Identificativi delle variabili Timer – dal file “krTimerGlo.h”

2.4.2 Diagramma delle classi - Strutture e gerarchia dei moduli “ui”



Diag. 2: Diagramma delle Classi – Descrizione della gerarchia dei componenti "ui"

In questo diagramma si descrivono le strutture dati e il tipo di funzioni presenti nei moduli interfaccia;

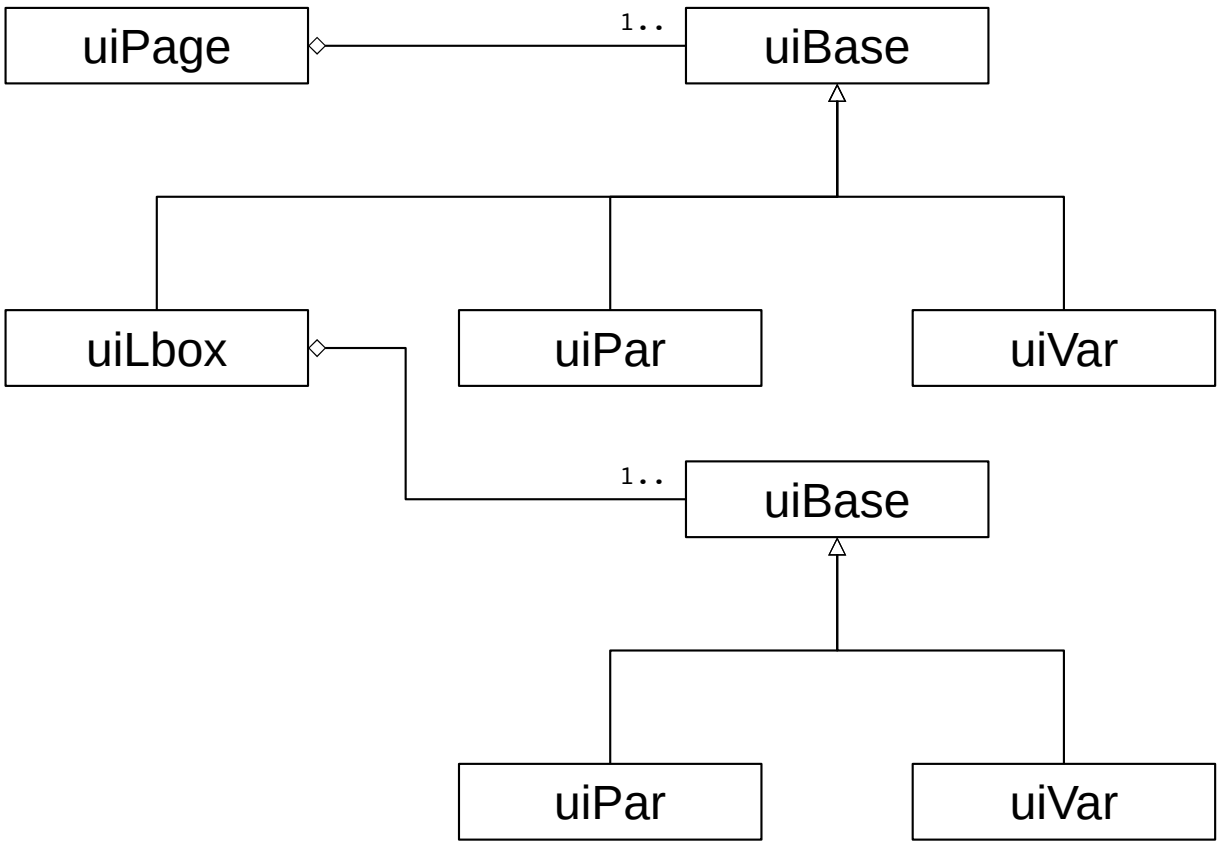
2.4.2.1 Descrizione dei componenti: cosa fanno

- ***uiPage***: questo componente è l'equivalente del componente Form delle applicazioni windows, ha la caratteritica di fare da contenitore per gli altri componenti “grafici” (vedere <Diag. 3: Diagramma delle Classi – Descrizione delle aggregazioni tra i componenti “ui”>);
- ***uiLbox***: questo componente è l'equivalente del componente ListBox delle applicazioni windows in cui si può selezionare un elemento da una lista;
- ***uiPar***: questo componente è l'equivalente dell'EditBox delle applicazioni windows ma specifico per parametri numerici; questo componente è usato per modificare i parametri del sistema tipicamante dei moduli funzionali ma non solo; le istanze statiche di questo modulo vengono definite nei file con postfisso Edt e poi raggruppate nella tabella delle istanze del modulo Par nel file uiApp_XXX_Tlb.c
- ***uiVar***: questo componente è l'equivalente di una Label dinamica (ovvero di una label il cui valore viene aggiornato periodicamente) delle applicazioni Windows; questo componente viene usato per visualizzare variabili di sistema tipicamente dei moduli funzionali ma non solo; le istanze statiche di questo modulo vengono definite nei file con estensione Dsp e poi raggruppate nella tabella delle istanze del modulo Var nel file uiApp_XXX_Tlb.c

2.4.2.2 Funzioni del modulo uiBase

Le funzioni definite nel modulo uiBase sono quelle che permettono l'interazione tra i moduli interfaccia, in particolare le funzioni ***Visaulizza***, ***Select***, sono le le funzioni che i moduli uiPage e uiLbox usano per comunicare con gli elementi che contengono, mentre la funzione ***EventOn*** è la funzione che gli tutti i componenti usano per impostare la tabella degli eventi del componente che si vuole attivare nella matrice dei processi, a tal proposito vedere i diagrammi <Diag. 8: Macchina a Stati – Passaggi di stato per gli oggetti del modulo uiPage> e <Diag. 9: Macchina a Stati – Passaggi di stato per gli oggetti dei moduli Lbox e Par>.

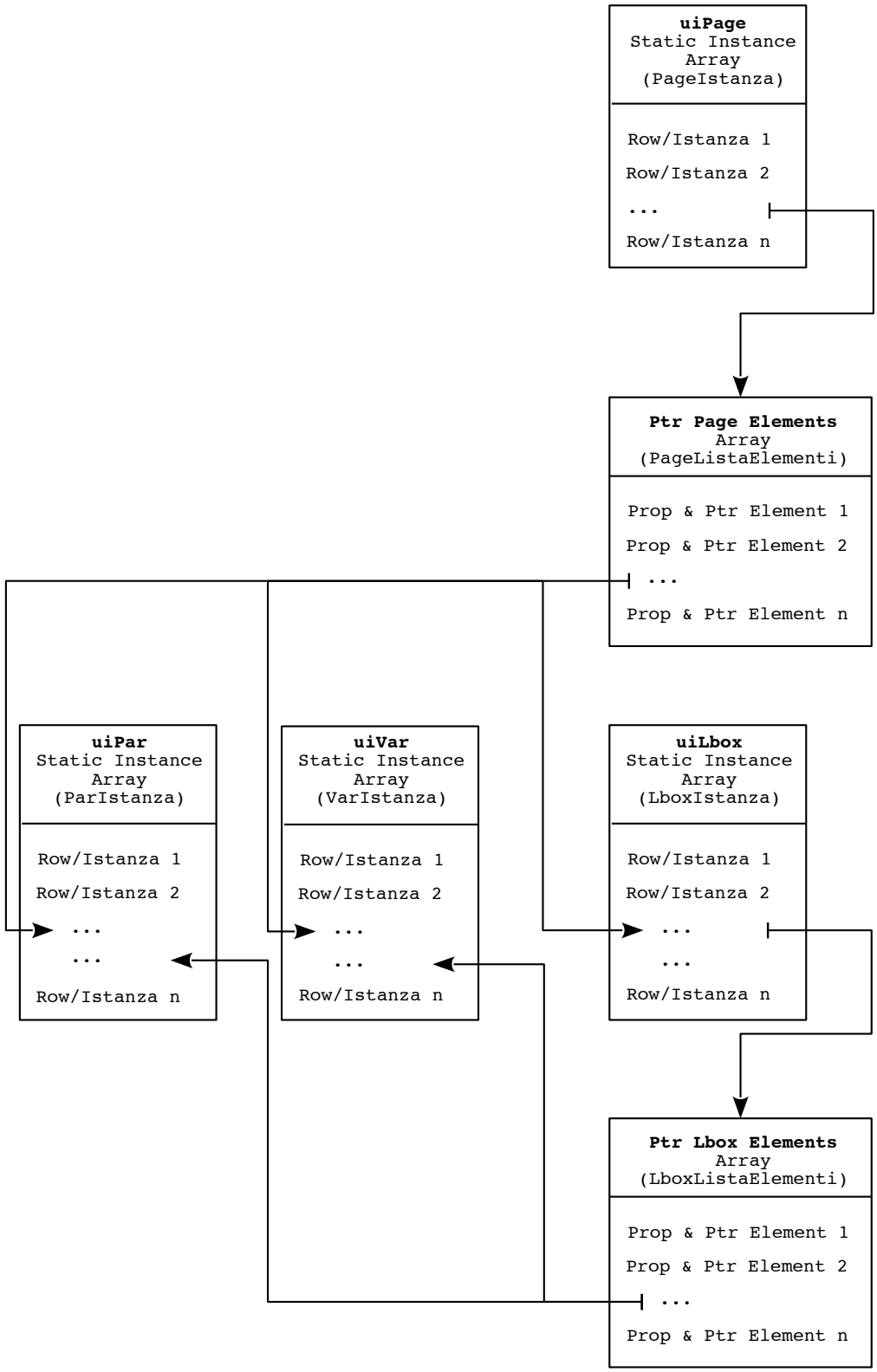
2.4.3 Diagramma delle classi - Relazione di aggregazione tra i moduli “ui”



Diag. 3: Diagramma delle Classi – Descrizione delle aggregazioni tra i componenti “ui”

Questo diagramma dice che un'istanza del modulo uiPage può contenere le istanze di tutti gli altri elementi uiLbox, uiPar e uiVar, mentre un'istanza di uiLbox può contenere solo le istanze dei componenti uiPar e uiVar; lo schema di aggregazioni tra componenti viene implementato nel codice con la gestione delle tabelle descritte nel diagramma che segue <Diag. 4: Tabelle dei componenti “ui” – Descrizione delle relazioni>, queste tabelle sono definite nel file “uiApp_XXX_Tbl.c” per maggiore chiarezza queste tabelle sono risportate anche nei riquadri <Codice 4: Tabella lista elementi dei moduli "uiPage" e "uiLbox" - dal file "uiApplication.c"> e <Codice 3: Tabella delle istanze dei componenti “ui” - dal file “uiApplication.c”>

2.4.4 Tabelle dei moduli “ui”



Diag. 4: Tabelle dei componenti “ui” – Descrizione delle relazioni

Come detto nel precedente paragrafo il presente diagramma descrive le relazioni tra le tabelle caratterizzanti i moduli interfaccia

Le tabelle riportate nel presente diagramma sono di due tipi:

- **Tabella delle istanze**
- **Tabella della lista di componenti**; solo per i componenti Page e Lbox che hanno la proprietà di contenere altri componenti interfaccia come uiPar e uiVar;

Di seguito si riportano due estratti del file “uiApp_XXX_Tbl.c” che mostrano l'implementazione delle suddette tabelle:

```
// MODULO VAR

// Definizione identificativi delle variabili di stato
#define  ID_VAR_TERMOSTATO_TEMPERATURA              0
#define  ID_VAR_TERMOSTATO_UMIDITA                  1
#define  NUMERO_VAR_ELEMENTI                        2

// Definizione tabella
_ROM VarIstanzaStruct VarIstanza[NUMERO_VAR_ELEMENTI] = {
    ROW_VAR_TERMOSTATO_TEMPERATURA     // ID_VAR_TERMOSTATO_TEMPERATURA       0
   ,ROW_VAR_TERMOSTATO_UMIDITA         // ID_VAR_TERMOSTATO_UMIDITA           1
};


// MODULO PAR

// Definizione identificativi dei parametri
#define  ID_PAR_TERMOSTATO_SETPOINT                 0
#define  ID_PAR_TERMOSTATO_SETPOINT_MIN             1
#define  ID_PAR_TERMOSTATO_SETPOINT_MAX             2
#define  ID_PAR_TERMOSTATO_DELTA                    3
#define  ID_PAR_TERMOSTATO_TEMPO_MINUTI             4
#define  ID_PAR_TERMOSTATO_TEMPO_SECONDI            5
#define  ID_PAR_GENERAL_LANGUAGE                    6
#define  NUMERO_PAR_ELEMENTI                        7

// Definizione tabella
_ROM ParIstanzaStruct ParIstanza[NUMERO_PAR_ELEMENTI] = {
    ROW_PAR_TERMOSTATO_SETPOINT        // ID_PAR_TERMOSTATO_SETPOINT             0
   ,ROW_PAR_TERMOSTATO_SETPOINT_MIN    // ID_PAR_TERMOSTATO_SETPOINT_MIN         1
   ,ROW_PAR_TERMOSTATO_SETPOINT_MAX    // ID_PAR_TERMOSTATO_SETPOINT_MAX         2
   ,ROW_PAR_TERMOSTATO_DELTA           // ID_PAR_TERMOSTATO_DELTA                3
   ,ROW_PAR_TERMOSTATO_TEMPO_MINUTI    // ID_PAR_TERMOSTATO_TEMPO_MINUTI         4
   ,ROW_PAR_TERMOSTATO_TEMPO_SECONDI   // ID_PAR_TERMOSTATO_TEMPO_SECONDI        5
   ,ROW_PAR_GENERAL_LANGUAGE           // ID_PAR_GENERAL_LANGUAGE                6
};


// MODULO LBOX

// Definizione identificativi dei list box
#define  ID_LBOX_TST_1                  0
#define  ID_LBOX_TST_2                  1
#define  ID_LBOX_SETTINGS_A             2
#define  ID_LBOX_SETTINGS_B             3
#define  NUMERO_LBOX_ELEMENTI           4

// Definizione tabella
_ROM LboxIstanzaStruct LboxIstanza[NUMERO_LBOX_ELEMENTI] = {
//    dimensione (numero righe)  ,Id_Start_lista_elementi              ,numero_elementi
    { 2                          ,ID_LBOX_TST_1_START_ELEMENT          ,4                }  // ID_LBOX_TST_1                0
   ,{ 1                          ,ID_LBOX_TST_2_START_ELEMENT          ,3                }  // ID_LBOX_TST_2                1
   ,{ 3                          ,ID_LBOX_SETTINGS_A_START_ELEMENT     ,6                }  // ID_LBOX_SETTINGS_A           2
   ,{ 2                          ,ID_LBOX_SETTINGS_B_START_ELEMENT     ,2                }  // ID_LBOX_SETTINGS_B           3
};


// MODULO PAGE

// Definizione identificativi delle pagine
#define  ID_PAGE_HOME                   0
#define  ID_PAGE_SETTINGS_A             1
#define  ID_PAGE_SETTINGS_B             2
#define  ID_PAGE_INFO                   3
#define  NUNERO_PAGE_ELEMENTI           4


// Definizione tabella
_ROM PageIstanzaStruct  PageIstanza[NUNERO_PAGE_ELEMENTI] = {
//    Titolo                ,Id_Start_lista_elementi          ,n ,IdPrevNext         ,IdPageNext
    { ID_STR_PAGE_HOME      ,ID_PAGE_HOME_START_ELEMENT       ,2 ,ID_PAGE_INFO       ,ID_PAGE_SETTINGS_A  } // ID_PAGE_HOME        0
   ,{ ID_STR_PAGE_SETTINGS_A ,ID_PAGE_SETTINGS_A_START_ELEMENT ,1 ,ID_PAGE_HOME       ,ID_PAGE_SETTINGS_B  } // ID_PAGE_SETTINGS_A  1
   ,{ ID_STR_PAGE_SETTINGS_B ,ID_PAGE_SETTINGS_B_START_ELEMENT ,1 ,ID_PAGE_SETTINGS_A ,ID_PAGE_INFO        } // ID_PAGE_SETTINGS_B  2
   ,{ ID_STR_PAGE_INFO       ,ID_PAGE_INFO_START_ELEMENT       ,3 ,ID_PAGE_SETTINGS_B ,ID_PAGE_HOME        } // ID_PAGE_INFO        3
};
```

Codice 3: Tabella delle istanze dei componenti “ui” - dal file “uiApplication.c”

come si vede nella tabella delle istanze dei moduli Page e Lbox in ogni riga (o istanza) ci sono le informazioni della lista degli elementi che l'istanza di quel componente contiene

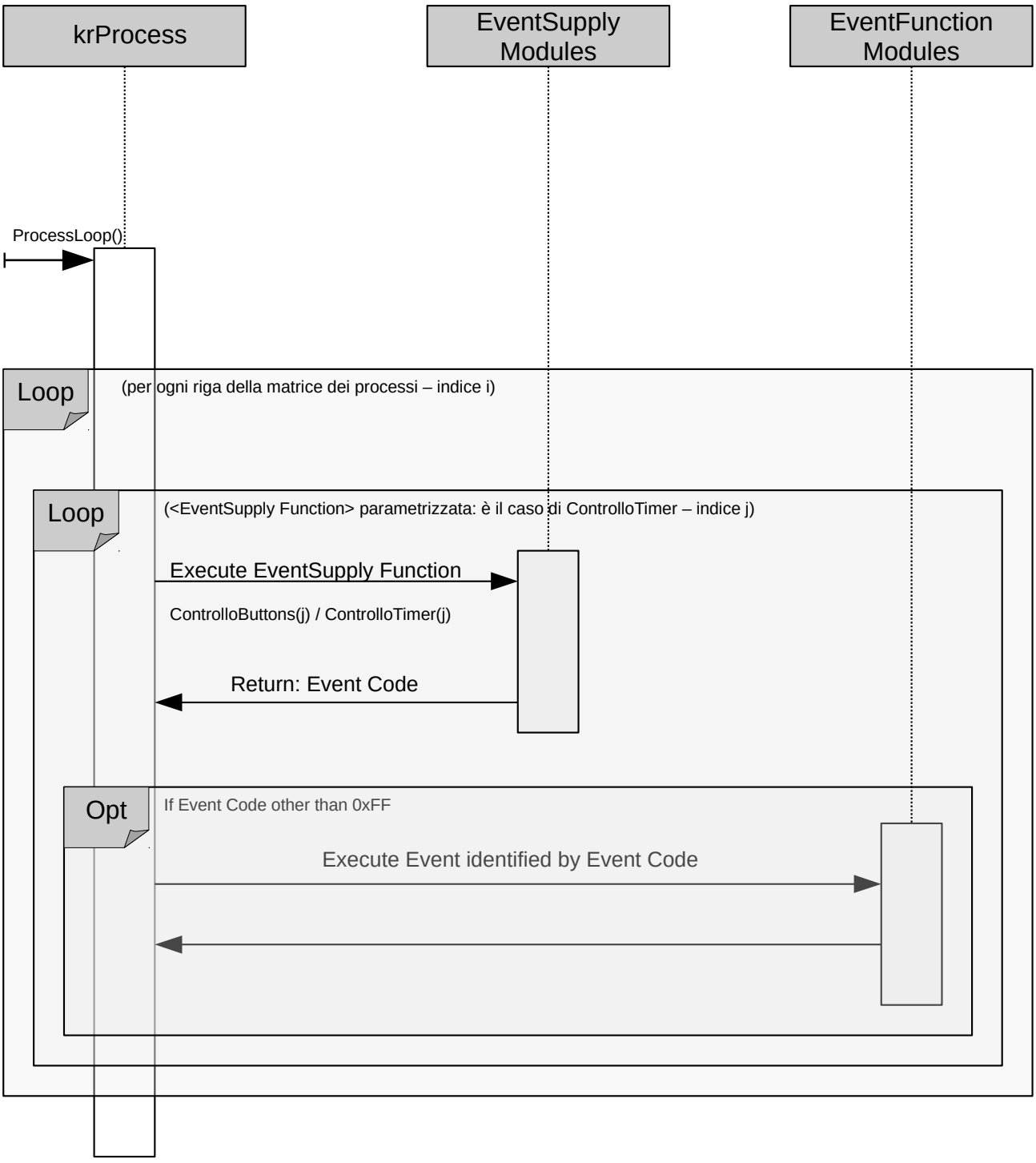
```
// SEZIONE - TABELLE DELLE LISTE ELEMENTI DEI MODULI PAGE E LBOX

//
_ROM LboxListaElementiStruct LboxListaElementi[NUMERO_LBOX_LISTA_ELEMENTI] = {
//    IdParametro                       ,TipoElemento (Label, EditBox)
    { ID_PAR_TERMOSTATO_SETPOINT         ,EL_PAR                       } // ID_LBOX_ELEMENT_0               0
   ,{ ID_PAR_TERMOSTATO_SETPOINT_MIN     ,EL_PAR                       } // ID_LBOX_ELEMENT_1               1
   ,{ ID_PAR_TERMOSTATO_SETPOINT_MAX     ,EL_PAR                       } // ID_LBOX_ELEMENT_2               2
   ,{ ID_PAR_TERMOSTATO_DELTA            ,EL_PAR                       } // ID_LBOX_ELEMENT_3               3
   ,{ ID_PAR_TERMOSTATO_TEMPO_MINUTI     ,EL_PAR                       } // ID_LBOX_ELEMENT_4               4
   ,{ ID_PAR_TERMOSTATO_TEMPO_SECONDI    ,EL_PAR                       } // ID_LBOX_ELEMENT_5               5
   ,{ ID_VAR_TERMOSTATO_TEMPERATURA      ,EL_VAR                       } // ID_LBOX_ELEMENT_6               6
   ,{ ID_VAR_TERMOSTATO_UMIDITA          ,EL_VAR                       } // ID_LBOX_ELEMENT_7               7
   ,{ ID_PAR_GENERAL_LANGUAGE            ,EL_PAR                       } // ID_LBOX_ELEMENT_8               8
};


//
_ROM PageListaElementiStruct  PageListaElementi[NUMERO_PAGE_LISTA_ELEMENTI] = {
//   IdElemento                      ,TipoElemento ,Pos:Row ,Pos:Column ,Dim:Width
   { ID_PAR_TERMOSTATO_SETPOINT_MIN  ,EL_PAR       ,{1      ,0          ,0}   }  // ID_PAGE_HOME_ELEMENT_0          0
  ,{ ID_LBOX_SETTINGS_A              ,EL_LBOX      ,{2      ,1          ,0}   }  // ID_PAGE_SETTINGS_A_ELEMENT_0    5
  ,{ ID_LBOX_SETTINGS_B              ,EL_LBOX      ,{2      ,0          ,0}   }  // ID_PAGE_SETTINGS_B_ELEMENT_0    6
  ,{ ID_STR_WEB_SITE                 ,EL_STR       ,{2      ,0          ,0}   }  // ID_PAGE_INFO_ELEMENT_0          7
  ,{ ID_STR_E_MAIL                   ,EL_STR       ,{3      ,0          ,0}   }  // ID_PAGE_INFO_ELEMENT_1          8
  ,{ ID_STR_PHONE                    ,EL_STR       ,{4      ,0          ,0}   }  // ID_PAGE_INFO_ELEMENT_2          9
};
```

Codice 4: Tabella lista elementi dei moduli "uiPage" e "uiLbox" - dal file "uiApplication.c"

2.5 Flusso di esecuzione del codice

In questo paragrafo vengono descritte le parti dinamiche del codice si farà quindi uso dei diagrammi delle sequenze e delle macchine a stati

2.5.1 Diagramma delle sequenze – Main Loop



Diag. 5: Diagramma delle Sequenze - Descrizione "main loop"

2.5.1.1 Schema Produttore/Consumatore

Il diagramma delle sequenze riportato nella pagina precedente illustra il flusso di esecuzione principale del firmware e allo stesso tempo il meccanismo del modello Produttore/Consumatore.

Per maggiore chiarezza di seguito si riporta il codice corrispondente al diagramma.

```
void ProcessLoop (void)
{
   char i, j;
   unsigned char evento;

   for(i=0;i<NUMBER_PROCESS;i++)
   {
      for(j=0;j<ProcessTable[i].NumeroRipetizioniControllo;j++)
      {
         evento = (ProcessTable[i].Controllo)(j);
         if(evento != 0xFF)
         {
            (ProcessTable[i].EventMatrix[0].EventTable[evento])();
         }
      }
   }
}
```

Codice 5: "main loop" e modello Produttore / Consumatore – dal file “krProcess.c”

Questo codice implementa attraverso la gestione della matrice dei processi (“ProcessTable”, si veda il riquadro “Codice 2”) il modello Produttore/Consumatore. Ogni riga di questa matrice rappresenta un processo (in cui è presente la generazione ed il conseguente processamento di un evento), gli elementi del processo sono:

- la funzione di generazione eventi che ha il ruolo di Produttore; ad esempio: “ControlloButtons”
- la tabella delle funzioni eventi che ha il ruolo di Consumatore; ad esempio “ButtonsEventMatrix”
- limite per l'inidce di parametrizzazione della funzione Produttore (per maggiori dettagli leggere la nota che segue)

i primi due elementi sono in relazione tra loro più precisamente devono condividere la stessa codifica degli eventi, per maggiori dettagli vedere la nota seguente.

```
_ROM ProcessStruct ProcessTable[NUMBER_PROCESS] = {
{ControlloTimer0     ,EventTimerMatrix       ,TMR_NUMBER},     // PROCESS_TIMER        0
{ControlloButtons    ,ButtonsEventMatrix     ,1}               // PROCESS_BUTTONS      1
};                                                             // NUMBER_PROCESS       2
```

Codice 6: Matrice dei processi – dal file “krProcessTbl.c”

Note sul codice:

- *la relazione tra i due elementi consiste nella condivisione della codifica degli eventi*
- *la posizione delle funzioni evento all'interno della tabella rispettano la codifica degli eventi, come esempio si veda la tabella eventi del componente uiLbox riportata nel riquadro Codice 3*

```
_ROM EventTableDef uiLboxEvent[NUMBER_EVENT_BUTTONS] = {
    uiLboxEventButtonsRight         //    EVENT_BUTTONS_RIGHT     0
   ,uiLboxEventButtonsUp            //    EVENT_BUTTONS_UP        1
   ,uiLboxEventButtonsEnter         //    EVENT_BUTTONS_ENTER     2
   ,uiLboxEventButtonsDown          //    EVENT_BUTTONS_DOWN      3
   ,uiLboxEventButtonsLeft          //    EVENT_BUTTONS_LEFT      4
};                                  //    NUMBER_EVENT_BUTTONS    5
```

Codice 7: Tabella eventi – dal file "uiLbox.c"

- *dal codice (riquadro Codice 1) si nota che la funzione Controllo (che sarebbe la funzione EventSupply) è parametrizzata; il parametro, che ha come limite superiore il terzo elemento presente nella riga "processo" della matrice dei processi, viene usato solo nel caso della funzione "ControlloTimer" mentre per la funzione "ControlloButtons" non viene considerato; questo è stato fatto per gestire in modo uniforme le funzioni di controllo (o EventSupply) dei timer e dei tasti e di altri eventuali mdouli;*

2.5.2 Diagrammi Macchina a Stati – Sistema Interfaccia

2.5.2.1 Introduzione

Nel presente firmware ci sono tre situazioni che possono essere schematizzate/implementate secondo un modello di macchina a stati:

- Gestione dell'interfaccia utente;
- Gestione della visualizzazione di una lista di n elementi all'interno di una finestra con m righe con m<=n;
- Gestione lettura tasti.

Le macchine a stati possono implementarsi a grandi linee in tre modi differenti:

- via codice con l'uso delle istruzioni di **if-then-else** o più in generale di **switch**;
- via **matrice stato-evento**;
- via schema **produttore/consumatore** con la definizione di una classe per ogni stato; a livello di codice questo implica la creazione di più modlui ed inoltre necessita anche dell'implementazione di un modello produttore/consumatore e quindi anche di una funzione generatore evento (ad esempio implementata nel modulo **rs2Buttons**) e di una tabella degli eventi (ad esempio implementate nei **moduli interfaccia**).

Tutte e tre queste modalità sono state implementate nel presente firmware, in particolare la:

- gestione interfaccia fa uso del **modello Produttore/Consumatore**;
- gestione della visualizzazione di una lista elementi fa uso di una **matrice degli stati-eventi** in cui le righe e le colonne sono rappresentate dallo stato e dall'evento mentre gli elementi sono dei puntatori a funzione che identificano l'azione da intraprendere al verificarsi delle coppie stato-evento;
- gestione lettura tasti fa uso delle istruzioni **if-then-else** / **swtitch**;

2.5.2.2 Istanze e Oggetti

Come accennato sopra la gestione dell'interfaccia utente passa per la definizione di classi/moduli di componenti “grafici” le cui strutture e relazioni sono gia state descritte nei paragrafi delle classi, mentre adesso descriveremo il loro aspetto dinamico e quindi il comportamento delle loro istanze.

Nell'ambito della programmazione ad oggetti

- L'istanza è un puntatore alla struttura della classe, il puntatore contiene il riferimento ad una porzione di memoria, presente nello heap, riservata alla struttura dati della classe; sotto certi punti di vista l'istanza potrebbe essere vista come il record di un database ovvero una riga all'interno di una matrice (quest'ultima analogia bene si adatta alle tabelle delle istanze descritte nel diagramma delle tabelle ***Diag xxx***)

- L'oggetto di una classe è rappresentato dall'istanza della stessa.
- Un istanza viene generata **run time** con la funzione **new();**

2.5.2.3 Istanze statiche e dinamiche

Nell'ambito del presente firmware, dovute a tecniche di ottimizzazione del codice

- l'istanza è spezzata in due parti una parte dinamica (presente nella RAM) ed una statica (presente nella ROM)
- per ogni modulo il numero di istanze dinamiche (poniamo d), è inferiore al numero di sitanze statiche (poniamo s), quindi d<s; in particolare per le funzionalità implementate nel presente fw si è constato che è suffciente usare una sola istanza dinamica ovvero d=1 per i moduli uiPage, uiLbox e uiPar mentre le cose cambiano per il modulo uiVar in cui le istanze dinamiche possono essere anche maggiori di 1;
- un oggetto, essendo l'istanza spezzata in due e il numero di istanzae dinamiche inferiore al numero di istnaze statiche (in particolare come già detto ogni modulo gestisce una sola istanza dinamica), può trovarsi in due situazioni o stati
    - **Congelato** (quando all'istanza statica non è associata nessuna istanza dinamica, quindi l'oggetto nello stato di congelato è rappresentato dalla sola istanza statica) e
    - **Selezionato** (quando all'istanza statica è associata un'istanza dinamica, in questo stato l'oggetto è completo è può essere usato, ad esempio può essere visualizzato ed eventualmente attivato),
- un oggetto si considera A**ttivo** quando la tabella degli eventi del modulo di cui è istanza è presente nella tabella dei processi
- le proprietà identitarie dell'oggetto sono definite nell'istanza statica
- le istanze, sia statiche che dinamiche, non vengono create attraverso la funzione **new()** ma, a design time, definendo tabelle in ROM (per le istanze statiche) e variabili in RAM (per le istanze dinamiche)

Esempi: Proprietà statiche e dinamiche

2.5.2.4 Stati del Sistema interfaccia

Per descrivere lo stato del sistema interfaccia ci sono tre situazioni che vengono considerate

- **istanza della pagina attiva**: si vede quale oggetto del componente pagina è selezionato ovvero quale istanza statica è allineata con l'istanza dinamica; il passaggio di stato è stabilito a design time, gli eventi che generano questo passaggio di stato sono i tast right e left; per il passaggio di stato vedere il diagramma (a)
- **modulo/componente attivo**: si vede quale modulo/componente è attivo ovvero quale tabella eventi, di quale componente, sia presente nella matrice dei processi; per i passaggi di

stato vedere il diagramma (b)

- **stato degli oggetti**: come già detto un oggetto, a causa della scissione dell' istanza che lo rappresenta in statica e dinamica puo assumere lo stato di **Congelato** e **Selezionato**, oltre a questi due stati ce ne è un terzo, lo stato di **Attivo** che dice se la tabella eventi associata all'oggetto sia presente nella tabella dei processi; per i passaggi di stato vedere il diagramma (c) questo diagramma è riferito ai soli componenti Lbox, Par e Var, per il componente uiPage la gestione degli stati congelato, selezione e attivo sono gestiti in modo diverso, per questo si faccia riferimento al diagramma (d)

Osservazioni:

- *le macchine a stati descritte dai diagrammi a e b possono intendersi tra loro annidate; si può pensare il diagramma a annidato nel diagramma b e viceversa*
- *la macchina a stati che descrive lo stato degli oggetti (diagramma c) lavora invece in parallelo con le altre due macchine a stati;*
- *a voler essere rigorosi la macchina a stati descritta nel diagramma a è conseguenza delle altre due, ovvero può essere dedotta dalle altre due*
- *il componente uiVar non ha tabella eventi quindi i suoi oggetti non possono assumero lo stato di Attivo*
- *per il componente uiVar lo stato di Selezionato equivale a quello di Visualizzato, infatti tutte le istanze selezionate rappresentando valori variabili vengono aggiornate periodicamente attraverso una funzione Timer*

2.5.2.5 Passaggi di stato

I diagrammi di seguito riportati illustrano i cambi di stato; ogni cambio di stato è generato da un evento e viene compiuto per mezzo di una funzione ovvero conseguenza di un evento è la chiamata di una funzione che esegue il cambio di stato, di seguito si riporta la lista degli eventi e delle funzioni coinvolte nei diagrammi delle macchine a stati:

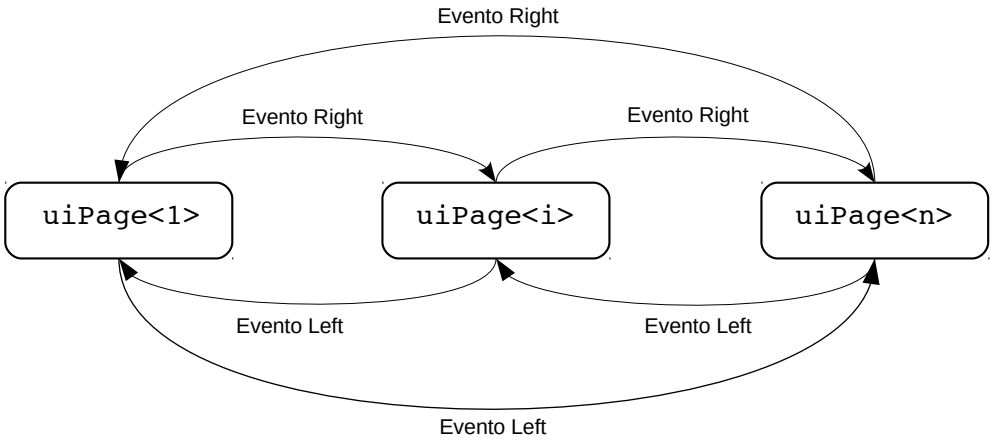
- **Eventi**: Enter, Left, Right, Up, Down
- **Funzioni**: seleziona_successivo, seleziona_precedente, incrementa_valore, decrementa_valore, conferma, annulla, vai_pagina_successiva, vai_pagina_precedente, passa_controllo (all'elemento selezionato), restituisci_controllo (al padre), EventOn, Select, Deselect

Osservazioni:

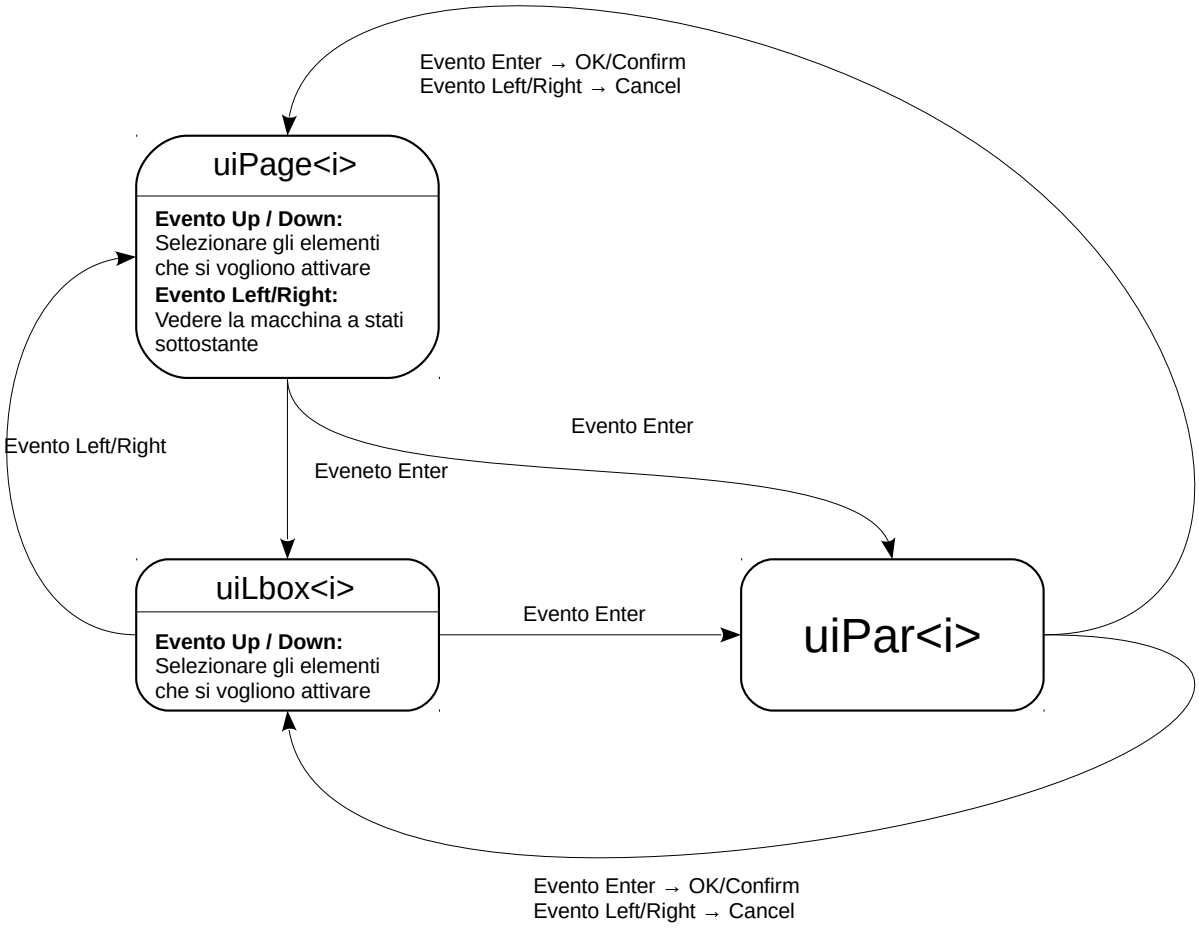
i passaggi di stato possono essere stabiliti a

- design time, è questo il caso del diagramma 2.2 in cui il passaggio da un istanza all'altra del componente uiPage è stabilita a design time ed in particolare è parte integrante dell'istanza statica del modulo uiPage

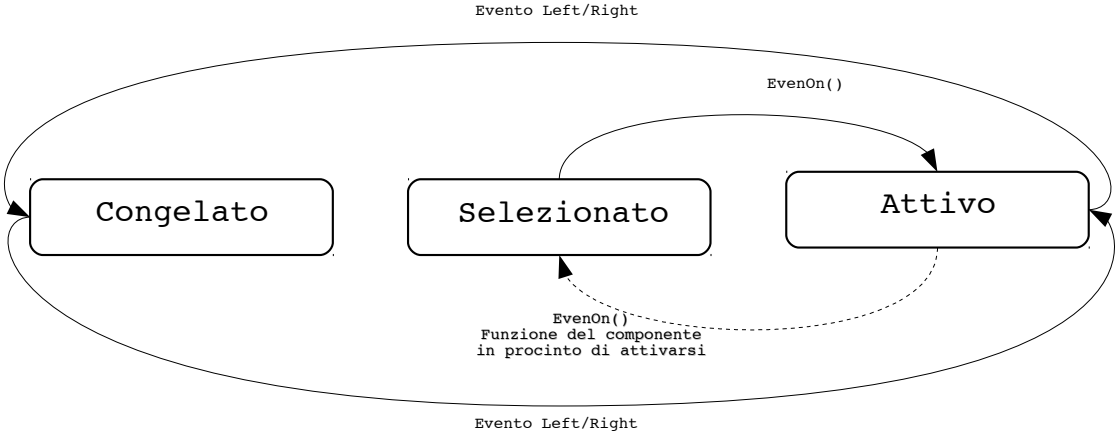
- run time, è questo il caso del diagramma 2.3 in cui il passaggio dello stato di attivo da un componente ad un altro viene stabilito a run-time attraverso la procedura di selezione



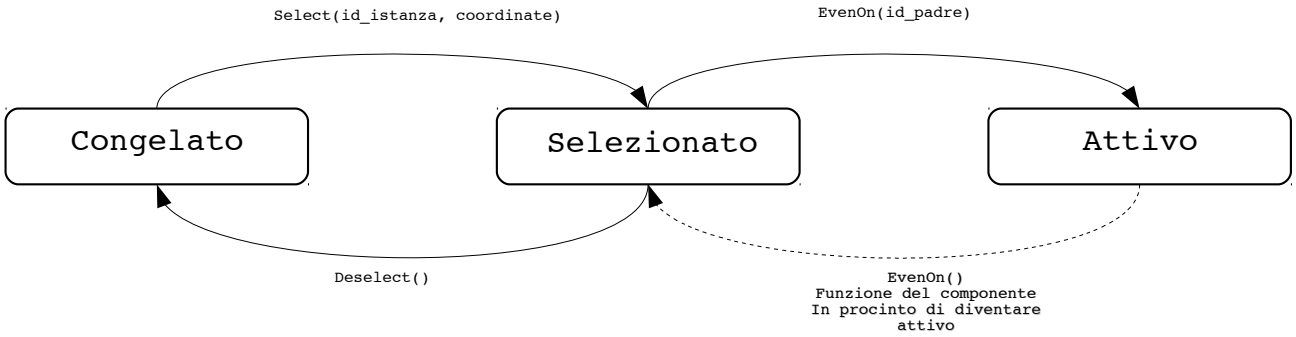
Diag. 6: Macchina a Stati – Passaggi di stato tra le istanze del modulo uiPage



Diag. 7: Macchina a Stati – Passaggi di stato tra i moduli



Diag. 8: Macchina a Stati – Passaggi di stato per gli oggetti del modulo uiPage



Diag. 9: Macchina a Stati – Passaggi di stato per gli oggetti dei moduli Lbox e Par

2.5.3 Diagramma Macchina a Stati - Visualizzazione lista di elementi

Il componente uiLbox ha il compito di visualizzare una lista di “**n**” elementi (che possono essere Parametri, Variabili ma anche altre tipologie) all'interno di una finestra il cui numero di righe “**m**” è inferiore ad “**n**” (**m < n**). Per svolgere questo compito è stata implementata una macchina a stati attraverso una tabella.

Il diagramma verra presentato in una futura versione della presente documentazione.

2.5.4 Diagramma Macchina a Stati - Lettura tasti

Anche la lettura dei tasti viene svolta implementando una macchina a stati, questa volta però usando la tecnica più semplice quella degli if-then-elese.

Il diagramma verra presentato in una futura versione della presente documentazione.

3. Lavorare con il codice

3.1 File .Def

3.2 Servizio “Timer”

3.3 Matrice dei Processi

Appendice

Cose Interessanti da inserire

Timer a word

Nel presente firmware sono considerati solo timer Byte, per gestire anche timer a word si dovranno duplicare le funzioni di init, SupplyEvent UpdateState e naturalmente la tabella eventi.

Variabili globali

Uso delle variabili globali e parola chiave “extern”

Uso di variabili globali per usare le variabili globali si usano gli extern, per una querstione di ordine del codice è utile definire gli extern in un unico file, il <prefisso>Def.c appunto, poi per un un po' di paranoia si puo voler suddividere la definzione per gruppii di file o più esattamente per moduli Per un ulteriore meticolosità si potrebbe poi definire le variabili stati in modo da renderle private Notare poi che per poter deinire le variabili in un unico file senza però preoccuparsi ogni volta di definire un extern nel file di utilizzo e ridefinirla nel file def Per fare questo si usa la seguente tecnica che sfrutta le define del preprocessore del compilatore C le variabili globali vanno definite in un file header e nel particolare del presente firmware in un file Glo inizio file pippoGlo.h

#ifdef DEF_C

#define _EXTERN

#elseif

#define _EXTERN extern

#endif

…

_EXTERN char var_pippo;

…

end file "pippoGlo.h"

inizio

file pippo1.c oppure pippo2.c (files in cui si usa la variabile "var_pippo")

#include "pippoGlo.h" (adesso la viariabile "var_pippo" viene definita come extern)

…

// generico uso della variabile

var_pippo = 2;

…

end file "pippo1.c" oppure "pippo2.c"

inizio file def.c

#define DEF_C (questa definizione deve precedere l'istruzione di include del file "pippoGlo.h")

#include "pippoGlo.h" (adesso la variabile "var_pippo" viene definita senza extern e quindi qui è il punto in cui viene realmente definita)

…

end file "def.c"

Per un modo alternativo di spiegare questo aspetto vedere "Sistemi Operativi" di Thannembaum editore Utet edizione 3a Pag. 70

Implementazione dipendenze indirette

IMPLEMENTAZIONE DIPENDENZE INDIRETTE USO DEL INIT, TIMER, EVENT-SUPPLY / EVENT-PTR-TABLE, Le dipendenze indirette hanno associato un nome.