INAF - Osservatorio
Astrofisico di Arcetri

ARCOSWEB

L'interfaccia per il controllo remoto di ARCOS

G. Comoretto[1] e F. Palagi[2] [1]

[1] INAF - Osservatorio Astrofisico di Arcetri, l.go E. Fermi 5, 50125 Firenze (Italy)
[2] INAF - IRA, Sezione di Firenze, l.go E. Fermi 5, 50125 Firenze (Italy)

[1] file:strumenti/arcos/remote/doc/arcosweb.tex. Versione del 25 ottobre 2006

Sommario *Si descrivono le modifiche apportate ad ADLB4ARC per renderlo controllabile da un calcolatore remoto. In sostanza si rimuove la parte relativa all'interfaccia grafica realizzata con le librerie LabWindows (DOS), lasciando il solo controllo tramite consolle. Il nuovo programma si chiama arcosrem e sarà eseguito sotto linux, in una shell remota.*

Notazioni:
Programma
Classe
Operazione
Attributo
`Valore`

Introduzione

Questo documento descrive lo sviluppo del programma **arcosweb**, che costituisce l'interfaccia utente del programma di controllo dello spettrometro ARCOS.

L'interfaccia e il programma di controllo sono basati su architettura *client/server* e possono essere eseguiti su nodi diversi della rete.

Lo sviluppo del programma inizia con una descrizione del porgramma ADLB4ARC, attualemnte in uso sotto sistema operativo DOS/Windows.

Successivamente si descrivono le modifiche necessarie per trasformarlo nel modulo server che riceve comandi da una socket e restituisce il risultato di una osservazione.

L'ultima parte è dedicata allo sviluppo di un'interfaccia grafica semplificata, in grado di eseguire una schedula, ma che non visualizza la configurazione del correlatore. Questa parte sarà sviluppata in un secondo momento.

Capitolo 1

Descrizione sintetica di ADLB4ARC

Si descrive il comportamento di ADLB4ARC nella fase di lettura e di definizione dei parametri di un'osservazione. La parte relativa all'esecuzione dell'osservazione (misura di ACF) rimane inalterata.

Il comportamento del programma è rappresentato attraverso diagrammi di attività, uno per ogni modulo.

1.1 Il modulo principale

Il modulo principale controlla il flusso ad alto livello (fig 1.1):

- inizializzazione: definizione dell'ambiente di lavoro
- loop centrale in cui si definiscono i parametri osservativi e si esegue la misura
- operazioni di chiusura della sessione

Inizializzazione

La fase di inizializzazione del programma comprende i seguenti passi:

1. Si salva il contesto grafico
2. Si apre il file di log.
3. Si assegnano i valori default ai parametri.
4. Si inizializza l'interfaccia utente in modo grafico. Il modo corrente viene salvato in una variabile statica.
5. Si apre il file toolbox di uscita. Si legge l'ultimo numero di scan dal file scan.num.
6. Si inzializza l'interfaccia grafica
7. Si inizializza la porta GPIB

8. Si apre il collegamento con l'antenna (Porta seriale).

Di questa fase non si riporta il diagramma di attività perchè è sostanzialmente sequenziale. La sequenza viene interrotta solo in caso di errore.

Definizione dei parametri osservativi

I parametri osservativi possono essere inseriti tramite un'interfaccia grafica, oppure possono essere specificati tramite comandi di testo. In questo caso la lettura può avvenire da tastiera o da un file (batch).

Il diagramma di attività del processo di lettura è rappresentato in fig. 1.1 dalla attività *get_para* ed è dettagliato nel diagramma di fig. 1.2. Le funzioni che collaborano alla sua realizzazione sono: *get_param()*, *getmenu()* e *getcommands()*.

La funzione *get_params()* passa il controllo alle funzioni di lettura in base al modo selezionato: *getcommands()* se in modo ST_CMDMODE o *getmenu()* se in modo ST_MENU. La lettura dei parametri è delegata rispettivamente alle funzioni *obs_pars()* (ST_CMDMODE) e *read_menu()* (ST_MENUMODE).

Entrambe le funzioni riportano il codice di stato per gestire l'inizio dell'osservazione (ST_START) o per terminare il programma (ST_EXIT).

Operazioni di chiusura

In questa fase si aggiorna il file scan.num e si chiude il file di output (acf).

1.2 La libreria cmdint.a

Il programma usa la libreria cmdint.a per gestire la lettura e decodifica dei parametri. Durante il suo trasferimento sotto linux, si è evidenziata una dipendenza dalle strutture dati usate da Arcos, attraverso un riferimento alla funzione *sendmk3()* nel eseguire i comandi Field System. Per eliminare questa dipendenza si sposta la chiamata alla funzione *sendmk3()* nella funzione *getcommands()*, mentre il comando snap del field system viene memorizzato nella variabile val[0] della struttura kommando.

Nel caso di comandi (attivià) la cui esecuzione richiede un intervallo di tempo non trascurabile è necessario leggere i messaggi di log per identificare la fine dell'attività, come accade durante il puntamento di una sorgente.

Nel caso di una calibrazione (comando snap **onoff**) la stringa che identifica il termine della procedura contiene le due parole: onoff e source.

E' necessario prevedere il caso in cui si verifichi un errore e il programma onoff sia terminato in modo anomalo per evitare un'attesa infinita.

I comandi Field System si distinguono in due tipi:

1. FS_ACTION comandi la cui esecuzione può considerarsi istantanea
2. FS_CALIB si tratta di una calibrazione che richiede una verifica specifica per determinare la conclusione dell'attività. In questo caso la stringa scritta da onoff deve contenere le due parole onoff e source.

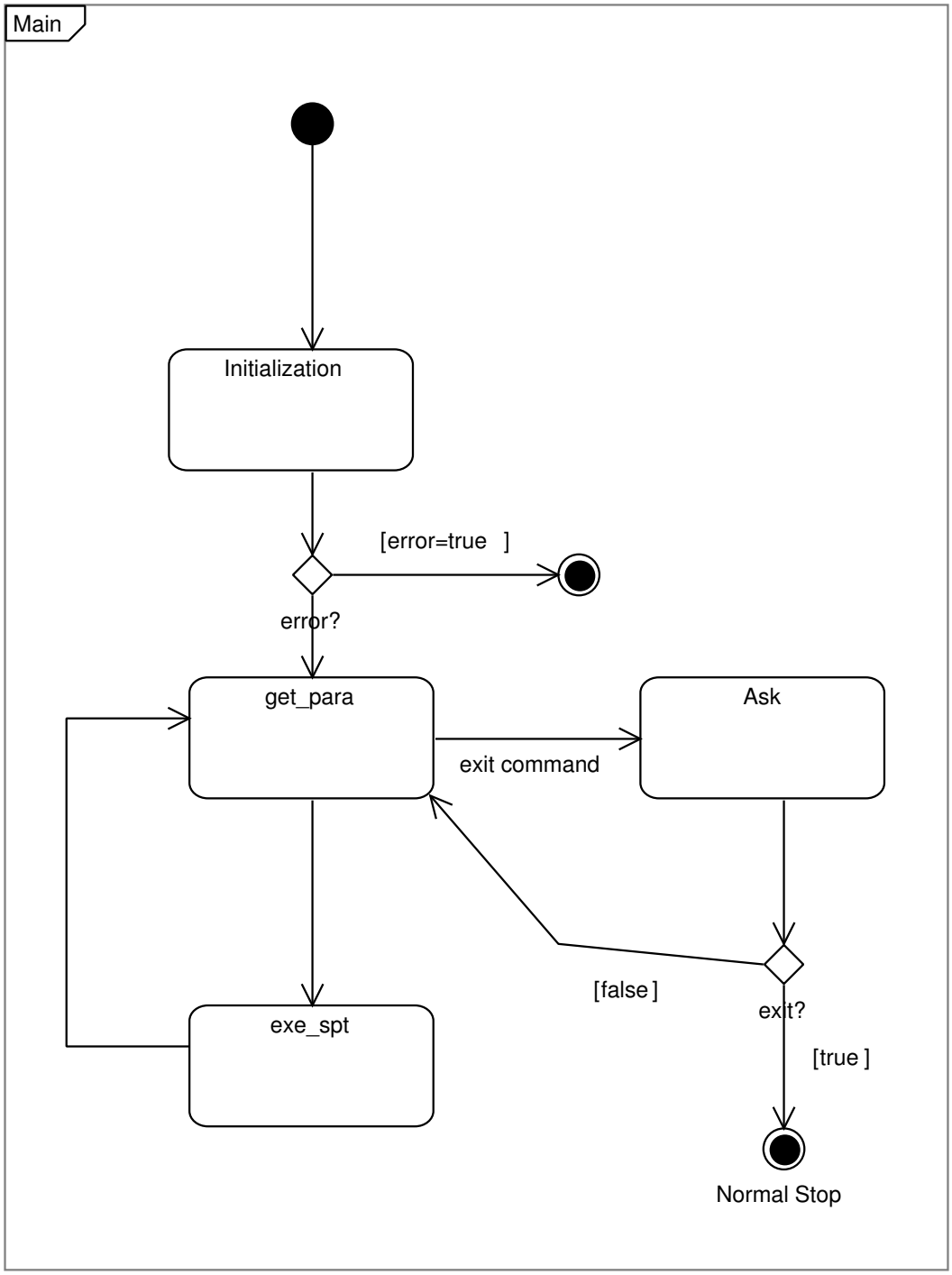

Figura 1.1: ADLB4ARC: diagramma di attività del modulo principale.

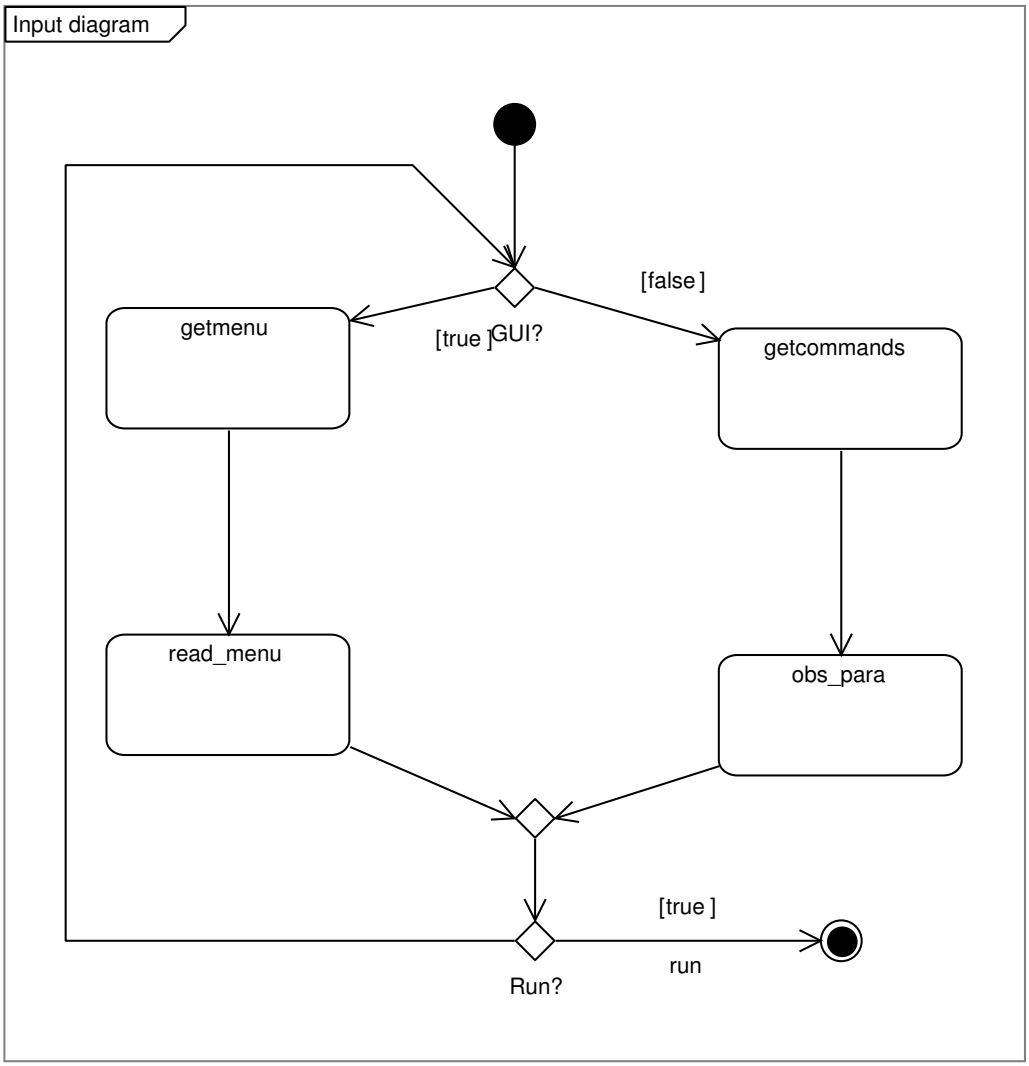

Figura 1.2: ADLB4ARC: diagramma di attività per la fase di definizione dei parametri osservativi.

Queste due macro sono usate nella funzione *getcommands()* che gestisce i comandi di controllo tramite la variabile globale status. Poichè questi comandi saranno usati durante l'esecuzione di schedule possono essere eseguiti direttamente da *getcommands()*.

In pratica:

1. Si definiscono le variabili FS_CALIB e FS_SNAP in cmdint.h.
2. Si dichiarano le funzioni *fs_calib()* e *fs_snap()* in fieldsystem.h.
3. In *get_param()* si aggiungono due stati ST_CALIB e ST_SNAP, che richiamano le due funzioni precedenti. Questi due stati sono definiti in *getcommands()* [per ora] quando la funzione *obs_pars()* ritorna i valori FS_CALIB e FS_SNAP.
4. In *obs_pars()* (obs_par.c) si aggiungono i due *case* corrispondenti.
5. Nel *case* FS_CALIB si definise lo stato ST_CALIB
6. Nel *case* FS_SNAP si definisce lo stato ST_SNAP.
7. La funzione *fs_calib()* (file fs_commands.c) invia il comando di calibrazione specificato tramite *sendmk3()*. Inizia un loop di attesa costituito dalla lettura dei messaggi di log e dalla ricerca delle due parole onoff e source, che indicano la fine dell'attività di calibrazione. Terminata la calibrazione la funzione ritorna con ST_CMDMODE in modo che il controllo torna a *getcommands()* per la lettura del comando successivo.
8. La funzione *fs_snap()* (file fs_commands.c) invia il comando snap specificato, legge il messaggio di risposta, lo scrive nel file di log e ritorna con ST_CMDMODE.

Il diagramma di stato della funzione *get_param()* è rappresentato in figura 3.

Il rombo posto al centro del diagramma rappresenta il switch che gestisce il comportamento della funzione in base alla variabile status.

1.3 Interfaccia Utente

L'interfaccia utente è un programma *client* del programma di controllo del correlatore. Essa è in grado di attivare le funzioni del programma server sia attraverso l'invio dei comandi che ne costituiscono il linguaggio, sia attivando direttamente le funzioni che formano la sua interfaccia esterna.

Nei casi in cui il programma server ha necessità di comunicare un evento all'interfaccia utente, esso utilizza dei segnali o una call-back forniti dall'interfaccia utente. Ad esempio, questo caso si verifica al termine di una misura quando il server segnala l'evento in modo che l'interfacca possa iniziare una nuova intergrazione e lo scambio dei dati.

Si descrive una GUI semplificata che fornisce le seguenti funzioni:

1. Selezione di un file di schedula
   (a) Visualizzazione elenco delle sorgenti

   (b) Selezione della sorgente da osservare per prima

2. Sequenza delle osservazioni

   (a) Visualizzazione della sorgente osservata
   (b) Lettura della linea di comando
   (c) Visualizzazione nella finestra di log
   (d) Aggiornamento tempo di integrazione
   (e) Interruzione schedula (Halt).
   (f) Interruzione immediata (Abort)

3. Inserimento comando di testo

4. Visualizzazione comandi eseguiti (log)

5. In fase di inizializzazione, apertura del file toolbox di uscita.

6. Inizializzazione del telescopio.

L'aspetto grafico dell'interfaccia è mostrato in fig.1.3. Il pannello contiene i seguenti elementi:

- elenco sorgenti
- Command input
- finestra di log
- Menubar: File/open/exit
- Buttons: HALT, BREAK
- Display: tempo di integrazione, cicli residui

1.4 Composizione della GUI

La GUI è basata sulle librerie QT, ed è progettata usando il QT Designer. Le routines di ARCOS dovrebbero essere compilate e raccolte in una libreria da linkare con il programma sviluppato da Qt Designer.

Le classi non direttamente coinvolte nello sviluppo dell'interfaccia e aggiunte dallo sviluppatore sono progettate e generate in Poseidon 4.1 in modo da mantenere allineati la loro documentazione e codice. Ad esempio la classe *ArcosProject* è inclusa nel progetto **arcosweb** sotto Poseidon 4.1. I relativi files .h e .cpp sono aggiunti al progetto interfaccia di QTDesigner.

Infine il presente documento tiene traccia dell'evoluzione del progetto a livello più elevato.

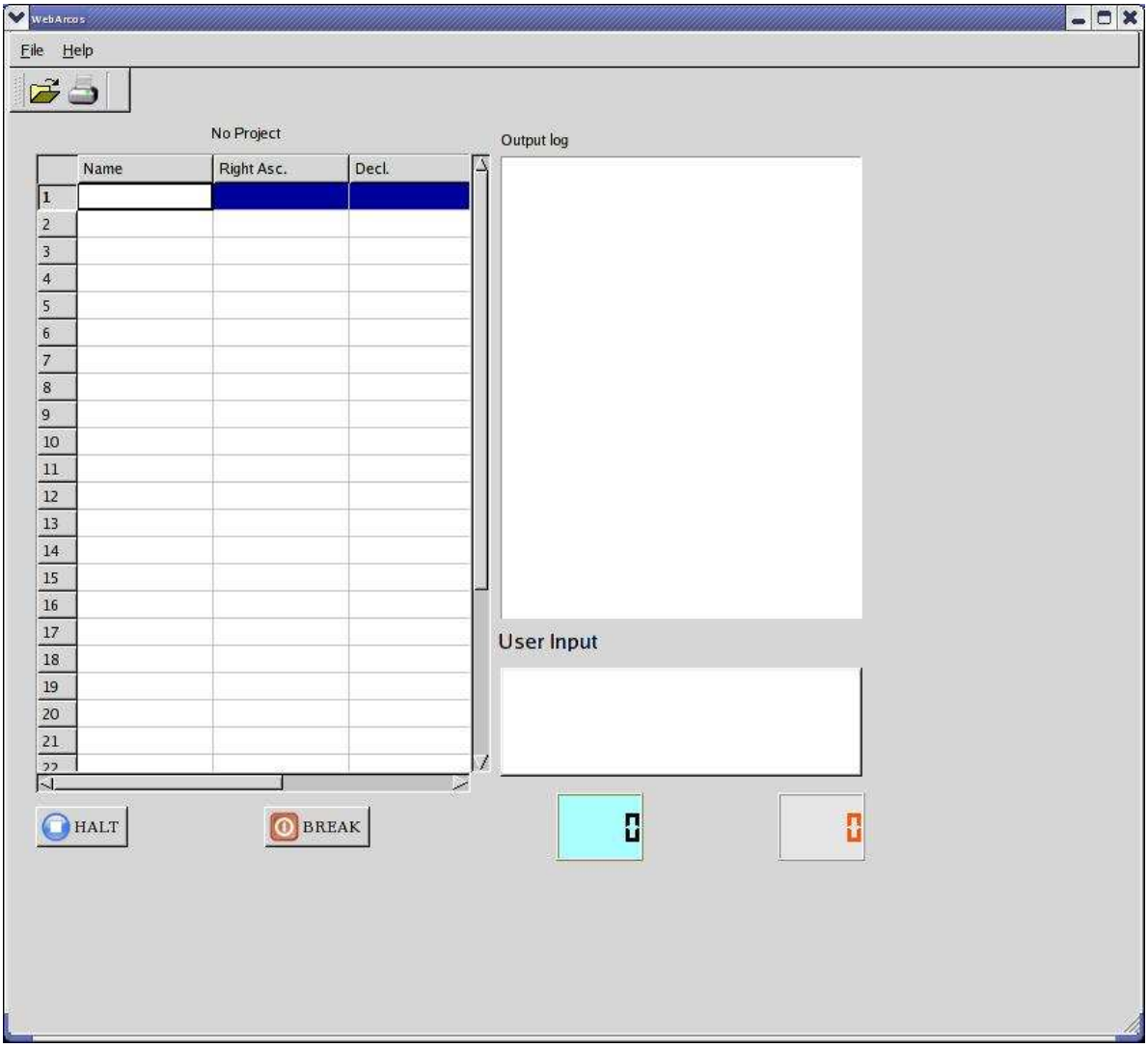

Figura 1.3: ARCOSWEB: l'interfaccia grafica composta dalla tabella delle sorgenti del progetto osservativo, una finestra di input e una di output. I bottoni in basso sono usati per interrompere l'osservazione.

1.4.1 Menubar

Qt Designer fornisce una finestra standard dotata di menubar con il menu File. Si tolgono le voci New e Save che in questa applicazione non hanno senso.

1.4.2 Source List

Si usa una widget QTable in cui si elencano il nome e le coordinate delle sorgenti (compresa l'epoca e la VLSR). In prima colonna si mette una casella di check per indicare le sorgenti osservate. Deve essere possibile resettare questa selezione.

1.4.3 Output log

Il programma scrive i comandi eseguiti e il risultato delle interrogazioni dell'utente. E' una widget di classe QEdit usata in LogText mode.

1.4.4 User Input

Finestra usata per accettare comandi dall'utente. E' una widget di classe Qedit.

1.4.5 HALT e BREAK

Due PushButtons usati per interrompere il programma osservativo o l'osservazion in atto (interruzione immediata).

1.4.6 Cycles e Time

Due contatori che indicano il numero di cicli residui e il tempo di integrazione residuo. Sono due QLCDNumber widgets.

1.5 Struttura delle classi

Il diagramma delle classi della GUI è rappresentato in fig. 1.4. La classe MainForm rappresenta il cuore della struttura. Essa usa la classe ArcosProject per ottenere la descrizione del progetto.

La classe *Schedule* è associata al file di schedula del progetto e fornisce le informazioni sulle sorgenti che saranno inserite nella tabella delle sorgenti sourceTable. La classe *Configuration* è associata al file di configurazione. Entrambe sono derivate dalla classe *InputStream* per cui possono essere assegnate alla variabile curr_input di CommandReader. In questo modo i comandi che esse contengono sono eseguiti dalla classe Vocabulary in cui MainForm inserisce i propri comandi.

In realtà MainForm non usa direttmente CommandReader, ma na classe derivata che modifica l'operazione command() in modo da riconoscere le etichette presenti nel file di schedula.

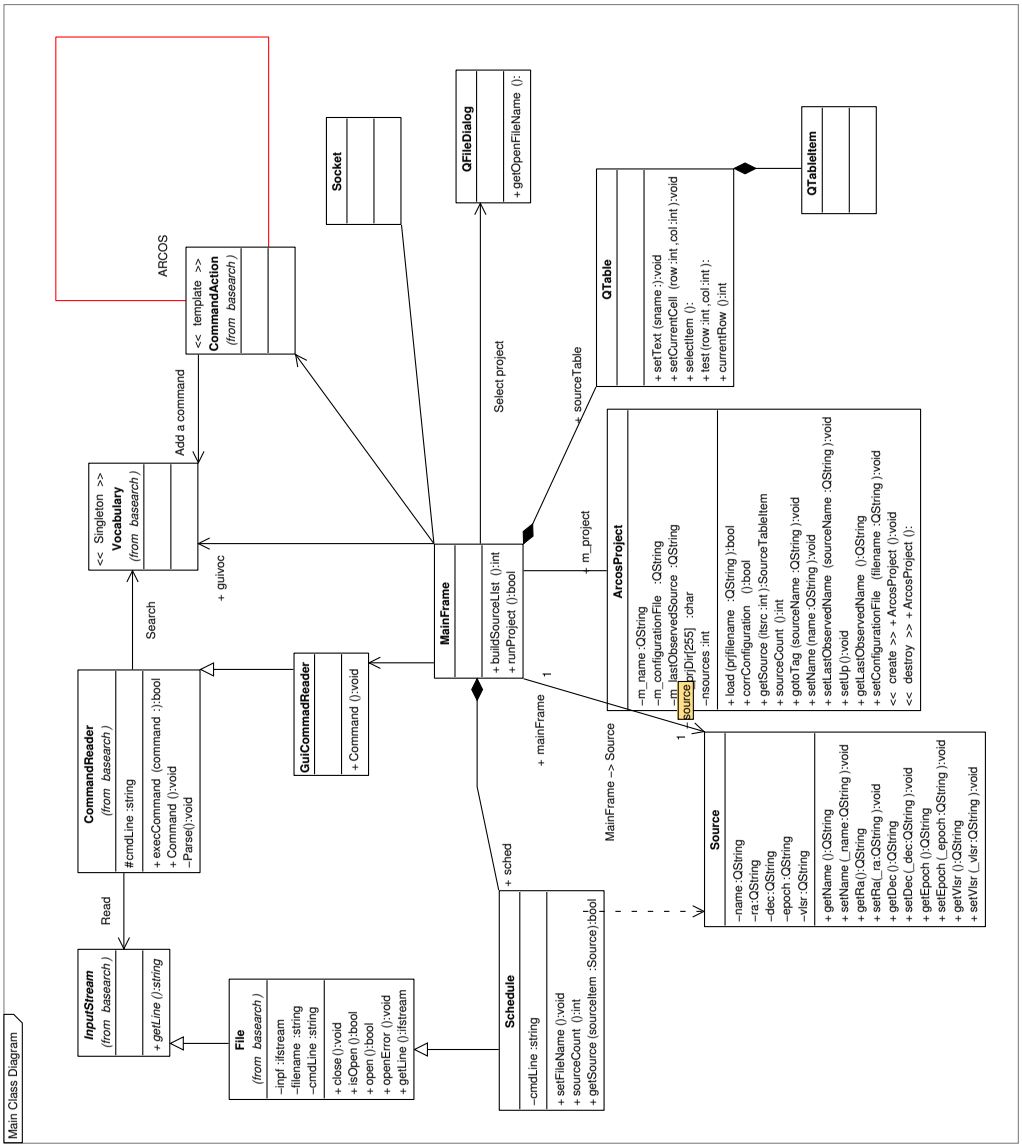

Figura 1.4: Diagramma delle classi della GUI. La classe Qt *MainForm* è il cuore della struttura.

1.6 Esecuzione di un progetto osservativo

Ogni progetto è descritto in un file `.prj` che contiene le seguenti informazioni:

1. Nome del progetto
2. Nome del file di configurazione.
3. Nome del file di schedula.
4. Nome dell'ultima sorgente osservata.

Nella gestione dei progetti il programma usano 2 variabili di ambiente per localizzare i file usati:

1. ARCOS_PRJ_DIR che contiene i files di descrizione dei progetti osservativi e le cartelle dei dati osservativi.
2. ARCOS_CONFIG_DIR che contiene il nome della directory dove sono memorizzati i files di configurazione.

La procedura di istallazione del progetto crea una cartella con il suo nome e copia al suo interno il file di schedula. Durante l'esecuzione del programma qui saranno memorizzati i risultati delle osservazioni, il file di *log* e altri files prodotti nell'ambito del progetto.

Ogni progetto osservativo è descritto da un oggetto di classe ArcosProject i cui attributi corrispondono alle informazioni descritte nell'elenco precedente, mentre le operazioni previste sono:

1. Selezione e Apertura del file di progetto
2. Inizializzazione della schedula:
   (a) Intestazione Nome progetto nella lista sorgenti
   (b) Creazione lista sorgenti
   (c) Posizionamento dell'evidenziatore sull'ultima sorgente osservata + 1
3. Configurazione del correlatore
4. Scelta della sorgente ed esecuzione della schedula

1.6.1 Selezione e Apertura di un progetto

Il diagramma di attività di questa funzione è rappresentato in figura 1.5. L'apertura di un progetto osservativo si ottiene dal Menù File→Open collegando il segnale *activated* di fileOpenAction con la slot *fileOpen* di MainForm.

MainForm dichiara una variabile membro m_project di classe ArcosProject e la slot fileOpen mostra i file di programma presenti su disco e permette all'utente di scegliere il progetto desiderato. In ambiente Qt questa funzione è realizzata dalla funzione *QfileDialog::getOpenFileName()* (fig. 1.6).

Il progetto scelto viene aperto e si assegnano i valori alle variabili membro della classe *ArcosProject* utilizzando l'operazione *ArcosProject.load()*.

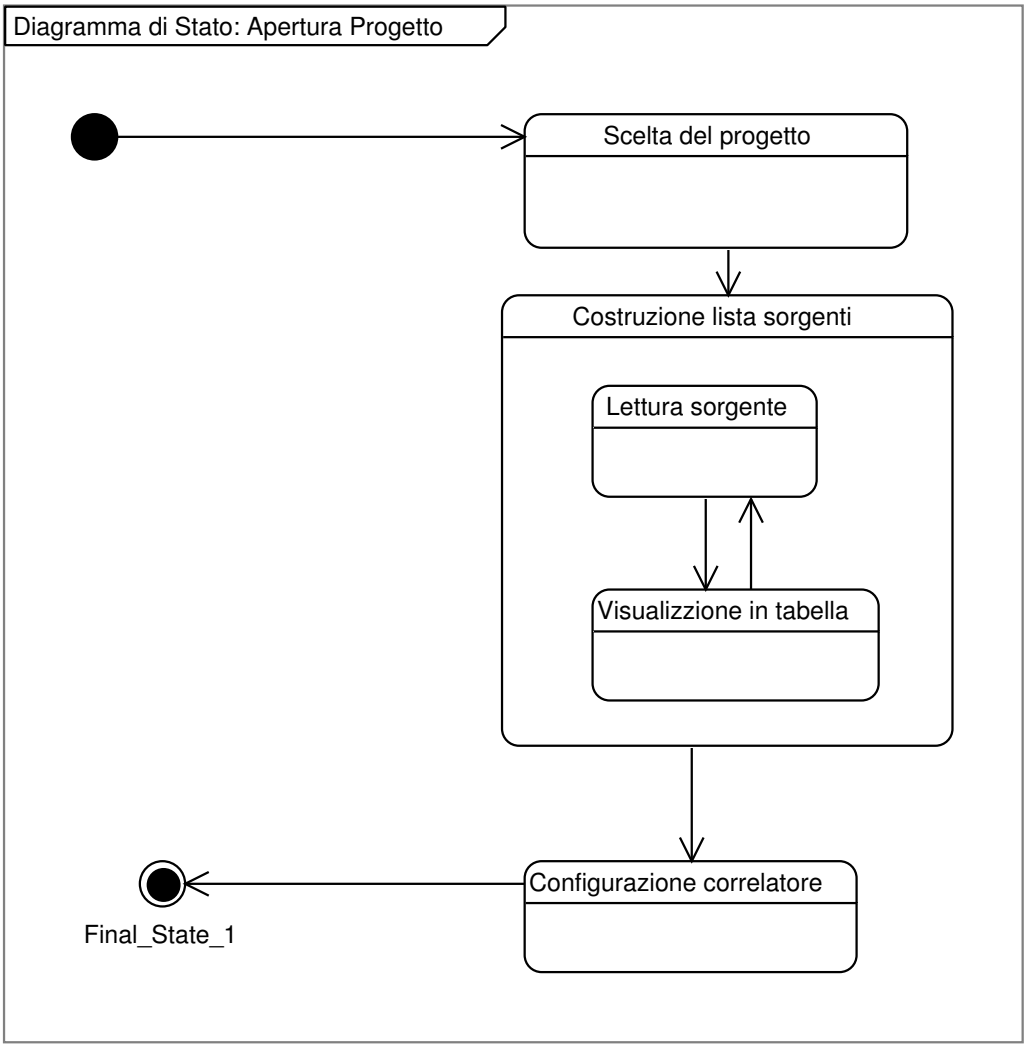

Figura 1.5: Diagramma di stato dell'apertura di un progetto osservativo

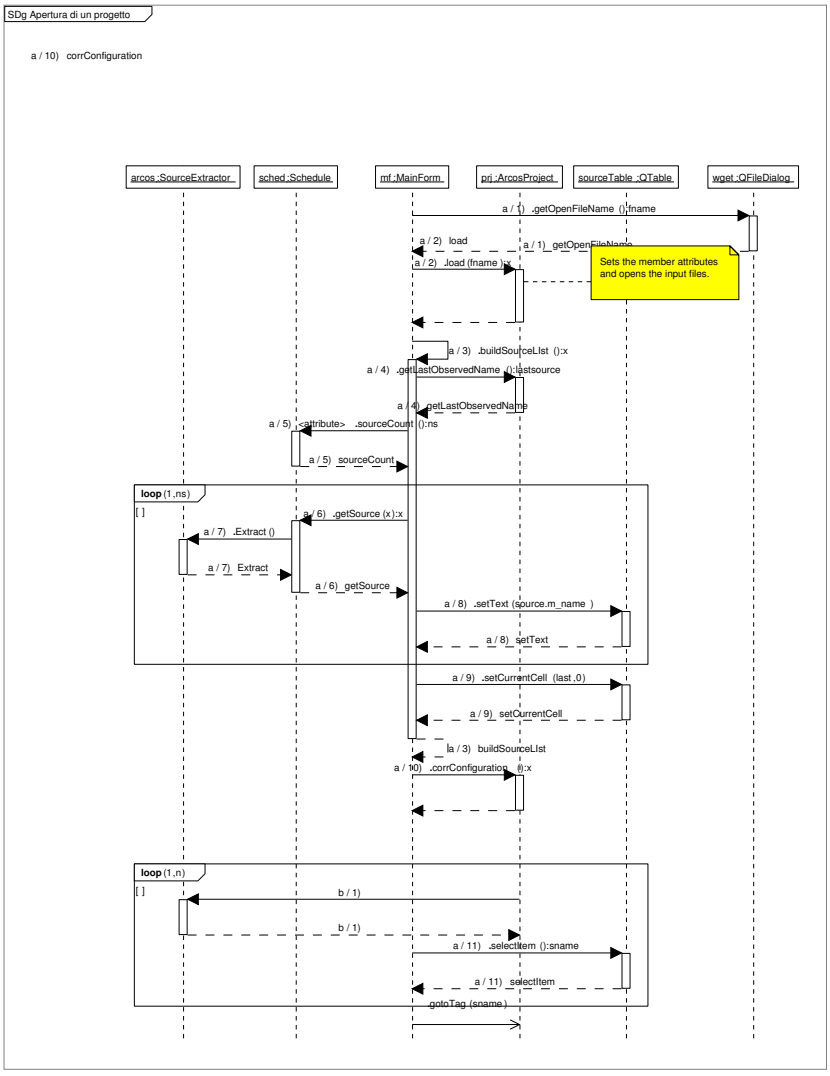

Figura 1.6: Diagramma di sequenza dell'apertura di un progetto osservativo

1.6.2 Lista delle sorgenti

La lista delle sorgenti è ricavata dal file di schedula interpretando alcuni comandi in modo diverso dalla fase di esecuzione della schedula stessa. Per questo motivo la classe Schedule tiene traccia del proprio stato e invoca i metodi appropriati. Nello stato LISTA i comandi aggiornano la tabella delle sorgenti mentre nello stato OSSERVAZIONE assegnano i parametri al programma di controllo di arcos.

I comandi utilizzati sono:

source Descrive il nome della sorgente

sra50 Ascensione Retta della sorgente (B 1950) [1]

sdec50 Declinazione della sorgente (B 1950)

vlsr Componente radiale della velocità della sorgente rispetto al LSR.

La sequenza di operazioni per la costruzione della lista delle sorgenti è la seguente:

- La classe MainForm costruisce l'elenco delle sorgenti con l'operazione *buildSourceList*.
- MainForm richiede a Schedule il numero delle sorgenti presenti nel file di schedula.
- MainForm richiede a Schedule la descrizione di ciascuna sorgente presente nel file di schedula per mezzo della classe SourceTableItem. La classe MainForm, chiama l'operazione *getSource()* che definisce lo stato LISTA per la classe Schedule.

  L'assegnazione dei valori agli elementi della classe SourceTableItem è eseguita dai corrispontenti metodi di accesso alle variabili membro della classe.

Il meccanismo di lettura ed esecuzione dei comandi attraverso la classe CommandReader funziona correttamente e quindi è preferibile non modificarlo, per esempio realizzando la possibilità di keywords duplicate. Per questo i due metodi che compongono l'interfaccia pubblica di Schedule (*getSource()* e *exec()*) definiranno lo stato del programma prima di iniziare la lettura del file e l'esecuzione dei comandi attraverso CommandReader.

I comandi corrispondenti ad ognuna di queste keywords sono creati dal costruttore della classe Schedule secondo lo schema descritto in **basearch**, associando un metodo della classe Schedule che, a sua volta, chiama il metodo richiesto dallo stato corrente, come descritto nella tabella 1.1.

Come esempio, lo scenario che segue descrive l'esecuzione del comando source nei due casi (fig. 1.7).

- Si apre il file di schedula. Schedule è in stato lista.
- Si costruisce la tabella delle sorgenti. Il comando source provoca l'esecuzione del metodo *Schedule::source()* che chiama il metodo `SourceTableItem::setName()`
- L'utente sceglie la sorgente W43 (la terza dell'elenco). Si costruisce l'etichetta.

[1] Per il momento l'epoca al 1950. **In futuro molto prossimo occorrerà accettare coordinate J2000.**

Stato	Keyword	Classe	Metodo
lista	source	SourceTableItem	setName
	ra50	*idem*	setRa
	dec50	*idem*	setDec
	vlsr	*idem*	setVlsr
	run	SourceExtractor	extract
osservazione	source	Arcos	srcName
	ra50	*idem*	ra50
	dec50	*idem*	dec50
	vlsr	*idem*	vlsr
	run	*idem*	run

Tabella 1.1: Corrispondenza fra keyword, classe e metodo per le realizzazione dei comandi elencati. Ad oggi la variabile m_epoch è definita 1950.

- MainForm chiama il metodo *Schedule:exec()*
- Schedule pone lo stato in osservazione.
- Schedule richiede la lettura ed esecuzione dei comandi a CommandReader.
- CommandReader trova la keyword source e chiama *Schedule:source()*
- Schedule trova lo stato osservazione e chiama il metodo *Arcos:setSource()*.

Se la sorgente da inserire coincide con l'ultima osservata si memorizzano le coordinate della cella corrispondente in modo da attivare la cella della riga successiva (prima sorgente NON osservata). Per attivare la selezione di un'intera riga della tabella si è assegnato il valore `SingleRow` alla proprietà *selectionMode*.

1.6.3 Configurazione del correlatore

Il file di configurazione è descritto dalla classe *Configuration* derivata dalla classe *File*.

La configurazione del correlatore è eseguita dall'operazione *corrConfiguration()*.

Il file viene aperto e le righe di comando sono lette e inviate al correlatore come nel caso dell'esecuzione di un'osservazione. La sola differenza è che non è ammessa l'istruzione `run`.

1.6.4 Esecuzione di un progetto osservativo

Una volta aperto un progetto il cursore della tabella delle sorgenti è posizionato sulla prima sorgente da osservare. L'osservatore seleziona la sorgente e l'interfaccia ricerca l'etichetta corrispondente e invia i comandi contenuti nella schedula. Se l'etichetta non viene trovata si emette un messaggio di errore.

L'operazione che implementa questa funzione appartiene alla classe MainForm e si chiama *sourceTable_ doubleClicked()*. Quando viene scelta la sorgente la classe MainForm chiama chiama l'operazione *exec* della classe Schedule che definisce lo stato OSSERVAZIONE.

Essa è collegata al segnale doubleclick. La funzione costruisce l'etichetta da ricercare aggiungendo il carattere : al nome della sorgente scelta, esegue la prima parte della schedula (preambolo) fino alla prima etichetta. Successivamente legge il nome della sorgente e lo usa per localizzare l'etichetta (vecchio comando Input goto). Infine legge ed esegue i comandi contenuti nel file di schedula.

La lettura dei comandi per ARCOS contenuti sia nei files di configurazione che nei files di schedula è demandata alla classe CommandReader. Questa classe separa la parola chiave (prima parola nella riga comando) dagli argomenti (parte rimanente). La decodifica degli argomenti è demandata al comando inserito nel vocabolario. Questa parte è completamente sviluppata nella libreria **basearch**.

1.7 L'interfaccia (API) di Arcos

La GUI del programma è essenzialmente un programma che, sulla base dei comandi presenti in un file di schedula o di configurazione oppure inseriti dalla finestra di *input*, chiama le funzioni *pubbliche* della classe Arcos, che rappresenta il correlatore all'interno del programma.

L'esecuzione di un comando può essere fatta in due modi:

1. Inviando il comando acquisito al programma attuale, sfrondato dalle operazioni di interazione con l'osservatore. Il comando viene decodificato dalle routines della librearia cmdint.

2. Associando al comando metodi specifici che interagiscono con gli attributi di Arcos. [2]

Da un'analisi del codice attuale risulta che il primo metodo richiede un tempo di realizzazione minore e quindi viene scelto per la prima versione. Tuttavia, per migliorare la manutenzione del programma la seconda versione sarà realizzata seguendo il secondo approccio.

Per orientare il codice già da ora verso la seconda versione, ciascun comando è associato ad metodi diversi di Arcos, che tuttavia si limitano a chiamare la funzione getparam opportunamente modificata.

Fa eccezione il comando run che chiama direttamente la funzione exe_spt().

I paragrafi seguenti descrivono le modifiche apportate ad alcune funzioni della libreria cmdint per eliminare ogni interazione sia con la vecchia interfaccia grafica sia con il modo testo.

Le funzioni modificate insieme ad un programma di test sono nella directory ../remote/source/arcoslib.

1.7.1 La funzione get_param

Si elimina ogni riferimento alla vecchia interfaccia grafica (funzione *get_menu()*). Alla funzione si aggiunge il parametro *strcommand* che contiene la riga di comando da decodificare ed eseguire.

La funzione chiama *getcommand()* passando la stringa di comando.

[2] Questa soluzione è più lineare in quanto evita di usare il meccanismo complesso della librearia cmdint.

1.7.2 La funzione getcommand

getcommands() chiama la funzione *obs_pars()* e controlla il flusso del programma in base al suo valore di ritorno. In pratica definisce lo stato del programma che viene riportato a *get_param()*. Si aggiunge il parametro *strcommand* da passare a *obs_pars()* e si introduce una nuova condizione di stato (ST_DONE) per indicare che il comando è stato eseguito correttamente.

1.7.3 La funzione obs_pars

Questa funzione gestisce la lettura del comando da tastiera o file. Nella versione modificata il comando viene passato come parametro e quindi occorre togliere le istruzioni che gestiscono il prompt e la lettura del comando (*read_line()*).

Le funzioni di lettura e decodifica usano un buffer statico dove viene memorizzato il comando acquisito, quindi la chiamata a *read_line()* viene sostituita da una funzione che copia il comando ricevuto come parametro nel buffer statico tramite la funzione *store_line()*.

Occorre anche eliminare le istruzioni che gestiscono i casi di EOF, commento e comando di sistema che sono trattati a livello di interfaccia Qt.

Dopo che i comandi che definiscono i parametri dell'osservazione sono stati eseguiti il metodo ritorna lo stato ST_DONE.

Queste modifiche sono state provate con il programma `testlib` (test.c).

1.8 Inizializzazione del programma

Queste operazioni avvengono all'interno della funzione init di MainForm. L'elenco delle operazioni da eseguire è il seguente:

- Creazione dell'oggetto guivoc (classe Vocabulary), degli oggetti sched (classe Schedule) e m_project (classe ArcosProject).
- Inizializzazione del correlatore.
  - Lettura dell'ultimo numero di scan.
  - Apertura del file di uscita (medi0000.acf).
  - Verifica del collegamento con il telescopio.
  - Apertura collegamento HPIB

1.9 Generazione del programma

Al progetto Qt dell'interfaccia si aggiungono alcune classi che non sono derivate dalle classi Qt:

ArcosProject Descrive un progetto osservativo per lo apettrometro Arcos, specificando il titolo, i files di configurazione e schedula, e l'ultima sorgente osservata.

SourceTableItem Contiene la descrizione della sorgente da mostrare nella tabella della GUI.

Schedule Descrive il file di schedula e ne estrae le sorgenti. Inserisce i comandi necessari nel vocabolario della GUI.

SourceExtractor E' derivata da CommandReader e filtra i comandi presenti nella schedula.

Queste classi sono progettate e generate in ambiente Poseidon. I relativi files .cpp e .h sono aggiunti al progetto Qt **arcosweb.pro**. Il progetto Poseidon riporta, anche alcune classi Qt coinvolte nello sviluppo dell'interfaccia e una classe fittizia Correlator per descrivere il contesto della GUI.

La classe Correlator rappresenta il programma attuale decurtato delle funzioni che interagiscono con l'utente. Il programma attuale sarà trasformato in una libreria da linkare al progetto Qt arcosweb0.pro.

Nel progetto arcosweb0.pro occorre definire le directories in cui cercare include files e librerie. Per questa operazione occorre aprire il menù Projetc←Project Settings... e inserire i valori nella tab C++.

Questo progetto usa i seguenti include paths:

- /export/home/palagi/palagi/astronomia/c
- /export/home/palagi/palagi/strumenti/basearch/source

  A causa di una diversa codifica dei nomi dei files che realizzano le classi nei due CASE Qt Designer (caratteri minuscoli) e Poseidon (nome file = nome classe) è utile usare relazioni unidirezionali da una classe Qt verso una classe generata da Poseidon per evitare inclusioni di files .h che non verrebero trovati.

  Una soluzione a questo problema consiste nell'utilizzare caratteri minuscoli per le classi Qt include dalle classi Poseidon, violando la regola sui nomi delle classi.

I comandi da usare per la generazione del programma sono i seguenti:

- In Poseidon si generano il files delle classi elencate nel paragrafo precedente.
- make aggiorna il Makefile sulla base delle modifiche apportate al progetto arcosweb0.pro, compila e esegue il link del programma.
- gdb si usa per il debug del programma.

In data 4/7/06 il programma completa l'elenco delle sorgenti nella tabella dell'interfaccia grafica.

1.10 Uso del programma

Il programma si lancia con il comando `arcosweb0`. Compare una finestra con il menu File in alto a sinistra secondo lo stile Windows. Dal menu File si sceglie la voce open e compare una nuova finestra di dialogo che mostra il files .prj presenti nella directory corrente. Si sceglie il file, in cui sono definiti i file di configurazione e il file di schedula. Dalla lettura del file di schedula si riempie la tabella delle sorgenti da osservare.

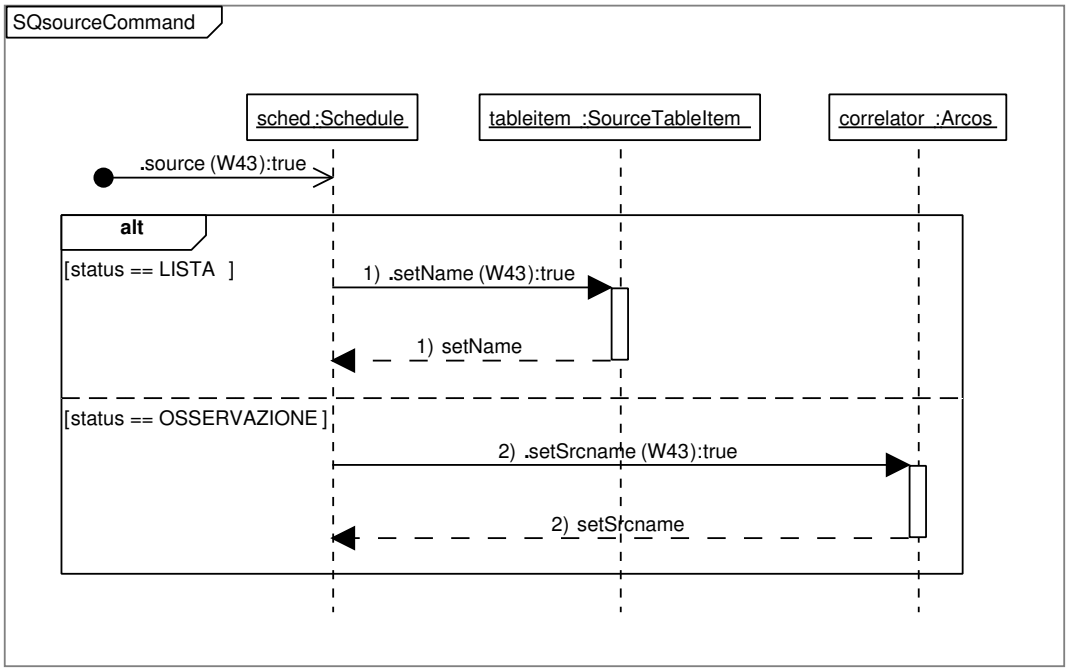

Figura 1.7: Scenario di esecuzione del comando source nei due stati lista e osservazione

Bibliografia

[1]

Indice