INAF-OAA Gruppo Strumentazione Infrarossa

Progetto Giano

gbridge-balor: il software middle-ware di Giano

Versione 1.0

E.Giani, C.Baffa

Rapporto Interno di Arcetri N° **7/2010**
Firenze, 12/2010

Sommario

Questo documento descrive gbridge*, elemento del sistema di acquisizione dello spettrometro infrarosso GIANO.*

Il sistema di acquisizione dati è realizzato con un sistema composta da moduli hardware e moduli software. La componente hardware comprende l'elettronica di acquisizione ed un sistema embedded connesso attraverso ethernet alla macchina sulla quale viene eseguito il sistema di acquisizione.

Quest'ultimo è composto di due applicazioni software distinte, una di alto livello a cui accede l'osservatore, l'altra è il software middleware gbridge*, oggetto del presente documento.*

Il software gbridge *è stato sviluppato in C su sistema Operativo Linux ed è attualmente installato e in funzione su un PC IBM compatibile, con distribuzione Ubuntu 8.04.*

Il programma è stato inoltre compilato ed eseguito su una macchina Linux con Ubuntu OS a 64-bit e su Ubuntu 10.04. La versione definitiva è prevista girare sotto quest'ultima versione di Linux.

Durante il completamento di questo rapporto interno, per mutate necessità del progetto Giano*, al programma* gbridge *sono state aggiunte alcune parti, ancora in sviluppo, e la nuova* version*e ha ricevuto il code-name* gbridge-balor*. È stato quindi deciso di lasciare nel presente rapporto interno la descrizione delle funzionalità correnti e rinviare la documentazione delle nuove caratteristiche ad un testo successivo.*

Figura 1: *Immagine fantastica del demone Balor, dal gioco Dungeons & Dragons[5]*

Capitolo 1

L'applicazione gbridge-balor

gbridge-balor è un'applicazione che funziona da interfaccia di medio livello tra l'applicazione utente Giano GUI e il programma di controllo del sistema embedded denominato server104[3], nascondendo i dettagli dell'implementazione delle funzionalità del sistema di acquisizione.
L'applicazione gbridge-balor funzione da "`ponte`" ed esegue funzioni di controllo e monitoraggio della comunicazione tra i due processi.
Il software gbridge-balor è stato sviluppato in linguaggio C su Sistema Operativo Linux. La sua compilazione richiede la presenza delle librerie standard C e delle librerie esterne:

- **cftisio** per la gestione dei file FITS.
- XPA (*X Public Access*) che fornisce gli strumenti per implementare un sistema di comunicazione tra programmi Unix, come ad esempio il programma di display astronomico *ds9*.

La prima, visto l'ampio uso in diversi settori, si trova generalmente in forma pacchettizzata per il sistema in uso, la seconda è disponibile solo come sorgente; deve essere compilata e installata seguendo le istruzioni specifiche per la distribuzione usata.

1.1 Il nuovo codename gbridge-balor

Le release più recenti delle nostre applicazioni sono indicate con una coppia di nomi: il primo rappresenta il *function name*, il secondo il *codename*.
Il *function name*, se presente, è il nome originale con cui è nata e conosciuta l'applicazione. Il *codename*, invece, è stato mutuato dal nome di personaggi del gioco "Dungeons & Dragons"[1], scelti in modo che le loro caratteristiche richiamino in qualche modo alla mente le proprietà salienti dell'applicazione.

[1]Dungeons & Dragons, D & D and Wizards of the Coast and related logos are trademarks or registered trademarks of Wizards of the Coast

Il software originario gbridge è stato ora rinominato con la coppia di nomi gbridge-balor. Il primo è appunto il nome con cui questa applicazione è stata individuata fin dall'inizio; balor rappresenta invece una classe di demoni tra i piú potenti di "Dungeons & Dragons".
Per questo motivo, tenuto conto del ruolo svolto da questa applicazione all'interno del sistema di controllo di Giano, ci è sembrato un nome adatto da affiancare alla vecchia nomenclatura.

1.2 Architettura del sistema di acquisizione

Il sistema di controllo dello spettrometro GIANO è distribuito su tre processi distinti:

- l'applicazione, eventualmente remota, Giano GUI, cioè il processo `cliente`, tramite il quale l'osservatore inoltra le richieste di esecuzione delle operazioni ed attende i relativi dati di risposta.
- il processo gbridge-balor in grado di accettare le richieste di connessione provenienti dal *client* e di tradurre le richieste della Giano GUI in comandi per il sistema embedded. Si occupa inoltre della ricezione e del salvataggio dei dati .
- l'agente di controllo server104 in esecuzione su una macchina *embedded* remota, a cui l'applicazione *middle-ware* gbridge-balor si appoggia per portare a termine le richieste dell'operatore

Da questo schema segue che l'applicazione Giano GUI vede il sistema *embedded* come un generico device di rete a cui accede *tramite* il canale di rete di gbridge-balor, che fa da ponte verso lo hardware.
Il canale di rete puó essere implementato come un semplice *socket* locale (Unix) oppure di rete (Internet) a seconda che le applicazioni Giano GUI e gbridge-balor siano processi locali in esecuzione su una stessa macchina oppure siano remoti.

1.3 Il modello client-server

Il modello che abbiamo utilizzato per la realizzazione della nostra struttura è quello noto come *client-server*, comune per applicazioni in rete di questo tipo.
Il paradigma *client-server* consiste in due o più programmi interagenti, da una parte vi è il *server*, un processo in attesa di essere contattato da ono o più altri processi, detti *client*, in modo da poter soddisfare le varie richieste. gbridge-balor in questo schema mentale *client/server* ha in realtà una doppia natura: si comporta come *server* nei confronti del programma Giano GUI

in quanto svolge una serie di operazioni per conto di questo, e come *client* nei confronti del sistema *embedded* in quanto fruisce dei suoi servizi per completare le richieste dell'osservatore.
La sua stessa implementazione software rispecchia questa duplice identità.
La comunicazione viene sempre avviata dal processo cliente Giano GUI e lo scenario tipico è il seguente:

- il processo gbridge-balor è attivato su una macchina in attesa di essere contattato dal processo *client* Giano GUI
- il processo *client*, in esecuzione sulla stessa macchina del programma gbridge-balor oppure su un altra macchina connessa attraverso la rete, trasmette una richiesta al processo gbridge-balor richiedendo uno specifico tipo di servizio
- gbridge-balor inoltra la richiesta al processo *embedded* server104 e rinvia la risposta all'applicazione *client*.

La comunicazione tra questi tre processi segue le specifiche del protocollo illustrato in [1] e [2].
Come molti dei processi *server*, gbridge-balor è un processo *demone* eseguito in *background* sulla macchina che lo ospita. Non necessita dei permessi di *super user* in quanto non accede a risorse di sistema e puó essere avviato in modo semplice dal programma Giano GUI, oppure da opportuni script di *shell*.

1.4 Server iterativi

gbridge-balor è strutturato come un server iterativo: gestisce le richieste e i messaggi provenienti dalle due applicazioni una alla volta, in modo continuo fino alla sua terminazione o alla disconnesione delle due applicazioni.
La funzionalità di un server iterativo è solitamente realizzata con un *loop* durante la cui esecuzione il sistema rimane semplicemente in attesa delle richieste da parte del *client*.
Il ciclo costituente il costrutto centrale del controllo di flusso dell'applicazione, viene indicato con il termine *loop principale* o *main loop*.
Se l'evoluzione del sistema è guidato da eventi software che si verificano durante l'esecuzione del ciclo principale, questo è detto anche *ciclo o loop degli eventi* e l'applicazione viene definita *applicazione ad eventi*.
L'applicazione gbridge-balor appartiene a questa categoria.
La natura di questi eventi puó essere varia: di solito si tratta di specifiche condizioni che si verificano su descrittori connessi all'applicazione server, in corrispondenza dei quali l'applicazione associa uno specifico gestore di funzione (*callback*).

Particolarmente utili per le applicazioni di rete sono, ad esempio, le funzioni di *input*, che vengono invocate quando un determinato descrittore di file è pronto per la lettura o scrittura (operazione di I/O).
Al ciclo principale possono essere aggiunte funzionalità, come chiamate di *callback* quando il ciclo è in pausa, oppure funzioni di timeout richiamate a determinati intervalli di esecuzione.

1.5 Il loop degli eventi

Il *loop degli eventi* comprende la successiva interrogazione delle varie *sorgenti di eventi*. Se una sorgente viene trovta in ricezione, la scansione si interrompe sino al completamento della ricezione ed alla chiamata del gestore associato.
Per applicazioni di rete le sorgenti di eventi sono principalmente descrittori di *socket* e gli eventi a cui essi sono sensibili si riassumono nella disponibilità del *socket* a operazioni di I/O.
Uno dei problemi che si possono avere quando si lavora su piú descrittori di *socket* ciascuno con i propri accessi di lettura/scrittura, consiste nel rischio, non prevedibile, di rimanere bloccati su un descrittore che che non è pronto nell'eseguire un'operazione quando ce ne potrebbe essere un altro disponibile.
Il metodo classico per prevenire un comportamento di questo tipo consiste nel ricorrere alla tecnica di *multiplexing* non bloccante dell' I/O.
Il *multiplexing* rappresenta una modalità di lavoro in cui con una sola *system call* è possibile tenere sotto controllo le richieste di lettura o scrittura per diversi dispositivi, con possibilità di accedere ai risultati non appena una di esse sia stata soddisfatta o di bloccare il processo se le operazioni volute non sono possibili.
Per ottenere questo risultato, il programma gbridge-balor si affida, a esempio, alla chiamata di sistema `poll` il cui prototipo POSIX

```
int poll(struct pollfd fds[], nfds_t nfds, int timeout);
```

Questo mostra come il suo funzionamento dipenda da un insiemi di descrittori di file che, organizzati in un array di strutture di tipo *pollfd* e di dimensione *nfsd*, individuano i canali di cui occorre monitorare lo stato, in attesa di un suo cambiamento. Ogni struttura *fds* raccoglie le informazioni dei descrittori da cui si vuole leggere, scrivere o di cui si vuole monitorare gli errori, e degli eventi a loro associati.
L'uso del *multiplexing* presenta il vantaggio di risolvere anche tutti i problemi di sincronizzazione in quanto gli eventi gestiti da un processo, se concorrenti, vengono serializzati. Inoltre l'applicazione puó conseguire un'elevata scalabilità potendo servire un numero elevato di connessioni contemporanee.
Il lato negativo è rappresentato da un codice meno leggibile a causa della gestione di diversi casi nello stesso flusso di controllo.

Capitolo 2

Strumenti software

Questa sezione illustra gli strumenti software con i quali l'applicazione *client-server* sviluppata si interfaccia al protocollo TCP.
Nelle applicazioni di *networking* l'interfaccia di programmazione più comunenemente usata è stata sviluppata dalla Berkeley Software Distribution, ed è disponibile in molti sistemi operativi sia Unix che non posix. Questo protocollo ruota intorno al concetto di *socket*.
Un *socket* rappresenta un punto terminale (*endpoint*), cioè una vera e propria *presa*, a cui il canale di comunicazione è collegato dal lato del sistema operativo da una parte, e dal lato dell'utente, dall'altra.

2.1 Creazione di una connessione

La vita di una connessione TCP si articola in tre fasi distinte:

- apertura della connessione;
- dialogo per mezzo della connessione;
- chiusura della connessione.

Se la socket si comporterà da *server* oppure da *client* e come questa viene creata dipende dalla tipologia dell'applicazioner.
Nel primo caso, una volta creato l'estremo del canale di comunicazione con la chiamata di sistema `socket` sul dominio in uso, viene eseguito un `bind` all'indirizzo del server e un `listen` per porre il *socket* in modalità di ascolto. La chiamata ad `accept` da' inizio per il processo, a un'attesa che si concluderà solo quando sul *socket* di ascolto sarà intercettata una richiesta di connessione da parte di un processo *client* remoto o locale.
Nel secondo caso, dopo la chiamata `socket` e la creazione di un *socket* del dominio richiesto, segue una richiesta alla *syscall* `connect`: la trasmissione di dati tra due processi avviene solo successivamente all'attivazione della

connessione tra questi. Questa operazione richiede che il processo *client* specifichi tra gli argomenti l'indirizzo del processo *server* a cui intende connettersi.
Per una connessione remota (dominio *Inet*), questo implica conoscere l'indirizzo IP dell'host sul quale il server è in esecuzione, nonché la specifica porta su cui quest'ultimo rimane in ascolto.
Per una connessione locale (dominio *Unix*), la comunicazione su *socket* è una forma alternativa di IPC (*Inter Process Communication*), e il *client* deve sapere solo il *pathname* Unix del processo destinatario.
Dal punto di vista dello sviluppo del codice sorgente la differenza sostanziale tra le due modalità si riflette solo nel diverso modo di indirizzare i *socket*.

2.2 Il software di comunicazione

La programmazione in C con l'uso dei *socket* e della chiamata di sistema `poll` dà come risultato programmi sequenziali relativamente efficienti. I *socket* e le chiamate di sistema costituiscono peró un'interfaccia di basso livello.
L'applicazione gbridge-balor comprende molte attività relative alla comunicazione come l'inizializzazione dei *socket*, la connessione, l'invio e ricezione di messaggi, il rilevamento e recupero degli errori. Inoltre l'applicazione esegue la parte di *demultiplexing* delle richieste e questo comporta la gestione di molti dettagli di basso livello che coinvolgono le *bitmask* degli eventi, i descrittori, i timeout e i segnali.
Per rendere il codice piú agevole e riusabile, abbiamo costruito una piccola libreria di funzioni di medio e alto livello che gestisce la complessità della comunicazione di rete sopra descritta, nascondendo i diversi dettagli dell'implementazione. Tale libreria è stata riutilizzata anche da altri demoni del progetto.
La ricezione e trasmissione dei messaggi è stata realizzata a basso livello ricorrendo alle chiamate di sistema `write`, `read` e `recv`. I *socket* in uso sono configurati `sempre` come non bloccanti, sia relativamente alle operazioni di lettura/scrittura, sia per per quelle di connessione portate a termine da `accept` e `connect`.

2.2.1 Il protocollo a livello di applicazione

Viene qui analizzata la comunicazione, a livello di applicazione tra gbridge-balor, il processo locale Giano GUI e l'agente di controllo server104, descrivendo brevemente il formato dei messaggi di richiesta ed il formato dei messaggi di risposta trasmessi dal server al client.
Il protocollo prevede la pacchettizzazione dei dati, operazione che consiste nel raggruppare le informazioni da spedire in pacchetti di lunghezza variabile (espressa in byte) aggiungendo ad essi un'intestazione o *header*.

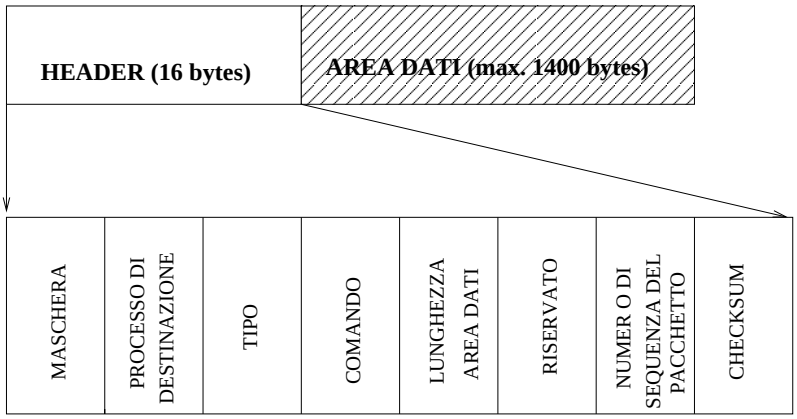

Figura 2.1: *Struttura del pacchetto e campi dell'intestazione*

La struttura generale di un messaggio, sia di richiesta sia di risposta, è quella riporta in fig.(2.1).
La lunghezza in byte dell'area dati non puó essere maggiore di 1400 byte. Lo *header* ha invece dimensione fissa (16 byte) ed è organizzato in 8 campi che definiscono la destinazione del pacchetto e la sua funzione (vedi fig 2.1).
Con questa struttura, l'applicazione gbridge-balor può identificare la funzione richiesta testando il valore di specifici elementi dell'intestazione (`Tipo` e `Comando`) e se la particolare funzione lo richiede, estrarre anche i dati.
In modo analogo, le applicazioni Giano GUI e server104 possono processare con la stessa tecnica i dati di risposta a un comando precedentemente trasmesso.
Per maggiori dettagli sul protocollo, rimandiamo comunque al documento [1].
Ci preme solo sottolineare come l'implementazione di un protocollo di comunicazione di questo tipo all'interno di un sistema di trasmissione bufferizzata come quella di rete, fornisca un controllo completo e sicuro sui dati trasmessi e ricevuti.

Capitolo 3

Dettagli sul funzionamento di gbridge-balor

gbridge-balor puó essere avviato da linea di comando oppure direttamente dal programma Giano GUI, attraverso un script apposito.
Essendo sviluppato come applicazione *server*, viene eseguito in *background* e il suo output viene rediretto su un file di log.
Il software gbridge-balor accetta al suo avvio una serie di opzioni a linea di comando, visualizzabili con il comando `gbridge -h`. L'output è il seguente:

```
-h          Output this help
-l          Disable logging on file (Default on)
-m          Message Level 0 1 2 3 (Default 3)
-n <n>      select local (0) or Inet (1) namespace (Default 0)
-q <n>      Acquire <n> quadrant (default 4)
-s <n>      set server name (Default pc104)
-v <level>  verbose level 0 1 2 (Default 0)
-V <level>  Embedded system verbosity level [0-9] (Default 1)
```

Il significato di queste opzioni è descritto nel seguito.
L'opzione `l` disabilita la creazione del file di log. La creazione di questo file è abilitata di default, perché in processi eseguiti in background, rappresenta l'unico modo che ha lo sviluppatore di seguire l'esecuzione del programma e rintracciare gli eventuali comportamenti anomali del sistema.
L'opzione `m` configura il livello minimo di severità dei messaggi inviati all'applicazione GUI. Ponendo tale valore uguale al massimo, la quantità di informazioni scambiate si riduce.
L'opzione `n` configura la connessione tra gbridge-balor e l'applicativo GUI come connessione di rete, oppure come connessione locale. Quest'ultima rappresenta il caso di funzionamento classico poiché è previsto che entrambe le applicazione siano eseguite sulla macchina di controllo.
L'opzione `q` consente di lavorare con 1 oppure tutti e 4 i quadranti. Questa modalità riflette uno specifico funzionamento del processo *embedded* che

prevede un modo ad un unico quadrante attivo, utilizzato nella prima fase di test del sistema di acquisizione.
L'opzione `s` permette di specificare il nome, oppure l'indirizzo di rete, della macchina su cui è in esecuzione il processo *embedded*.
L'opzione `v` configura il livello di verbosità dei messaggi prodotti da gbridge-balor.
L'opzione `V` permette invece di selezionare la verbosità dell'applicazione server104.
Alcune di queste opzioni risultano comode durante la fase di sviluppo e verifica della strumentazione. Altre, invece, rendono il programma facilmente adattabile alle possibili modifiche introdotte alla struttura hardware del sistema.

3.1 Le connessioni con Giano GUI e il server104

L'applicazione gbridge-balor controlla il funzionamento del server104 attraverso due *socket* permanenti aperte sullo *host* di controllo in corrispondenza delle porte `8082` e `8083`: il primo rappresenta il `canale di controllo` (o `canale comandi`) attraverso il quale passano i comandi e i messaggi da e verso il server104, formattati secondo le specifiche del protocollo illustrate in § 2.2.1.
Il secondo è il `canale dati` usato per ricevere dal processo *embedded* le immagini acquisite, con un protocollo diverso dal precedente.
La comunicazione tra gbridge-balor e il programma *client* Giano GUI è stabilita invece attraverso una *socket* locale, aperta in corrispondenza del *pathname unix* `/var/tmp/GUIlab.sock`. Anche quest'ultima è identificata come `canale di controllo`, in quanto su di esso passano esclusivamente informazioni formattate secondo il protocollo [1].
Il processo Giano GUI invia sul `canale di controllo` richieste di esecuzione comandi che hanno come destinatario principale il processo *embedded*. gbridge-balor traduce queste richieste in comandi elementari che invia successivamente all'applicazione server104. Sullo stesso canale passano le risposte originate dal processo remoto, oppure da gbridge-balor stesso, e che hanno come processo destinazione l'applicazione *client*.

3.2 Cenni sul protocollo

Le informazioni che passano sul canale di controllo possono essere classificate in quattro gruppi: comandi, messaggi, errori, informazioni e conferme (ack).

3.2.1 Pacchetti comando

I pacchetti di tipo comando consentono di operare sul sistema di acquisizione sia configurando i parametri specifici dell'elettronica, sia avviando le diverse procedure disponibili per la presa dati: integrazione, freerun e multicampionamento.
I pacchetti di tipo comando specificano nell'intestazione quale comando il sistema deve eseguire e nell'area dati gli eventuali parametri. Questi pacchetti sono sempre confermati con un pacchetto comando di tipo *acknowledge*.
Il processo destinatario è il processo che deve confermare il pacchetto, ma come spiegheremo più avanti, in alcuni casi può essere il software middleware che si incarica di confermare al programma Giano GUI un pacchetto comando destinato al programma server104.

3.2.2 Pacchetti messaggio e errore

I pacchetti di tipo messaggio (MESSAGGIO) e errore (ERROR) includono sempre un'area dati in cui è specificato il testo del messaggio o dell'errore. Quest'ultimo è identificato anche attraverso un codice esadecimale, che viene specificato in uno dei campi dell'intestazione (mettere spiegazione).

3.2.3 Pacchetti informazione

I pacchetti di tipo informazione (INFO) sono generati principalmente dal processo server104 e contengono le informazioni necessarie a individuare lo stato del sistema durante tutto la fase di durata della presa dati. Pacchetti di tipo INFO comunicano l'inizio dell'integrazione, il numero del frame in integrazione, il termine di questa e la fine del trasferimento dell'immagine.
Le indicazioni ricevute permettono a gbridge-balor di seguire l'evoluzione del processo di acquisizione, configurando opportuni timer per intervenire direttamente nel caso siano rilevate condizioni di timeout nella misura.

3.2.4 Pacchetti conferma

Come descritto in § 3.2.1, tutti i pacchetti di tipo `comando` spediti dal processo Giano GUI devono essere confermati dal processo di destinazione che può essere gbridge-balor o il server104. Questo avviene attraverso i pacchetti di tipo `acknowledge` (ACK) che nell'intestazione specificano sia il processo destinatario della risposta, sia il comando confermato.
Questo tipo di pacchetto prevede anche la presenza di un'area dati.

3.3 Stato del sistema di acquisizione

Lo *stato del sistema* descrive sia la disponibilità dell'elettronica all'acquisizione cioè lo *stato dell'acquisizione* (`pronto` oppure `occupato`), sia l'insieme

dei valori programmati dell'elettronica di controllo: livelli di offset e bias, tempi di scansione, numero di reset etc.
Nella funzionalità del software gbridge-balor il tracciamento dello *stato dell'acquisizione* del sistema risulta essere un punto delicato: questo deve risultare sincronizzato con quello del processo di acquisizione *embedded*, in modo che per l'osservatore le informazioni fornite dal programma di controllo dello strumento rispecchino fedelmente lo svolgimento delle funzioni.
Oltre a inoltrare al processo *embedded* le richieste dell'operatore, un altro compito di gbridge-balor consiste nello stabilire se il processo destinatario è pronto, oppure no, ad accettarle per processarle. È quindi fondamentale che il processo *middle-ware* abbia una completa conoscenza dello stato interno del sistema di controllo.
Lo *stato di acquisizione* del sistema varia durante lo svolgimento delle diverse operazioni, e di seguito illustreremo come ciò avviene, e il software *middle-ware* ne mantiene traccia al suo interno insieme alla configurazione dello *hardware*.

3.3.1 Stato del sistema di acquisizione

In condizioni di inattività, il sistema *embedded* legge continuamente il rivelatore (*dummy acquisition*) operazione che garantisce la cancellazione degli effetti di memoria, fenomeno presente in rivelatori del tipo utilizzato dello spettrometro GIANO.
Il sistema così funzionante risulta essere nella stato `ready` ed è disponibile ad accettare una richiesta di misura.
La ricezione da parte del software gbridge-balor di un comando di acquisizione (INTEGRA, MULTI, FREERUN) o reinizializzazione dell'*hardware* (REINIT) predispone lo stato come `busy`: in questa configurazione un qualunque altro comando esecutivo o di programmazione, non viene né processato, né inoltrato al server104.

PROCESSO ORIGINE	COMANDO	GBRIDGE STATO DELL'ACQUISIZIONE
GUI	INTEGRA →	READY → BUSY
SERVER104	ACK INTEGRA →	BUSY → RUNNING
SERVER104	INFO FRAME_STARTED →	RUNNING (calcolo timeout acquisizione)

Figura 3.1: *Stato dell'acquisizione dopo un comando di INTEGRA.*

Il rientro del sistema nello stato `pronto` dipende dallo specifico comando.

Per i comandi di reinizializzazione, il sistema torna disponibile al termine dell'esecuzione del comando stesso.
Per i comandi di acquisizione, lo stato del sistema da `busy` passa a `running` quando il processo gbridge-balor riceve la conferma del comando (vedi Fig.3.1). La conferma del comando non corrisponde però all'effettivo inizio della misura: tra le due azioni intercorrono una serie di operazioni, come ad esempio la programmazione dei livelli di offset, del sequencer, l'accensione del sensore, che possono fallire e determinare l'interruzione della misura in corso.
L'inizio dell'integrazione vera e propria avviene allo *start* del sequencer e viene segnalata con un pacchetto di tipo INFO, alla cui ricezione il processo gbridge-balor configura un timer per la gestione di eventuali condizioni di timeout nell'operazione di presa dati. Il valore calcolato come tempo massimo di attesa per completare la procedura di acquisizione tiene conto del tempo di integrazione della singola misura (tempo di ciclo), del numero delle misure programmate e del tempo medio richiesto per il trasferimento in rete della singola immagine.
Lo stato del sistema ritornerà infine allo stato iniziale `ready` quando gbridge-balor avrà ricevuto sia il pacchetto INFO di fine procedura di acquisizione sia tutte le righe dell'ultimo frame immagine.
Nel caso di acquisizioni multiple[1], il sistema rimane nello stato `running` per tutto il tempo necessario all'acquisizione del set completo delle misure.
La sequenza di acquisizione può essere interrotta volontariamente dall'utente richiedendo un comando di ABORT o STOP. Entrambi causano la terminazione del processo di presa dati, ma con modalità diverse.
Nel primo caso abbiamo una sospensione immediata del *sequencer* e la perdita completa del frame in acquisizione; nel secondo caso il *sequencer* si ferma alla prima occorrenza del comando RESTART nel programma di sequenza, il frame in corso di acquisisione viene letto interamente dalle FIFO di quadrante e trasferito al processo di destinazione gbridge-balor. In un caso quindi l'ultimo frame acquisito viene perso, nell'altro è trasferito integralmente al programma di alto livello.
Per quanto riguarda lo stato di acquisizione, questo da `running` passa ad `aborting` o `stopping` alla ricezione della richiesta inviata dal programma Giano GUI (vedi Fig.3.2), e permane nello stesso stato finché l'applicazione gbridge-balor non riceve la conferma della richiesta di ABORT o STOP da parte del *server embedded*. Una volta rilevato il pacchetto di conferma, gbridge-balor configura lo stato dell'acquisizione come `abort` o `stop`. Questo non significa che il processo server104 abbia effettivamente fermato il processo di acquisizione, ma solo che il sistema sta procedendo a soddisfare la richiesta. Quando ciò accade, il processo *server* invia a gbridge-balor un pacchetto INFO specificando nel campo dell'intestazione la modalità di fermata

[1] Con acquisizione multipla si intende una procedura in cui il numero di frame richiesti con lo stesso tempo di integrazione (gruppo), è maggiore di 1

(abort o stop) e il numero del frame in corrispondenza del quale la procedura di integrazione si è interrotta. Solo alla ricezione di questa informazione, il software *middle-ware* ritorna pronto per una nuova serie di misure.

PROCESSO ORIGINE	COMANDO	GBRIDGE STATO DELL'ACQUISIZIONE
GUI	ABORT →	RUNNING → ABORTING
SERVER104	ACK ABORT →	ABORTING → ABORT
SERVER104	INFO FRAME_ABORT →	ABORT → READY

Figura 3.2: *Evoluzione dello stato dell'acquisizione dopo un comando di ABORT.*

3.3.2 Sistema occupato e comandi non eseguiti

Tutti i comandi di acquisizione e configurazione dello *hardware* sono interdetti al processo *cliente* con il sistema nello stato diverso da `ready`: questa situazione si verifica quando, come abbiamo visto, il sistema è in acquisizione o in fase di reinizializzazione dello *hardware*.
Il rifiuto all'inoltro verso il sistema *embedded* viene segnalato da gbridge-balor al programma Giano GUI con un messaggio di *warning* del tipo "System is busy".

PROCESSO ORIGINE	COMANDO	PROCESSO DESTINAZIONE	GBRIDGE STATO DELL'ACQUISIZIONE
GUI	INTEGRA →	SERVER104	READY → BUSY
GUI	WRITEPARM →	SERVER104	BUSY
GBRIDGE	"System is busy" →	GUI	BUSY
GBRIDGE	ACK WRITEPARM →	GUI	BUSY

Figura 3.3: *Esecuzione del comando di configurazione WRITEPARM quando il sistema è* `busy`.

Il comando, anche se non inviato al processo server104 (vedi Fig.3.3), viene

comunque confermato al processo mittente da gbridge-balor: in questo modo si vuole evidenziare che il comando è stato ricevuto correttamente ma non è stato mandato in esecuzione perché il sistema di destinazione è impegnato in operazioni che potrebbero andare in conflitto con la richiesta.
Nel caso di sistema `non pronto`, solo i comandi di richiesta informazioni (STATUS, READPARM,...) sono accettati ma questi sono processati direttamente da gbridge-balor (vedi Fig.3.4) che si incarica di aggiornare il processo Giano GUI restituendo a quest'ultimo lo stato complessivo del sistema come era subito prima dell'inizio della procedura in esecuzione.

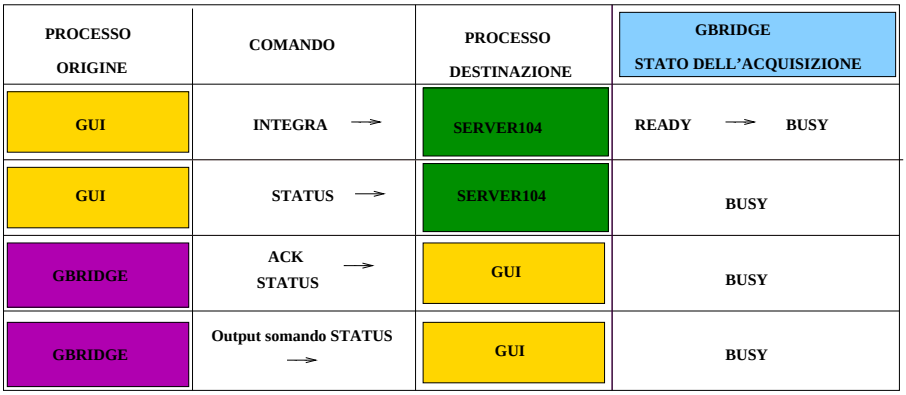

Figura 3.4: *Esecuzione del comando di informazione STATUS quando il sistema è* `busy`.

Gli unici comandi inviati al sistema *embedded* quando questo risulta `non pronto`, sono quelli relativi all'interruzione della misura in corso e cioé: ABORT e STOP.
Situazione analoga si verifica durante la configurazione dei parametri relativi all'operazione di acquisizione.
Il software *middle-ware* funziona da filtro tra i valori programmati dall'utente e quelli accettabili dal sistema di acquisizione, non inviando al sistema di acquisizione i parametri selezionati dall'osservatore qualora non appartengano all'intervallo di validità.
La decisione presa viene segnalata al processo *client* con un appropriato messaggio di errore. Anche in questo caso il pacchetto di conferma del comando viene generato da gbridge-balor stesso che nell'area dati specifica come ulteriore informazione per il processo utente, il valore del parametro attualmente programmato all'interno del sistema *embedded*.

Capitolo 4

Funzionalità aggiuntive svolte da gbridge-balor

Come accennato nei primi paragrafi (vedi § 1.4), un'applicazione ad eventi puó aggiungere altre funzionalità durante l'esecuzione del `ciclo degli eventi`. Di seguito illustriamo alcune operazioni che gbridge-balor esegue e monitora durante il *loop principale*.
In particolare, in tutti i casi sono operazioni che gbridge-balor svolge su base temporale a istanti di tempo ben precisi e quindi adatte ad essere gestite all'interno del *ciclo degli eventi*.

4.1 Il monitoraggio della rete

Una funzione essenziale svolta dal software *middle-ware* consiste nel monitoraggio dello stato della comunicazione di rete con il sistema remoto per stabilire se questo rimane contattabile nel tempo.

4.1.1 Sistema embedded non raggiungibile

I classici esempi di *host* non raggiungibile, ammesso ovviamente che la configurazione di rete sia corretta, si hanno quando la macchina *embedded* esegue un riavvio anomalo, oppure viene spenta o disconnessa dalla rete mentre è stabilita la connessione con l'applicazione remota gbridge-balor.

Reboot anomalo del PC embedded

Un'operazione di riavvio è da considerarsi anomala quando il sistema non ha la possibilità di terminare le applicazioni in esecuzione in modo regolare: ad esempio per le applicazioni *server* questo corrisponde all'invio del segnale SIGTERM.
Nel caso di un reboot non regolare, lo stack TCP/IP del PC *embedded* non ha la possibilità di inviare un pacchetto FIN per chiudere la connessione

in modo appropriato. D'altra parte, quando al termine della procedura di *bootstrap* del PC embedded l'applicazione *server* è mandata in esecuzione, questa non ha "memoria" delle vecchie connessioni: l'applicazione gbridge-balor rimane quindi con delle connessioni che hanno come riferimento una destinazione inesistente.
Nel momento in cui al sistema embedded arriva un pacchetto da gbridge-balor relativo alla connessione non più esistente, lo stack TCP/IP invia al mittente un pacchetto RST (reset): solo allora l'applicazione gbridge-balor è in grado di rilevare l'errore e puó procedere a chiudere la vecchia connessione.
C'é da notare che le operazioni di scrittura sul *socket* da parte dell'applicazione gbridge-balor non falliscono immediatamente: la *system call* `write` si limita a copiare i dati nel buffer di invio del *socket* per poi ritornare immediatamente. È compito dello stack TCP/IP stabilire il momento più adatto per trasmettere i dati: può quindi accadere che il software gbridge-balor debba eseguire più scritture sul *socket* prima che l'applicazione *embedded* generi un pacchetto RST in risposta.

PC spento o disconnesso dalla rete

Il caso di PC spento bruscamente o disconnesso temporaneamente dalla rete è una condizione piú subdola da rilevare. Per lo *stack* TCP è impossibile distinguere questa situazione da quella piú semplice di una perdita temporanea di connessione al server. Il comportamento classico dello *stack* TCP/IP consiste nel tentativo ripetuto di rispedire i dati. Questa operazione prosegue per un tempo abbastanza lungo prima che la connessione venga considerata definitivamente persa. Nel frattempo, la connessione tra le due applicazioni remote risulta per entrambe sempre attiva.
Per accertarsi di questa affermazione, è sufficiente eseguire il comando
`#netstat -nt`
e controllarne l'*output*: nel caso specifico in corrispondenza delle connessioni alle porte **8082** e **8083** lo stato della connessione risulta `ESTABLISHED`.
Questa significa che se gbridge-balor scrive qualcosa sul `canale di comando`, l'operazione non fallisce [1]: l'imprevisto è che la richiesta non viene inoltrata, ma questo l'applicazione mittente non puó saperlo.

4.1.2 Il pacchetto di protocollo NOP

Da entrambe le situazioni descritte si deduce che se non viene mandato qualcosa in modo regolare all'applicazione *embedded*, si può rimanere con una connessione "zombi" per un tempo piuttosto lungo.
Come lavora dunque gbridge-balor?

[1] In realtà l'operazione di scrittura fallisce quando il buffer di uscita è pieno. Prima che questo errore si manifesti sono comunque necessarie diverse operazioni di scrittura.

Ad intervalli regolari, escluso durante la serie di integrazioni, l'applicazione gbridge-balor interroga il sistema remoto inviando al processo *embedded* un pacchetto comando di tipo NOP (Not Operative Command) per determinare lo stato della connessione.
Questo comando forza la risposta da parte del server104 di un pacchetto di conferma (ACK) che, se ricevuto entra un tempo prestabilito, consente a gbridge-balor di stabilire che la macchina remota è contattabile. In caso contrario il sistema *embedded* è considerato non è raggiungibile e gbridge-balor procede a chiudere entrambi i *socket* connessi, segnala il problema al processo utente inviando una serie di messaggi esplicativi, chiude il *socket* locale con l'applicazione Giano GUI e infine si pone nuovamente in attesa di una connessione da parte di quest'ultimo.
Questa procedura fornisce all'operatore il tempo necessario per individuare il problema e, una volta risolto, riavviare il sistema di acquisizione nel modo corretto.

4.2 Lista dei comandi in attesa di conferma

gbridge-balor tiene traccia di tutti i comandi inviati al server104 registrandoli in un `Lista di attesa conferma` dalla quale sono rimossi una volta che il programma agente server104 li ha confermati con un pacchetto di *acknowledgment*.
Quando gbridge-balor riceve (o genera) un pacchetto di protocollo con destinazione il server104, ne controlla il campo `Tipo`. Se questo corrisponde al tipo `comando` viene aggiunto alla `Lista di attesa conferma` e permane in tale lista finché il comando non è convalidato dal processo *server*.
Al momento della registrazione, oltre al codice identificativo della funzione, viene tenuto in memoria anche il processo originario del pacchetto, il tempo di invio (`timestamp`) e il numero di sequenza.
Quando gbridge-balor riceve un pacchetto di `Tipo Ack`, esegue un controllo sulla lista dei comandi in attesa e se trova la corrispondenza tra il pacchetto ricevuto e uno registrato, procede a rimuoverlo dall'elenco e invia la conferma del comando al processo destinazione.
Durante l'esecuzione del `ciclo degli eventi` il processo gbridge-balor aggiorna la lista controllando quali pacchetti sono ancora in attesa e da quanto tempo. Se la permanenza del pacchetto nella lista ha superato il valore limite configurato come tempo massimo di attesa, il pacchetto viene rimosso e al processo Giano GUI viene notificato l'errore di comando non convalidato.
L'operazione di cancellazione dei comandi dalla lista deve essere eseguita con attenzione specialmente quando riguarda funzioni relative al processo di integrazione: as es. INTEGRA, MULTI, FREERUN, STOP, ABORT.
L'analisi è piuttosto semplice visto la registrazione del codice del comando. gbridge-balor in questi casi controlla il valore dello *stato di acquisizione*

del sistema. Se questo risulta `busy`, `aborting` o `stopping` significa che il comando inviato è stato accettato da gbridge-balor ma non è stato ancora confermato dal sistema *embedded* entro l'intervallo prestabilito.
Allora è possibile che la procedura di acquisizione o la sua interruzione, non sia stata avviata e quindi il sistema *embedded* risulti inattivo (idle), oppure che sussistano problemi di comunicazione tra le due applicazioni. Comunque sia, lo *stato dell'acquisizione* viene riportato a `ready` e l'operazione viene segnalata al processo di alto livello Giano GUI.
Il tempo limite di permanenza nella lista di attesa è configurato diversamente a seconda che si tratti di un comando di acquisizione oppure di configurazione o richiesta di stato[2]
Questo diverso comportamento è legato al fatto che l'applicazione server104 impiega piú tempo per confermare il primo tipo di comandi. Al momento della ricezione di un comando di *avvio misura*, infatti, il sistema impiega diversi secondi per fermare le acquisizioni *dummy* e, in modo analogo, per come è strutturata la routine di lettura dei frame, impiega diverso tempo per processare un eventuale comando di ABORT o STOP della misura.
Situazioni analoghe non si verificano per i comandi che richiedono informazioni sullo stato del sistema o che programmano una particolare configurazione dell'elettronica.

4.3 La gestione degli errori: il buffer circolare

Abbiamo implementato un buffer circolare in cui sono trascritti gli errori generati dal sistema *embedded*. Quando il sistema lavora con tutte e quattro le schede analogiche, il messaggio di errore generato da una si ripete, talvolta, per tutte le altre con la consequenza che il programma utente riceve lo stesso tipo di avvertimento per quattro volte consecutive.
Risulta utile implementare un sistema di filtraggio dei messaggi di errore che limiti la quantità di informazioni spedite al processo Giano GUI.
Questa operazione potrebbe essere introdotta a livello del server104 e in una versione successiva questo sarà il caso, ma al momento abbiamo ritenuto oppurtuno inserire questo controllo a livello del software *middle-ware*.
Quando gbridge-balor riceve un pacchetto di tipo errore dal processo server104, invece di inoltrarlo direttamente al programma Giano GUI, trascrive l'informazione relativa all' errore in un *buffer* circolare di dimensione finita, registrando il codice identificativo dell' errore, il messaggio ASCII ad esso collegato e il tempo di ricezione (`timestamp`).
Ad ogni esecuzione del *loop principale* il buffer viene analizzato e se è trascorso un tempo superiore a 1 sec. dal controllo precedente, gbridge-balor procede a esaminare gli errori registrati. Se vengono trovati due o più errori con stesso codice con `timestamp` a distanza di meno di mezzo secondo, i

[2]Si va da un timeout di circa 18 sec. per i primi a 3 sec. per gli altri.

messaggi in eccesso sono scartati e il messaggio risultante viene etichettato con la stringa ''last message repeated n time'' dove n indica il numero complessivo di errori identificati aventi lo stesso codice e la stessa severità (si fa cioé distinzione tra il caso di errore e quello di *warning*). Dopo questa operazione, il messaggio di errore viene infine inviato al processo Giano GUI.

4.4 Interfacciamento con l'applicazione couatl

L'applicazione couatl, di recente sviluppo e strutturata come applicazione *server*, realizza un'interfaccia software per utilizzare da remoto ed in automatico parte delle funzionalità dell'applicazione gianoMotors, più specificatamente quelle relative al solo controllo delle movimentazioni (vedi [4]).
Invece di interfacciarsi direttamente con l'applicazione Giano GUI, couatl riceve i comandi attraverso un canale che, all'avvio, apre verso gbridge-balor. Tutte le richieste di movimento motori e le relative risposte fornite dal sistema di controllo gianoMotors, provenienti o dirette verso l'interfaccia GUI da cui opera l'osservatore, passano cosí in modo del tutto trasparente, da gbridge-balor.
Il compito specifico di quest'ultimo consiste nell'avviare, su richiesta del programma GUI, il *server* couatl e di mantenere attivo e controllare l'attività del *socket* di connessione con esso stabilito.
Le informazioni che viaggiano su questo canale fanno uso del medesimo protocollo a cui gbridge-balor ricorre per comunicare su canali di `tipo comandi` (vedi§ 2.2.1).
Il *socket* aperto da couatl appartiene alla famiglia di protocolli Unix. Si tratta quindi di un *socket* locale, con le due applicazioni, gbridge-balor e couatl in esecuzione sulla stessa macchina.

4.4.1 Avvio e controllo dell'applicazione couatl

gbridge-balor avvia il server couatl prima di entrare nel ciclo principale degli eventi, controllando la presenza di un'altra istanza dello stesso programma. Una volta in esecuzione, couatl apre un *socket* Unix in corrispondenza del *pathname* `/var/tmp/Couatl.sock` e rimane in attesa di una richiesta di connessione da parte di gbridge-balor. Questa avviene solo in risposta ad una esplicita richiesta di accesso al sistema di controllo dei motori (vedi [2]) da parte dell'operatore.
Durante il ciclo degli eventi gbridge-balor controlla l'attività del descrittore di *socket* connesso con couatl e manda in esecuzione le funzioni registrate in corrispondenza degli eventi che si verificano.
I comandi ricevuti dall'interfaccia di alto livello e destinati al processo couatl, vengono direttamente instradati verso la connessione con couatl. In modo si-

mile,i messaggi e le informazioni originate dal sistema di controllo dei motori sono spedite direttamente all'operatore.
gbridge-balor non mantiene alcuna informazione sullo stato dei motori, né sullo stato del sistema di controllo di questi. Tale compito è lasciato in gestione esclusivamente al *server* couatl. In questa circostanza l'applicazione gbridge-balor agisce come un semplice "passa-carte" tra i programmi Giano GUI e couatl.

4.4.2 Estensione del protocollo di comunicazione per l'interazione con couatl

L'introduzione della nuova applicazione *server* couatl ha reso necessario l'estensione della lista dei comandi riconosciuti da gbridge-balor e Giano GUI per includere quelli relativi alla gestione dei motori.
Questi ulteriori comandi sono, al momento, relativamente pochi: avremo un comando di stato dei motori, un comando per il movimento, uno per lo stop di emergenza, uno per la terminazione del programma gianoMotors e uno, infine per la terminazione del demone couatl.
Il comando di movimento specifica come argomento alcune configurazioni che codificano una precisa combinazione delle ottiche: quelle piú comunemente usate nelle osservazioni spettroscopiche.

Indice

1 L'applicazione gbridge-balor **3**
- 1.1 Il nuovo codename gbridge-balor . . . 3
- 1.2 Architettura del sistema di acquisizione . . . 4
- 1.3 Il modello client-server . . . 4
- 1.4 Server iterativi . . . 5
- 1.5 Il loop degli eventi . . . 6

2 Strumenti software **7**
- 2.1 Creazione di una connessione . . . 7
- 2.2 Il software di comunicazione . . . 8
  - 2.2.1 Il protocollo a livello di applicazione . . . 8

3 Dettagli sul funzionamento di gbridge-balor **10**
- 3.1 Le connessioni con Giano GUI e il server104 . . . 11
- 3.2 Cenni sul protocollo . . . 11
  - 3.2.1 Pacchetti comando . . . 12
  - 3.2.2 Pacchetti messaggio e errore . . . 12
  - 3.2.3 Pacchetti informazione . . . 12
  - 3.2.4 Pacchetti conferma . . . 12
- 3.3 Stato del sistema di acquisizione . . . 12
  - 3.3.1 Stato del sistema di acquisizione . . . 13
  - 3.3.2 Sistema occupato e comandi non eseguiti . . . 15

4 Funzionalità aggiuntive svolte da gbridge-balor **17**
- 4.1 Il monitoraggio della rete . . . 17
  - 4.1.1 Sistema embedded non raggiungibile . . . 17
  - 4.1.2 Il pacchetto di protocollo NOP . . . 18
- 4.2 Lista dei comandi in attesa di conferma . . . 19
- 4.3 La gestione degli errori: il buffer circolare . . . 20
- 4.4 Interfacciamento con l'applicazione couatl . . . 21
  - 4.4.1 Avvio e controllo dell'applicazione couatl . . . 21
  - 4.4.2 Estensione del protocollo di comunicazione per l'interazione con couatl . . . 22

Bibliografia

[1] "**Progetto Giano**. Protocollo di comunicazione tra gbridge-balor e Giano GUI", Vers 1.08, 2007, C.Baffa, E.Giani, Memo.

[2] "Definizione del protocollo interno del software di basso livello", versione 1.9, 2006, C.Baffa, E.Giani, Memo

[3] "**Progetto Giano**.Il programma server104 ed il sistema embedded", 2010, C.Baffa, V. Biliotti, E. Giani, Rapporto interno dell'Osservatorio di Arcetri, 2/2010.

[4] "**Progetto Giano**. Interfacciamento con il programma gianoMotors: il server couatl", E.Giani, C.Baffa, 2010, Arcetri Technical Report 6/2010

[5] "Dungeons & Dragons - Monster Manual 3.5", 2003, J. Tweet, M. Cook e S. Williams, TSR.