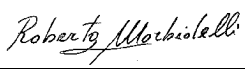
Relazione Scientifica del Progetto
TECNO- INAF 2010 – CIWS (CRA 1.05.01.15.04)

R.I. INAF IASF Bologna n. 641/2014

Preparato da:	M. Trifoglio et al. (vedi lista autori)		05/05/2014
Verificato da:	F. Pasian		05/05/2014
	R. Morbidelli		05/05/2014
Approvato da:	M. Trifoglio		05/05/2014
	G. Malaguti		05/05/2014

Lista Autori

Massimo Trifoglio, Andrea Bulgarelli, Massimo Cappi, Vito Conforti, Mauro Dadina, Enrico Franceschi, Fulvio Gianotti, Luciano Nicastro, Andrea Zoli	INAF/IASF Bologna, Italy
Roberto Morbidelli, Ricky Smart	INAF/OA Torino, Italy
Roberto Cirami, Paolo Di Marcantonio, Marco Frailis, Fabio Pasian, Stefano Sartor, Claudio Vuerli, Andrea Zacchei	INAF/OA Trieste, Italy
Marcello Lodi	INAF/TNG La Palma, Spain

Lista di distribuzione

INAF - Direzione Scientifica	direzione.scientifica@inaf.it, cns-inaf@astropa.inaf.it
CIWS e-mail list	ciws@iasfbo.inaf.it

INDICE

1. Introduzione

Il fiorire di progetti astronomici ed il contemporaneo sviluppo di strumentazione sempre piu' progredita ha, negli ultimi decenni, imposto lo sviluppo di tecniche di archiviazione e utilizzazione dei dati acquisiti sempre più complesse. Tale necessità ha avuto il costo di dirottare importanti risorse di intelligenza e tempo verso lo sviluppo di tecnologie atte a soddisfare queste esigenze.

La lunga tempistica (tipicamente decennale) e il dislocamento spaziale dei centri di sviluppo dei programmi e della strumentazione astronomica, ha, fino ad oggi, reso complicato razionalizzare la fase del processo relativo alla creazione e fornitura di workstation scientifiche relative agli strumenti, nonostante le molteplici analogia fra tutti (o quasi) gli strumenti per l'astronomia.

Partendo dalle premesse sopra presentate è nato il progetto CIWS (Customizable Instrument Workstation Software) che può permettere una notevole semplificazione per lo sviluppo di workstation scientifiche per i progetti futuri. Il CIWS, infatti, ha riguardato la progettazione e lo sviluppo di software per l'archiviazione e la visualizzazione on-line e off-line di dati acquisiti da strumentazione scientifica posta sia in telescopi di terra che a bordo e di telescopi spaziali.

Lo scenario di riferimento è quello della Instrument Workstation che fornisce supporto al team di sviluppo dello strumento durante la fase di integrazione, verifica e calibrazione e al team operativo dell'Osservatorio o del Segmento di Terra, durante le successive fasi di collaudo e operativa.

Come indicato nel nome, la configurabilità è la caratteristica fondamentale del progetto. A questa sono state dedicate molte risorse sia in fase di definizione dei requisiti e di progettazione sia in fase di realizzazione.

Il framework ottenuto (CIWS-FW) consente allo sviluppatore di modellare molti aspetti caratteristici dello strumento mediante file di configurazione. Altre funzionalità specifiche richieste per il trattamento dei dati dello strumento possono essere realizzate dallo sviluppatore utilizzando moduli di base (templates) e librerie offerte dal CIWS-FW.

1.1 Scopo

Le attività esposte in questa relazione riguardano:

- la composizione e organizzazione del team e l'addestramento di personale;
- la metodologia di sviluppo adottata;
- la strumentazione acquisita e messa in opera per realizzare l'ambiente di sviluppo e test;
- i risultati della ricerca in relazione a quanto originariamente proposto;
- la definizione e analisi dei requisiti e di progettazione del CIWS-FW;
- la realizzazione, l'integrazione e il test delle componenti del CIWS-FW;
- la realizzazione di due prototipi di "*proof of concept*" del CIWS-FW:
    - un prototipo di catena di acquisizione, archiviazione e Quick Look per i dati simulati della Camera di piano focale del costruendo Telescopio ASTRI;
    - un prototipo del Quick Look per i dati del satellite AGILE, dell'Agenzia Spaziale Italiana, in orbita dal 2007.

1.2 Acronimi e abbreviazioni

API Application Program Interface

AIV	Assembly, Integration and Verification
CIWS	Customizable Instrument Workstation Software
DAS	Data Acquisition System
DDL	Data Definition Language
DPS	Data Processing System
GUI	Graphical User Interface
IW	Instrument Workstation

1.3 Riferimenti

1.3.1 Documenti prodotti

[DD.1] CIWS User Requirement Document (URD), CIWS-IASFBO-TN-001

[DD.2] CIWS Software Specification Document (SSD), CIWS-IASFBO-TN-006

[DD.3] DAS Software Specification Document (SSD), CIWS-OATS-TN-001

[DD.4] DAS User Manual (SSD), CIWS-OATS-TN-002

[DD.5] PacketLib Programmers Guide, CIWS-IASFBO-TN-010

[DD.6] PacketLib Interface Control Document, CIWS-IASFBO-TN-011

[DD.7] ProcessorLib Programmers Guide, CIWS-IASFBO-TN-012

[DD.8] DPS Data Model, CIWS-IASFBO-TN-013

[DD.9] CIWS Reference Catalogues, CIWS-OATO-RLS-001

[DD.10] The CIWS-FW Persistence" - V. Conforti, A. Bulgarelli, F. Gianotti, M. Trifoglio, A. Zoli - 03/04/2014, Internal Report IASF Bologna Nr. 638/2014

[DD.11] "The CIWS-FW Control Panel" - V. Conforti, A. Bulgarelli, F. Gianotti, M. Trifoglio, A. Zoli - 20/04/2014, Internal Report IASF Bologna Nr. 639/2014

[DD.12] "The CIWS-FW Web Site" - V. Conforti, A. Bulgarelli, F. Gianotti, M. Trifoglio, A. Zoli - 02/04/2014, Internal Report IASF Bologna Nr. 640/2014

[DD.13] "PacketLib Tutorial: how to create simple telemetry generator software using the PacketLib" - V. Conforti, A. Bulgarelli, M. Trifoglio, F. Gianotti- 25/09/2011, Internal Report IASF Bologna Nr. 612/2012

[DD.14] "ProcessorLib Tutorial: how to implement a simple processor task using the ProcessorLib" - V. Conforti, A. Bulgarelli, M. Trifoglio, F. Gianotti- 25/09/2011, Internal Report IASF Bologna Nr. 613/2012

1.3.2 Documenti di riferimento

[RD.1] Guide to applying the ESA software engineering standards to small projects, ESA BSSC(96) Issue 1 May 1996.

[RD.2] Tailoring of ECSS Software Engineering Standards for Ground Segments in ESA - Part A: Software Engineering, BSSC 2005(1) Issue 1.0 June 2005

[RD.3] ECSS Packet Utilization Standard (PUS), ECSS-E-70-41A, 30 January 2003

[RD.4] W. D. Pence, et al, “Definition of the Flexible Image Transport System (FITS), version 3.0”, A&A, Volume 524, December 2010.

[RD.5] Walmsley, P., “Definitive XML Schema”, Prentice Hall; 2 edition (September 14, 2012)

[RD.6] Armour, F. & Miller, G., “Advanced Use Case Modeling: Software Systems”, Addison-Wesley, Boston, 2001.

[RD.7] Bersanelli, M., et al, A coherent polarimeter array for the Large Scale Polarization Explorer balloon experiment, Proc. of SPIE Vol. 8446 84467C-14

[RD.8] Ken Schwaber, K., Sutherland, J., “The SCRUM Guide, https://www.scrum.org.

2. Executive Summary

2.1 Il Team CIWS

La composizione e l'impegno (FTE) del personale strutturato e non strutturato che ha partecipato al progetto sono stati complessivamente quelli previsti nella proposta. Le variazioni principali rispetto a quanto previsto sono evidenziate in grassetto nelle tabelle qui sotto.

Research Unit 1:

	Name	Institute	Position	1st year (months)	2nd year + **6 months extension** (months)	Contribution
1	Massimo TRIFOGLIO(*)	IASF Bologna	Staff (Tecnologo)	6	6	Project Manager, AIT, Pre-Processing, Database design and implementation
2	Andrea BULGARELLI	IASF Bologna	Staff (Tecnologo)	3	3	System Manager, Software/DB design and Management, Quick Look
3	Fulvio GIANOTTI	IASF Bologna	Staff (Tecnologo)	5	5	System Engineer, h/w configuration, Data Acquisition, Storage, Inter-process API
~~**4**~~	~~**Giuliano TAFFONI**~~	~~**IASF Bologna**~~	~~**Staff (Tecnologo)**~~	~~**3**~~	~~**3**~~	~~**Database design and implementation and VO interfacing**~~
5	Enrico FRANCESCHI	IASF Bologna	TD (Tecnologo)	3	3	ESA-SCOS Interfacing
6	Mauro DADINA	IASF Bologna	Staff (Ricercatore)	3	3	Requirement definition, Application data analysis s/w
7	Massimo CAPPI	IASF Bologna	Staff (Ricercatore)	3	3	Requirement definition, Application data analysis s/w
8	Richard SMART	OA Torino	Staff (Tecnologo)	3	3	Requirement definition, Data Simulation, scientific evaluation

9	Roberto MORBIDELLI	OA Torino	Staff (Tecnico)	3	3	Data-mining, quality assurance and development of end user products
10	**Luciano Nicastro**	**IASF Bologna**	**Staff (Ricercatore)**	**0**	**+1**	**DB software design and development**
11	**Vito Conforti**	**IASF Bologna**	**Borsa di Studio CIWS**	**+3**	**+9**	**Software design and development**
12	**Andrea Zoli**	**IASF Bologna**	**Borsa di Studio CIWS**	**0**	**+8**	**Software design and development**

(*) National and local coordinator

Research Unit 2:

	Name	Institute	Position	1st year (months)	2nd year **+ 6 months extension (months)**	Contribution
1	Fabio PASIAN (*)	OA Trieste	Staff (Astr.Ord.)	3	3	Coordination , system design and testing
2	Claudio VUERLI	OA Trieste	Staff (I Ricercatore)	3	3	Software design and control
3	Roberto CIRAMI	OA Trieste	Staff (Tecnologo)	3	3	System requirements *vs* telescope/instrument control, testing
4	~~**Patrizia MANZATO**~~	~~**OA Trieste**~~	~~**TD (Tecnologo)**~~	~~**6**~~	~~**6**~~	~~**Database design and implementation, DAS development, VO interfaces**~~
5	Marcello LODI	TNG	Staff (Tecnologo)	3	3	Requirement definition, Auxiliary Data Source Interfacing, Prototyping
7	**Andrea Zacchei**	**OA Trieste**	**Staff (Ricercatore)**	**+3**	**+3**	Requirement definition, system design, management

8	**Marco Frailis**	**OA Trieste**	**Staff (Tecnologo)**	**+3**	**+4**	Requirement definition. DAS design, DAS Implementation
9	**Stefano Sartor**	**OA Trieste**	**Borsa di Studio CIWS**	**0**	**+14**	DAS detailed design, DAS Implementation
10	**Paolo Di Marcantonio**	**OA Trieste**	**Staff (Ricercatore)**	**+3**	**0**	Requirement definition, Alma Common Software

(*) Local coordinator

2.2 Addestramento di personale

Come previsto, il progetto si è avvalso di personale in formazione dedicato al progetto. Con il finanziamento ricevuto è stato possibile bandire una Borsa di Studio per OA Trieste e una Borsa di Studio per IASF Bologna.

I tempi richiesti per la ricerca e la selezione sono stati superiori alle previsioni. In compenso il personale selezionato ha dimostrato di avere un ottimo livello di preparazione che ha consentito di fornire in tempi brevi un importante contributo al progetto.

A IASF Bologna si sono avvicendati:

- Vito Conforti, Laurea specialistica in Informatica, che ha partecipato al progetto dal 1/05/2012 al 30/04/2013;
- Andrea Zoli, Laurea specialistica in Informatica, che ha partecipato al progetto dal 3/06/2013 al 4/02/2014.

A OA Trieste la Borsa di Studio è stata assegnata a:

- Stefano Sartor, Laurea in Informatica, che ha partecipato al progetto dal 6/01/2013 al 28/02/2014.

Grazie anche a questa esperienza, dopo il progetto CIWS queste persone hanno ottenuto un Assegno di Ricerca o una Borsa di Studio da altri progetti INAF in svolgimento presso IASF Bologna e OA Trieste.

2.1 Albero del Prodotto

Il progetto CIWS ha l'obiettivo di fornire un sistema software che consenta allo sviluppatore di realizzare "facilmente" la Instrument Workstation da utilizzare per l'archiviazione e la visualizzazione on-line e off-line di dati generati durante le fasi di sviluppo, test e operazioni da strumentazione scientifica di telescopi da terra e di telescopi spaziali.

Dopo una analisi attenta degli scenari di utilizzo si è optato per una soluzione di tipo Framework con architettura modulare che è stato denominato CIWS-FW.

Il CIWS-FW offre allo sviluppatore dei moduli e meccanismi software configurabili. Richiede però allo sviluppatore anche lo sviluppo di codice ad hoc per risolvere esigenze specifiche dello strumento non modellabili a priori.

L'albero del prodotto CIWS-FW è basato sui due sottosistemi principali che costituiscono il software della Instrument Workstation: il Data Processing Subsystem (DPS) e il Data Access Subsystem (DAS). In Figura 2-1è riportato lo schema preso in considerazione in fase di proposta del progetto.

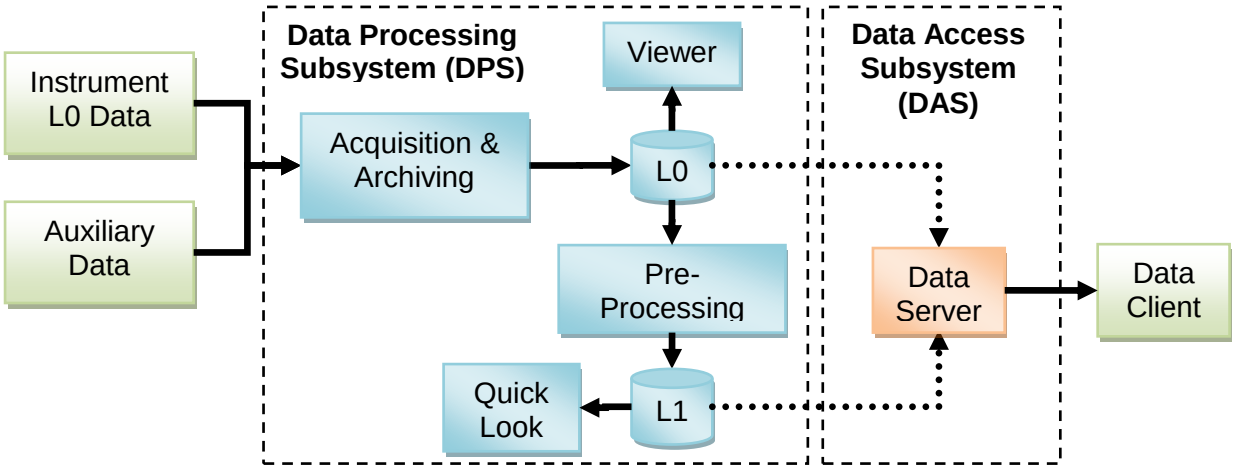


Figura 2-1 Struttura del software della Instrument Workstation definito in fase di proposta

In base a tale schema, è stato definito l'albero del prodotto mostrato in Figura 2-1.

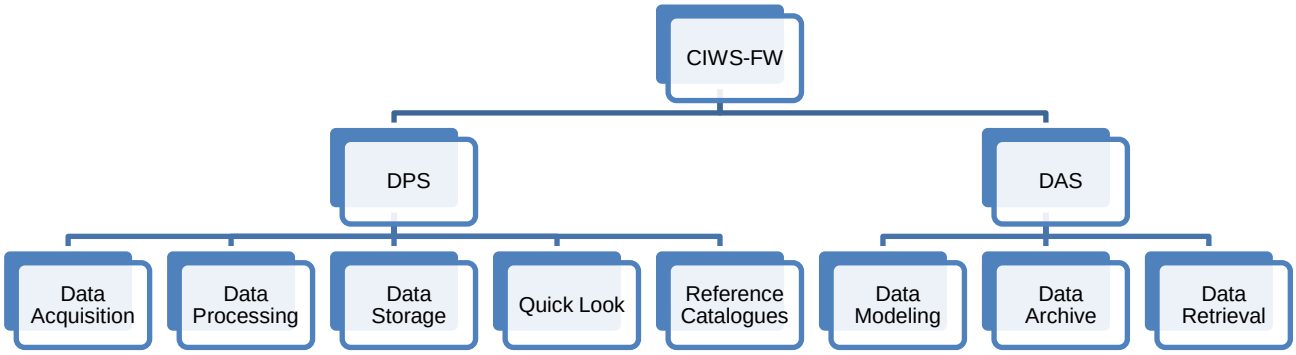


Figura 2-2 Albero del prodotto CIWS-FW

2.2 Organizzazione e svolgimento del lavoro

La distribuzione del lavoro e delle responsabilità ha rispecchiato sostanzialmente quanto previsto in fase di proposta. Come si evince dalla Figura 2-3, i pacchi di lavoro sono stati strutturati in base ai prodotti previsti nelle diverse fasi di sviluppo:

- Definizione dei Requisiti Utente (3 mesi);
- Progettazione (3 mesi);
- Sviluppo (12 mesi);

- Testing e Prototyping (6 mesi).

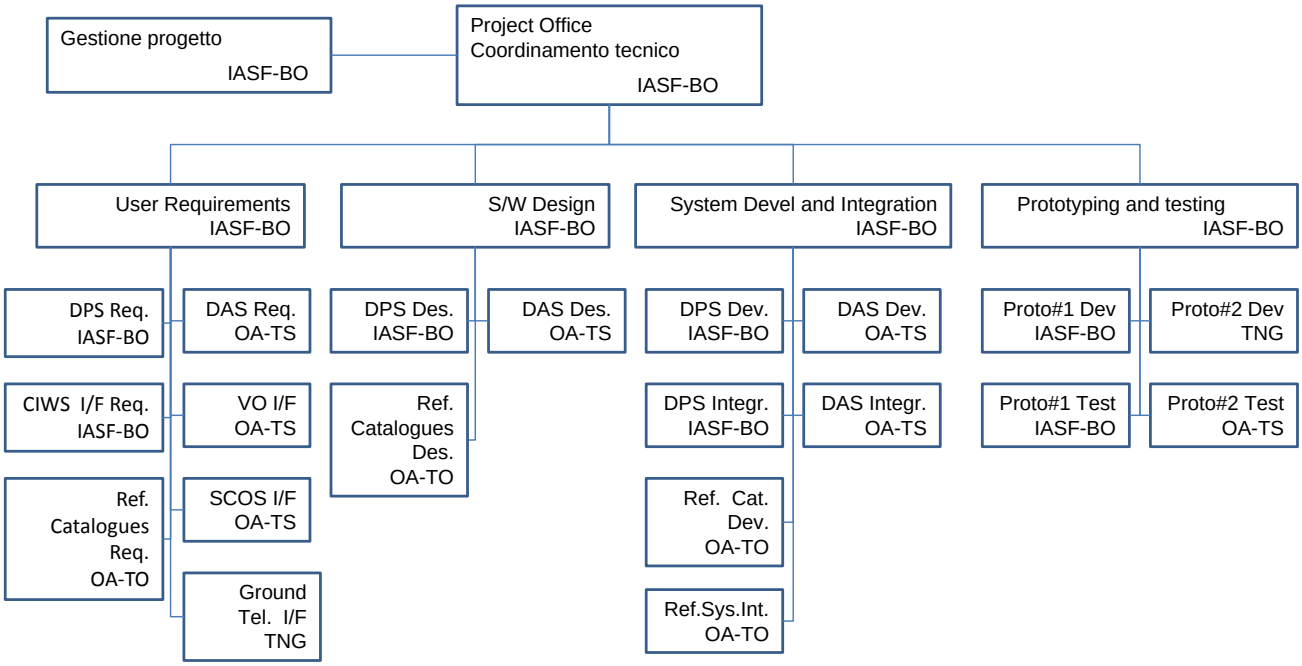


Figura 2-3 Progetto CIWS: distribuzione del lavoro e delle responsabilità

Il lavoro è stato svolto in modo coordinato e distribuito geograficamente presso gli Istituti proponenti:

- INAF / IASF Bologna
- INAF / OA Trieste
- INAF / OA Torino
- INAF / TNG La Palma.

Oltre alle risorse già in dotazione agli Istituti, il team ha beneficiato dell'ambiente di sviluppo distribuito descritto in sezione 3, realizzato con l'aggiunta di risorse hardware e software finanziate con i fondi di progetto.

Sono state svolte 3 riunioni "face-2-face" in occasioni di Milestones interne.

Per il resto il lavoro è stato condotto e svolto da remoto sfruttando le tecnologie disponibili.

1) Sono stati effettuati 24 Progress Meeting utilizzando il sistema di Videoconferenza del GARR con cadenza mensile.
2) È stato utilizzato il software Visual Paradigm (http://www.visual-paradigm.com/) di supporto alla definizione e analisi dei requisiti, alla progettazione e al *coding* del software.
3) È stato messo a punto un *repository Git* (git://ciws.iasfbo.inaf.it/CIWS) per la condivisione e il controllo di configurazione del software prodotto nelle varie fasi di sviluppo e manutenzione del CIWS-FW.
4) Sul sito Redmine di IASFBO (http://redmine.iasfbo.inaf.it) è stata creata una sezione privata del team dedicata alla gestione del progetto (http://redmine.iasfbo.inaf.it/projects/ciws) per
    a. archivio delle minute e della documentazione dei meeting;

b. gestione delle azioni;
c. gestione dello sviluppo software con metodologia AGILE (SCRUM);
d. accesso al repository Git;
e. gestione dei bug software (bug tracking).

5) È stata creata, sullo stesso sito Redmine, una sottosezione pubblica (http://redmine.iasfbo.inaf.it/projects/ciws-fw) che consente ai potenziali interessati all'utilizzo del CIWS-FW di:
   a. accedere alla documentazione;
   b. scaricare il software;
   c. segnalare eventuali problemi/bug riscontrati nell'utilizzo del software;
6) È stato creato il sito http://ciws-fw.iasfbo.inaf.it/ciws-fw/ come "vetrina" per pubblicizzare il CIWS-FW.

Con queste attività sono state create le condizioni per promuovere l'utilizzo del CIWS-FW e fornire supporto agli utilizzatori.

2.3 Metodologia di sviluppo

Nel corso del progetto si è passati da un modello di sviluppo a *"cascata"* ad uno sviluppo di tipo *"iterativo e incrementale".*

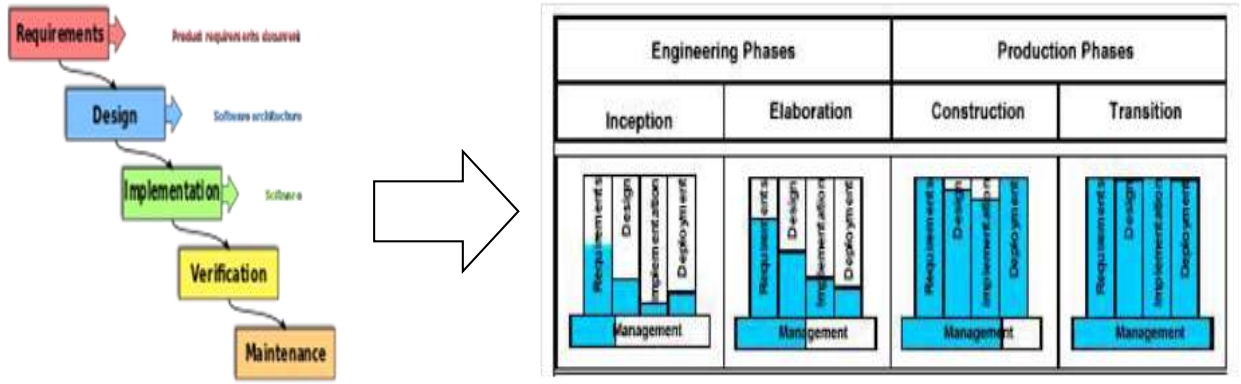


Figura 2-4 Modelli di sviluppo a "cascata" e "iterativo incrementale.

2.4 Pianificazione e realizzazione

L'adozione della modello di sviluppo iterativo e incrementale ha modificato la pianificazione prevista in fase di proposta e riportata in 2.2.

La definizione dei "Requirements" ha subito molte iterazioni che si sono succedute ben oltre i 3 mesi previsti.

Si è partiti cercando di definire i requisiti del prodotto Instrument Workstation "chiavi in mano", solo da configurare. Poi si è arrivati, come anticipato sopra, a optare per un prodotto di tipo Framework, che è configurabile per molti aspetti, ma richiede anche lo sviluppo di software *ad - hoc*.

Di conseguenza, anche le fasi di analisi dei requisiti e di progettazione hanno subito varie iterazioni e slittamenti.

In compenso le restanti fasi sono state anticipate rispetto a quanto previsto dalla pianificazione originaria consentendo di focalizzare gli sforzi sulle componenti principali del CIWS-FW e di conseguire l'obiettivo di realizzare due prototipi che ne fornissero la "*Proof of Concept*".

In alcune fasi dello sviluppo è stata adottata anche la metodologia AGILE Scrum, che è una implementazione dell'approccio iterativo ed incrementale adottato. Scrum non è un processo o una tecnica per costruire prodotti ma piuttosto è un framework all'interno del quale è possibile utilizzare vari processi e tecniche. Vengono utilizzati meccanismi propri di un "processo di controllo empirico", in cui cicli di feedback di diversa durata consentono un controllo di processo a diversi livelli. Ad esempio, lo "sprint planning meeting" consente di prendere decisioni sulle attività da svolgere nei successive 15 giorni e il "daily meeting" permette di correggere eventuali ritardi o criticità di progetto.

2.5 Risultati della ricerca

Come rappresentato in Figura 2-5, il software CIWS-FW realizzato dal progetto CIWS si configura come un insieme di strumenti per la realizzazione del software che consente alla Instrument Workstation di immagazzinare, trasformare, archiviare e visualizzare in tempo quasi-reale i dati generati da strumenti scientifici per telescopi da terra e spaziali durante le fasi di sviluppo, test e operazione.

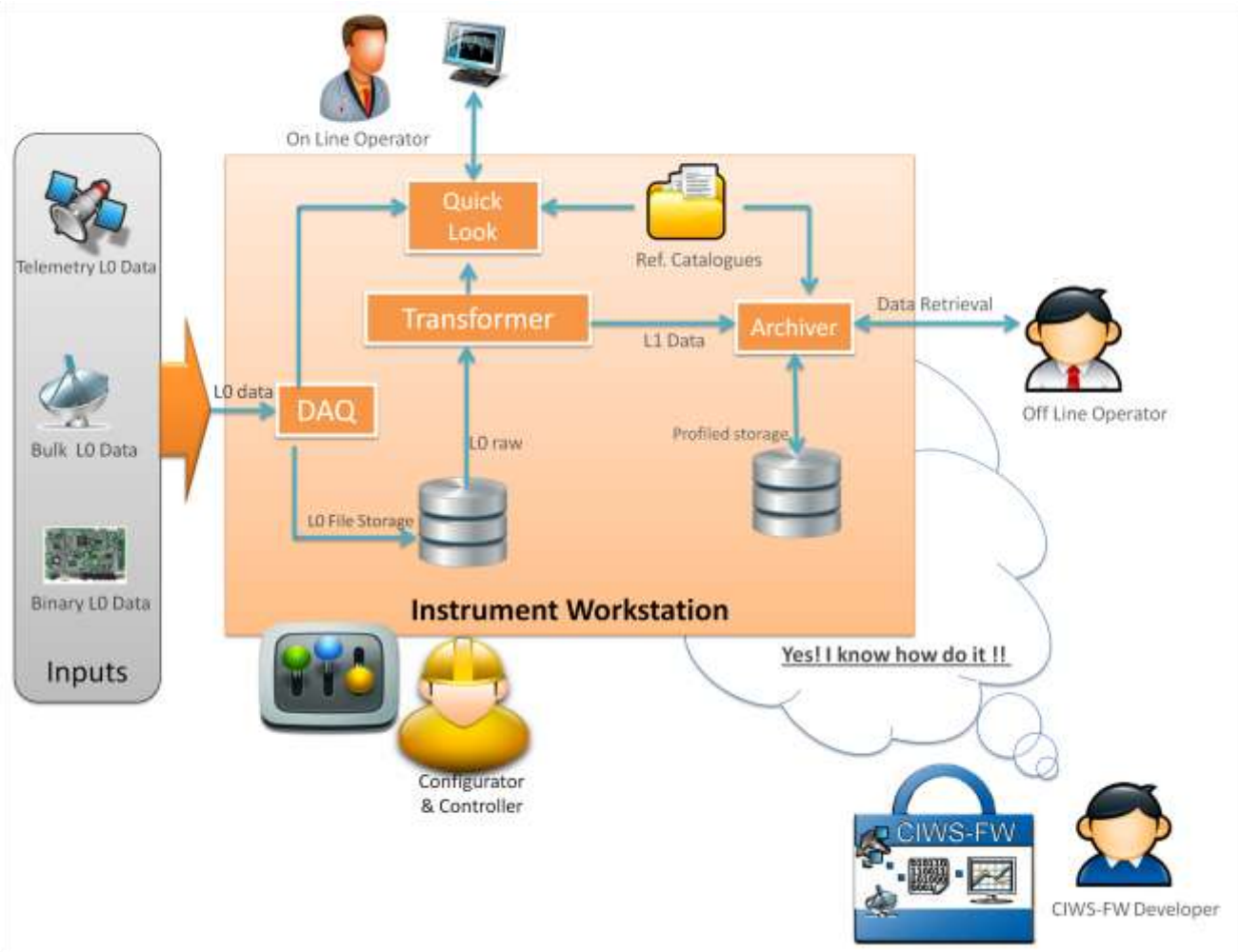


Figura 2-5 Ruolo del CIWS-FW nella realizzazione della Instrument Workstation

Il framework CIWS-FW è modulare ed è costituito dalle componenti mostrate in Figura 2-6.

Utilizzando file XML scritti dallo sviluppatore secondo una determinata grammatica (Data Definition Language – DDL), il CIWS-FW è in grado di modellare il tipo e il formato dei dati definiti dall'utilizzatore per lo specifico strumento scientifico lungo tutto il flusso di elaborazione eseguito dalla Instrument Workstation, e cioè:

- i dati L0 generati dallo strumento.
- i dati L1 generati dai moduli di trasformazione.
- i dati visualizzati dal Quick Look.

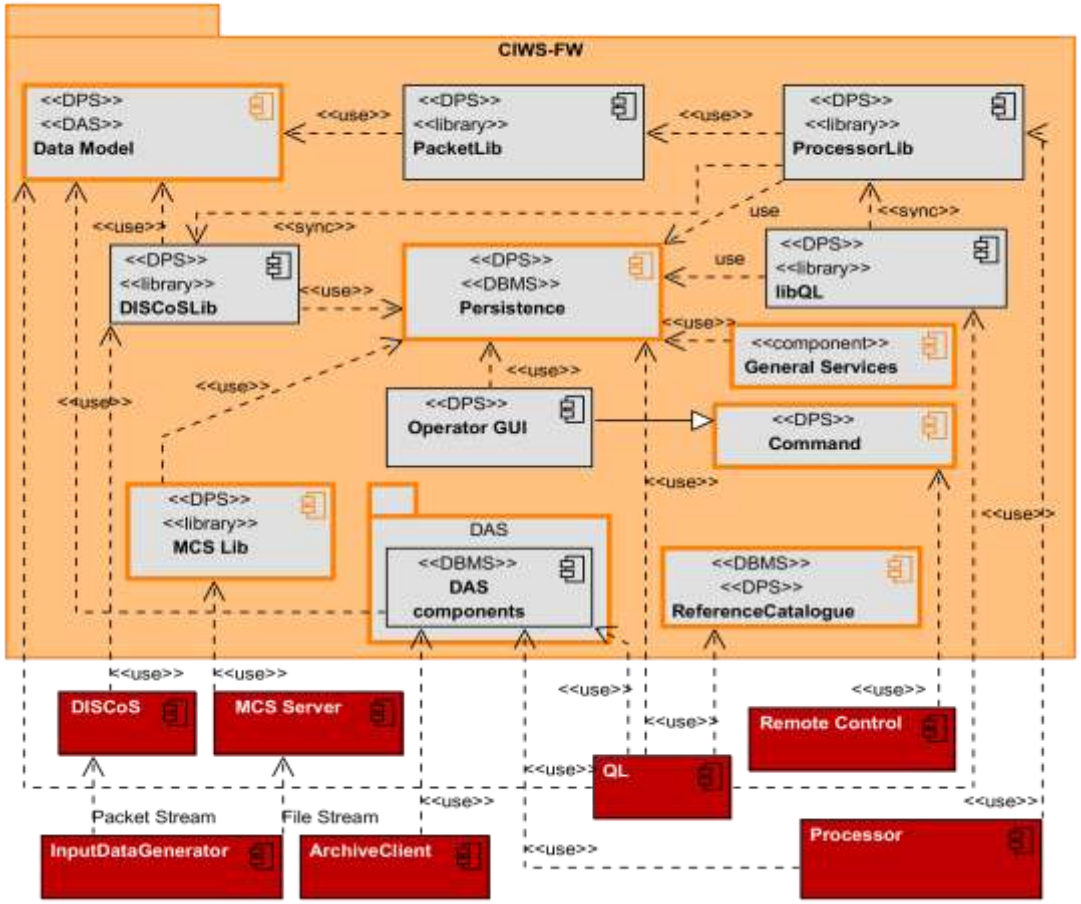


Figura 2-6 Diagramma delle componenti del CIWS-FW.

Il software del CIWS-FW è stato rilasciato nella versione 1.0 scaricabile via WEB insieme alla documentazione di base per lo sviluppatore.

Il software è stato validato sia mediante Unit Test sia mediante la realizzazione di due prototipi di "proof of concept":

1) un prototipo di catena di acquisizione, archiviazione e Quick Look per i dati simulati della Camera di piano focale del costruendo Telescopio ASTRI;
2) un prototipo del Quick Look per i dati del satellite AGILE, dell'Agenzia Spaziale Italiana, in orbita dal 2007.

La versione 1.0 soddisfa la maggioranza dei requisiti previsti originariamente nella proposta. Si tratta comunque di una prima versione che il team intende completare e migliorare nell'ambito di altri progetti INAF che utilizzeranno il CIWS-FW, fra cui:

1) Il progetto ASTRI, che utilizzerà il CIWS-FW per realizzare la *"ASTRI Camera DAQ Workstation"* (per le attività di AIV/AIT di camera) ed il *"ASTRI Camera DAQ Server"* per le operazioni all'Osservatorio di Serra La Nave dove la camera sara' installata nel piano focale del prototipo del Telescopio ASTRI;

2) Il progetto Euclid, che utilizzerà il CIWS-FW per realizzare la "*NISP Instrument Workstation*", prima per le attività di AIV/AIT dello strumento NISP a livello di strumento e di Payload e Satellite, e poi per le attività di throubleshooting presso il Mission Operation Centre del Ground Segment di Euclid.
3) il progetto LSPE (Large Scale Polarization Explorer, vedi [RD.7]), che utilizzerà il CIWS-FW per l'archiviazione e visualizzazione dei dati che verranno acquisiti dagli strumenti.

L'obiettivo sarà sicuramente raggiunto considerato che membri del team sono direttamente coinvolti con compiti di responsabilità nella realizzazione di queste applicazioni. Inoltre il progetto è stato un'opportunità per continuare lo sviluppo di strumenti software di interesse per le attività collegate alla messa in linea, con accesso pubblico, dei cataloghi astronomici gestiti presso l'Osservatorio di Torino. Infatti sono stati sviluppati sia un tool stand-alone che una libreria API che permettono l'interrogazione dei cataloghi dell'OATO da qualsiasi programma, scritto in qualsiasi linguaggio. L'API, scritta in C++, è utilizzabile ma non ancora completata. Sia questa che il programma stand-alone (utilizzabile da sistemi UNIX) sono open source e scaricabili dal sito CIWS. Queste ulteriori attività di sviluppo renderanno il *prodotto* CIWS-FW ancora più "robusto" e facilmente sfruttabile anche da utilizzatori che non hanno partecipato allo sviluppo della versione 1.0.

2.5.1 Partecipazioni a Conferenze Internazionali

Presentati i seguenti lavori alla XXIII conferenza dell'ADASS (Astronomical Data Analysis Software & Systems):

1) V. Conforti, et al., "CIWS-FW: a Customizable Instrument Workstation Software Framework for instrument-independent data handling", 2013 (in press).
2) M.Frailis, et al., "*DAS: a data management system for instrument tests and operations*", 2013 (in press).

Il primo è relativo all'architettura generale del CIWS-FW. Il secondo fornisce dettagli del sottosistema DAS.

2.5.2 Note tecniche e rapporti interni

Nel corso del progetto sono stati prodotti documenti di requisiti e di design del CIWS-FW, manuali di programmazione e di riferimento delle librerie e dei tools che costituiscono il prodotto CIWS-FW.

Tutti questi documenti sono elencati in sezione 1.3.1.

2.5.3 Sito WEB

La scopo di un framework è quello di supportare le attività di progettazione e sviluppo software in modo che ci si possa concentrare maggiormente sugli aspetti di business logic. Ogni framework ha delle caratteristiche proprie che lo rendono adatto ad una certa tipologia di progetti. Diventa quindi essenziale sponsorizzare il prodotto in modo che possa essere visibile dal maggior numero di potenziali interessati. Occorre inoltre supportare gli sviluppatori che abbiano già deciso di adottare il framework.
Sono stati quindi creati due siti web: il primo offre una vetrina del CIWS -FW a tutti i potenziali interessati, mentre il secondo offre tutte le funzionalità necessarie agli utilizzatori del CIWS-FW.

2.6 ciws-fw.iasfbo.inaf.it/fw

Il sito è stato installato su un server dedicato al progetto CIWS e sito nel centro calcolo di IASFBO. La creazione del sito web è stata supportata dall'uso del CMS WordPress che offre diverse

funzionalità tra cui la gestione degli utenti e dei contenuti, il supporto al layout grafico e la compatibilità con gli standard del W3C (World Wide Web Consortium).

Il sito è organizzato in 5 sezioni. Nella home page è redatta una breve descrizione del CIWS-FW. Nella pagina WHO viene presentato il team del CIWS-FW (Figura 2-7).

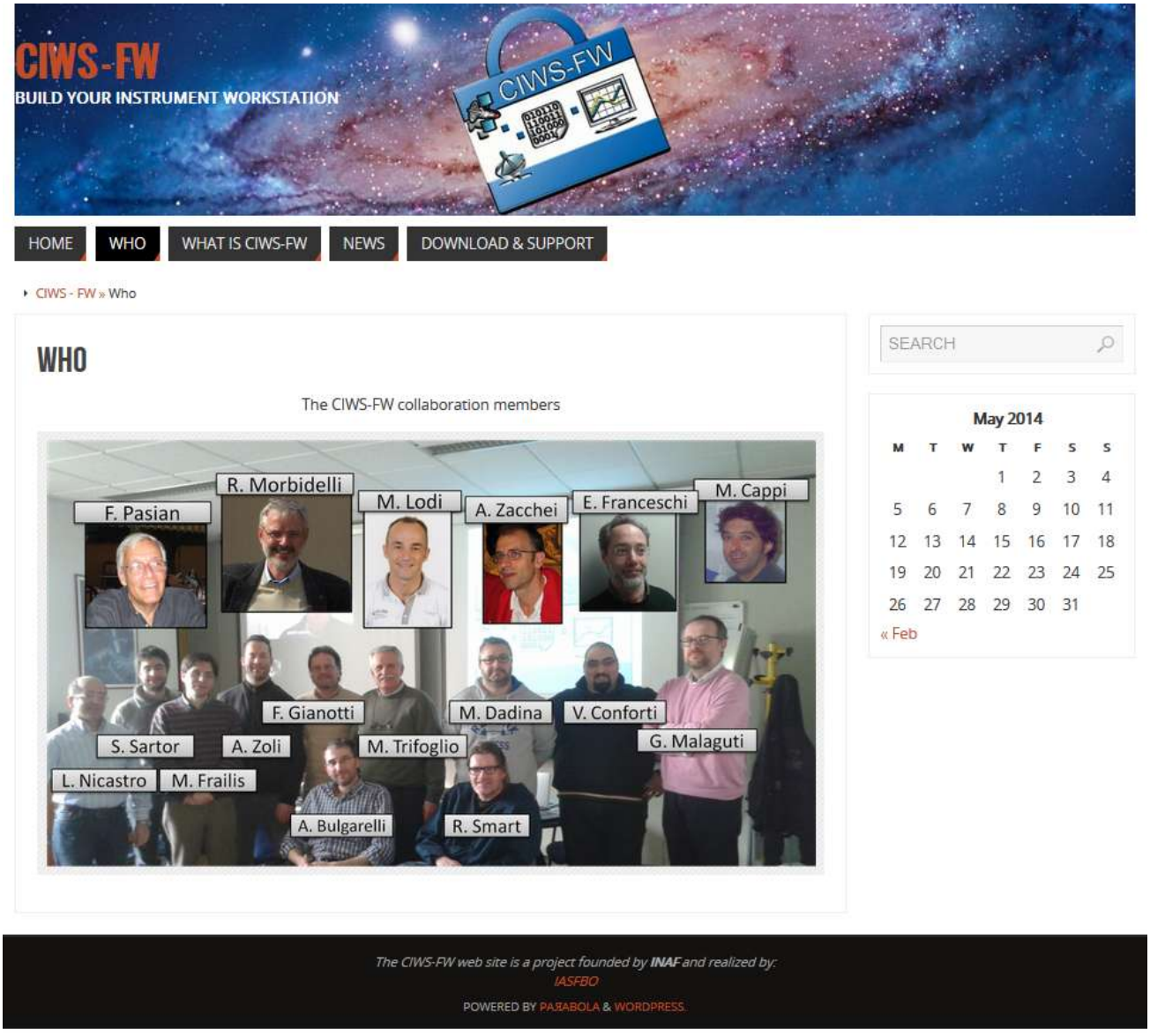


Figura 2-7 Il sito CIWS-FW per la divulgazione.

Le caratteristiche del CIWS-FW sono mostrate nella sezione WHAT IS CIWS-FW. La pagina delle NEWS riporta le notizie salienti mentre la pagina di DOWNLOAD & SUPPORT rimanda al sito del Redmine.

2.7 http://redmine.iasfbo.inaf.it/projects/ciws-fw

Il sito è stato realizzato con Redmine (ww.redmine.org) ed è dedicato agli utenti che vogliono utilizzare il CIWS-FW per le proprie applicazioni. Come mostrato in Figura 2-8, il sito contiene

sezioni dedicate al segnalazione di problemi riscontrati dall'utente ("New Issue") e alla loro gestione ("Issue"), alla documentazione tecnica, all'uso dei tutorial (Wiki). La sezione "Repository" consente all'utente di accedere ai repository Git del software CIWS-FW.

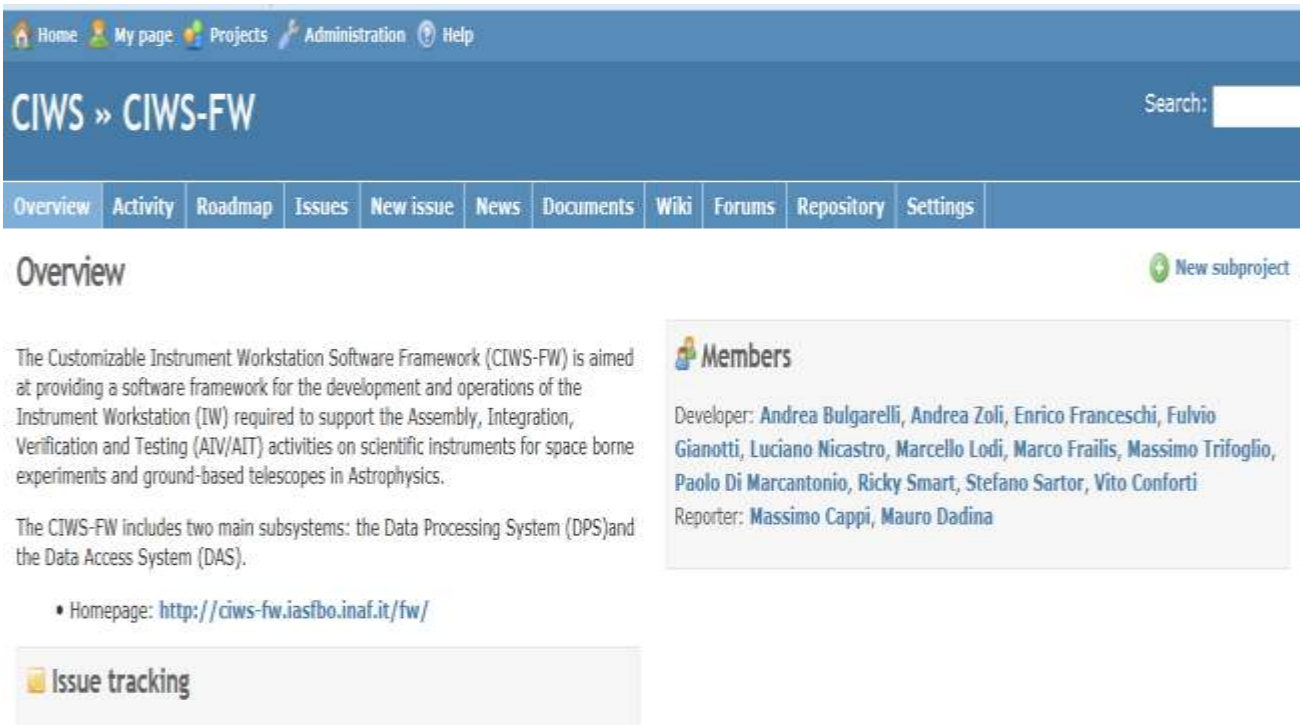


Figura 2-8 Il sito CIWS-FW per il supporto agli utilizzatori

3. Ambiente di sviluppo

3.1.1 Infrastruttura hardware e software

La strumentazione acquisita con il progetto CIWS e l'utilizzo di risorse già in dotazione hanno consentito di realizzare e mettere in opera presso INAF IASF Bologna l'infrastruttura per lo sviluppo, l'integrazione e il test del software di progetto, la cui configurazione iniziale è mostrata in Figura 3-1 è ha richiesto l'acquisto del server ciws02. Con questa configurazione sono stati messi in opera i servizi di supporto alla comunicazione e interscambio richiesti durante le prime fasi del progetto, rivolte principalmente alla definizione dei requisiti, e sono state poste le basi per le successive attività di sviluppo software.

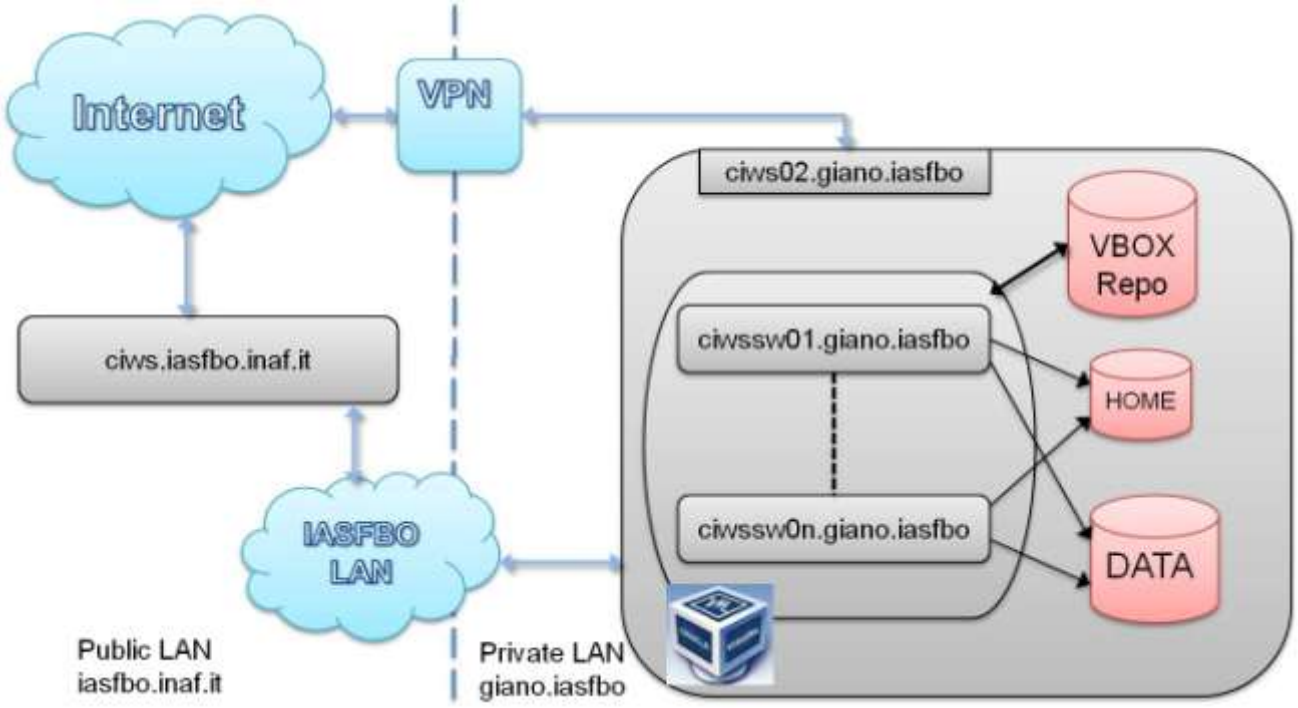


Figura 3-1Configurazione iniziale della struttura ICT del progetto CIWS

Sul server cws01 sono stati installati e configurati i servizi Web (Redmine, sito Web, ftp), i sistema per il controllo di configurazione del software (Git e SVN). Il server ciws02 è stato dedicato allo storage (home utenti, dati di test) e ad un primo tentativo di utilizzare tecniche di virtualizzazione utilizzando il sistema VirtualBox. Il sistema ha consentito di realizzare macchine virtuali configurate per il progetto CIWS che sono state esportate e utilizzate presso gli altri Istituti della collaborazione per svolgere le prime fasi di sviluppo del software. La soluzione adottata si è rivelata inadeguata sia dal punto di vista funzionale che dal punto di vista delle prestazioni. Per superare questi limiti sono stati acquistati altri due server (OVMS1 e OVMS2) con i quali è stato possibile realizzare un sistema di virtualizzazione, basato sul sistema OracleVM (Figura 3-2), ottenendo così la configurazione finale del sistema ICT del progetto CIWS mostrata in Figura 3-3, che è stata utilizzata nelle fasi conclusive di integrazione e test e che si prevede di utilizzare per le future attività di manutenzione e miglioramento del software CIWS.

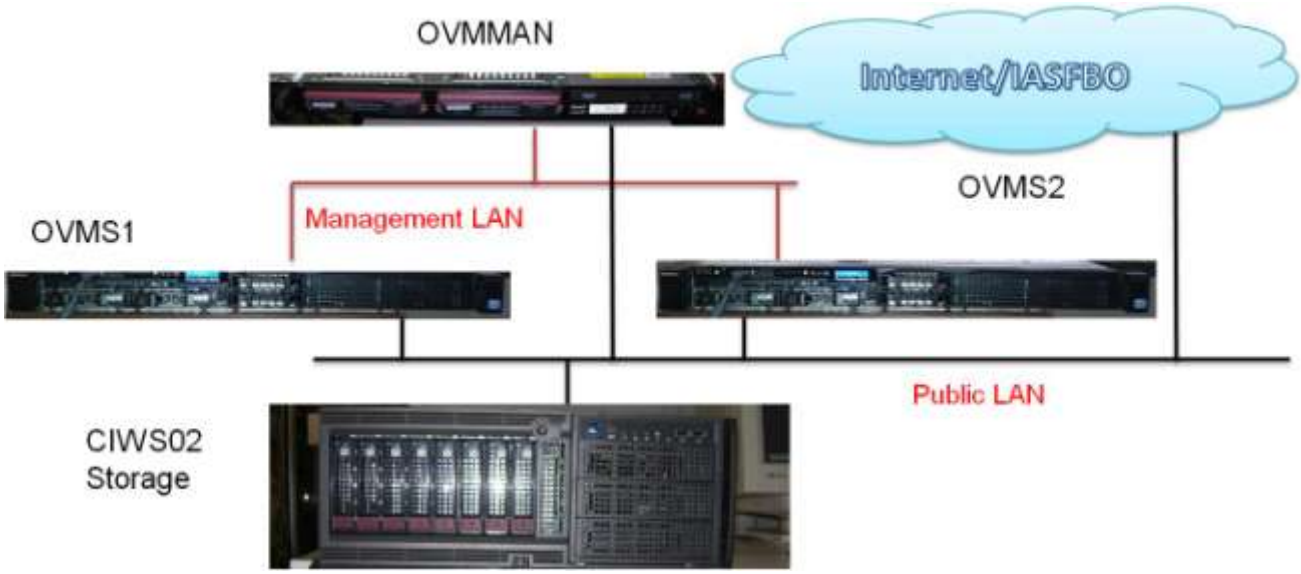


Figura 3-2 Configurazione del sistema di virtualizzazione OVM CIWS

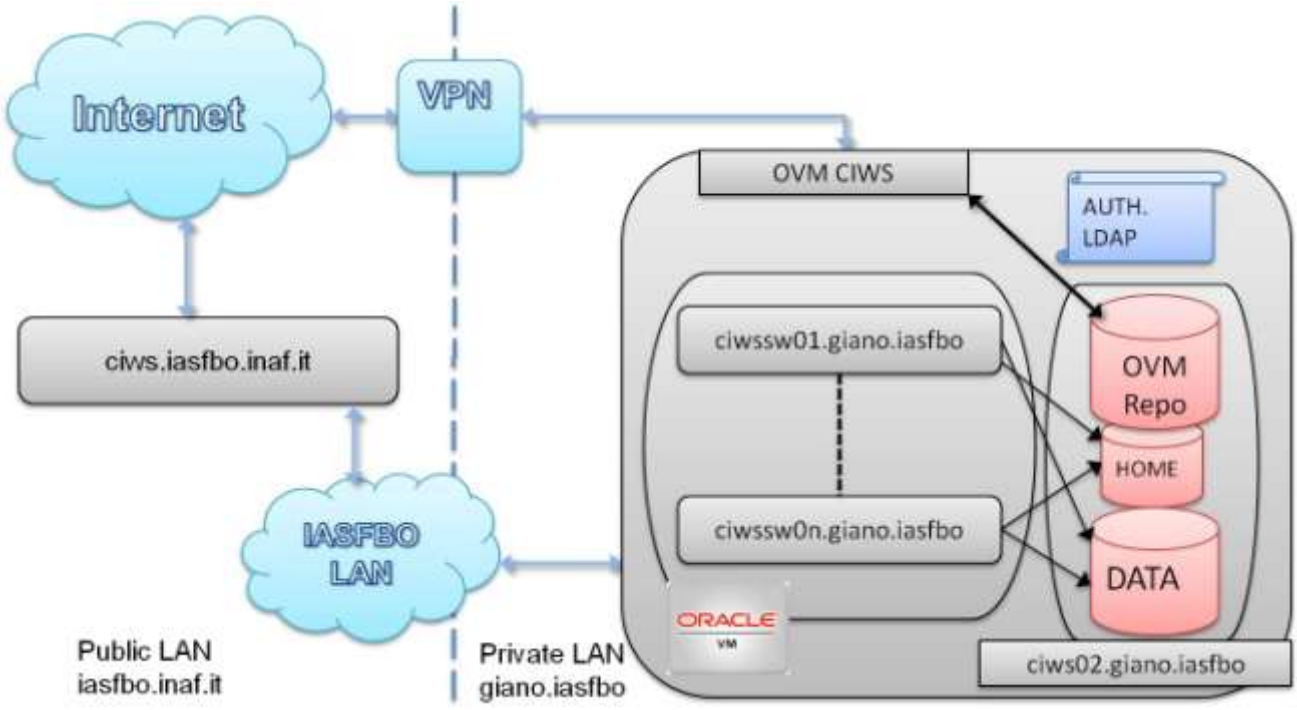


Figura 3-3 Configurazione finale della infrastruttura ICT del progetto CIWS

3.1.2 Selezione del case tool

Durante le prime fasi del progetto è stato selezionato il case tool, ampiamente utilizzato poi in tutte le successive fasi, dalla raccolta dei requisiti al design del software e al reverse engineering di software esistente. Dopo una iniziale panoramica dei principali prodotti esistenti, sono stati selezionati e messi a confronto nel dettaglio seguenti due prodotti commerciali:

a) MagicDraw UML (MD)
b) Visual Paradigm (VP)
c) Enterprise Architect.

Nel dettaglio, sono stati valutati i seguenti punti:

1) disponibilità di una versione gratuita non limitata nel tempo: VP ha una Community version che permette di disegnare tutti i diagrammi e di leggere i progetti fatti anche con version superiori. MD ha solo una trial version ed un reader

2) round-trip engineering del codice sorgente (importazione corretta, esportazione di commenti, metodi e attributi)

3) generazione di report: MD supporta la generazione di report in tutte le versioni, VP a partire dalla standard

4) numero di linguaggi supportati: VP supporta sia Java sia C++, MD supporta solo C++

5) collegamento e reverse engineering di databases: MD l'ha solo nella versione Enterprise, VP l'ha nella versione professional. Comunque VP ha la possibilità di disegnare schemi ER anche nella versione free.

6) requirement capturing: VP lo supporta già nella versione FREE, nelle versioni successive supporta anche il flow of events degli Use Cases e il prototyping di GUI

7) design review and commenting: VP lo supporta, MD lo supporta

8) team collaboration: VP lo supporta acquistando il server a parte. Anche MD lo supporta con server a parte

9) integrazione con ambienti di sviluppo: entrambi si integrano con Eclipse Workbench, ma solo per il linguaggio Java

10) supporto: VP rispondono subito alla richiesta di supporto (ad esempio per il problema del reverse engineering). Avviata richiesta di supporto a MD, non ricevuta nessuna risposta.

11) integrazione con processo di sviluppo12) Enterprise Architect è stato valutato sia per il Reverse Engineering che per il supporto all'UML. Su MAC e Linux puo' essere eseguito solo in ambiente Wine.

Per i motivi detti sopra la decisioni finale è ricaduta su Visual Paradigm.

3.1.3 Infrastruttura dedicata ai Reference Catalogues

Il contenuto dei cataloghi astronomici di riferimento utilizzati per il progetto e l'archivio delle immagini è sostanzialmente legato, in maniera consistente, al database "Compass" utilizzato per la realizzazione del II Guide Star Catalog (GSCII); al catalogo GSCII, a diversi altri cataloghi ancillari e all'archivio delle immagini utilizzato per la realizzazione del database stesso. I dati ai fini del progetto sono in parte collocati su H/W preesistente ed in parte sono stati ricollocati su H/W appositamente configurato e posto in opera con fondi del progetto CIWS. L'elenco che segue riporta le specifiche tecniche delle tre componenti utilizzate e la componente H/W su cui è ospitato l'archivio preesistente.

	Descrizione	Uso	Immagine
1	**Gateway DT50 (CPU 1) <u>H/W preesistente a supporto del progetto</u>** SO: Windows 7 Professional 64-bit, Intel Core i3 550 @ 3.20GHz, RAM: 12,0 GB, Grafica 1600x900, HD ~10 TB.	Web server **(preesistente)** (Database immagini)	
2	**Gateway DT50 (CPU 1) <u>H/W preesistente a supporto del progetto</u>** SO: Linux 64-bit, Intel Core i3, RAM: 12,0 GB, Monitor 1600x900, HD: 1863GB	DB WS **(preesistente)** (DB e Catalogues mirror)	
3	**HP Elitebook 8600 P** S. O.: Windows 7 Professional 64-bit SP1, Intel Core i7, RAM 8 GB, Dischi 238GB SSD + 931GB HD	Portable W/S (**fondi CIWS**) (Stazione operativa)	
4	**MacBook Pro 15"** Intel Core i7 quad-core a 2,0GHz , 8GB 1333MHz DDR3 SDRAM - 2x4GB, Disco 750GB, SuperDrive 8x (DVD±R DL/DVD±RW/CD-RW)	Portable W/S (**fondi CIWS**) (Stazione operativa)	
5	**Tablet ASUS TF700/10 Quadcore**	Mobile Client (**fondi CIWS**)	
6	**E7214: 2UR 2x Xeon E5620 2,40Ghz – 12GB – 1x 250GB**	DB Catalogues (**preesistente**)	

3.1.4 Infrastruttura dedicata al Data Access System (DAS)

Lo svolgimento presso OATS delle attività di sviluppo e test del software di archiviazione e gestione dei dati ha richiesto l'acquisto di materiale informatico per la messa in opera di un server con capacità di calcolo e storage adeguate allo scopo.

Con i fondi CIWS è stato acquistato uno storage server Supermicro con Processore Intel Core i5-3550 da 3.3 GHZ, 16GB ram DDR3, disco SSD 128GB, Controller Raid PCI-E a 16 canali SataIII in raid 5 o 6 con un corrispondente numero di HD da 2 TB (nominali e spare).

4. Definizione dei requisiti

Come detto, il CIWS si propone di essere un framework (CIWS-FW) in cui gli sviluppatori di s/w per futura strumentazione astronomica possano trovare un importante e solida base per il data handling e, al contempo, veder semplificato il loro lavoro.

L'attività' di definizione dei requisiti è stata impostata facendo sostanzialmente riferimento alle linee guida definite da ESA per i piccoli progetti software [RD1] e ha avuto il risultato di generare il primo documento previsto dal ciclo di vita del software, ossia il documento "CIWS User Requirements Document" [DD.1].

Il documento fornisce una descrizione generale delle caratteristiche del CIWS-FW e ne dettaglia i casi d'uso e i requisiti funzionali e non funzionali che sono stati identificati mantenendo come riferimento costante due direttrici:

1) cercare la più grande generalità di impostazione del s/w così da renderlo trasportabile a tutti o quasi gli esperimenti astronomici e in tutte o quasi le fasi di sviluppo degli stessi;
2) cercare di prevedere le possibili necessità dell'utilizzatore finale nei diversi scenari possibili così da inserire fin da subito funzionalità che spesso non sono facilmente riscontrabili nelle console scientifiche degli strumenti.

La caratteristica principale richiesta è la configurabilità e l'adattabilità' agli scenari in cui deve operare la Instrument Workstation utilizzata dal team di sviluppo dello strumento per verificare il suo corretto funzionamento durante le fasi di sviluppo e di operazione.

Questa caratteristica è relativa in particolare alle modalità e al formato con cui la Instrument Workstation riceve i dati dello strumento, li processa, li visualizza e li archivia localmente. Sono state considerate due modalità principali. La prima è tipica degli strumenti per telescopi spaziali, dove i dati dello strumento sono ricevuti tramite una socket TCP/IP e hanno un formato a pacchetto con standard ECSS [RD.2]. La seconda è normalmente utilizzata negli Osservatori da terra e prevede che la Instrument Workstation riceva per ogni osservazione uno o più file con formato FITS.

Per le attività da svolgere per integrare e operare lo strumento a livello di sistema sono stati presi in considerazione due sistemi di riferimento fondamentali per il s/w astronomico: ESA SCOS 2000 per gli esperimenti spaziali e ESO ALMA Common Software (ACS) per gli osservatori da terra. Il primo pone dei requisiti sulla capacità di esportare e importare il Data Base che definisce i parametri per il comando e il controllo dello strumento e per la decodifica dei dati. Il secondo impone specifiche modalità di interfacciamento per potere realizzare un sistema centralizzato di comando e controllo dei sottosistemi che partecipano alle operazioni.

Dopo varie iterazioni la discussione sui requisiti si è concretizzata con la richiesta al CIWS di sviluppare un framework che rendesse agile:

1) modellare i dati sorgente in formato raw (L0 data);

2) modellare il passaggio dei dati da L0 a L1 e la loro archiviazione secondo le necessità del progetto;

3) implementare s/w in grado di operare da locale o remoto messaggi o comandi per l'avvio delle misure;

4) implementare un s/w in grado di collezionare e catalogare i metadata relativi alle misure fatte;

5) implementare un s/w DAQ in grado di acquisire e immagazzinare i dati L0 data and i relativi metadata;

6) implementare un s/w di trattamento dei dati in modo da generare dati L1 sulle specifiche del modello dell'utente;

7) implementare software per archiviare e richiamare dati L0 e L1 data e i metadata relativi utilizzando il modello creato dall'utente;8) sviluppare una GUI per controllare i processi control and monitor elencati nei punti 3), 4) e 5);

9) implementare un Quick Look software che permetta all'operatore di plottare display in “real time” i dati L0 e L1;

10) implementare un s/w di Quick Look in grado di accedere a Reference Catalogues per ottenere informazioni su oggetti stellari in “near real time”;

11) implementare un software che permetta agli operatori di maneggiare i dati quando off-line;

12) implementare un software per accedere ai sistemi di misura;

13) implementare una GUI per accedere alle misure.

Come detto sopra, la definizione dei requisiti è stata fatta considerando sia lo sviluppatore in ambiente CIWS-FW che l'utente finale. Di quest'ultima famiglia, si è assunto che potessero far parte sia gli ingegneri addetti al test o al mantenimento della strumentazione, sia scienziati con una profonda o limitata esperienza sullo strumento. Come detto, abbiamo poi pensato ad una Instrument Workstation che potesse essere usata durante le fasi AIT/AIV, durante il funzionamento operativo a terra o in volo. Tutto questo si è trasformato in una mole di richieste e di scenari ipotetici importante.

Come mostrato in Figura 4-1 e Figura 4-2, si richiede alla Instrument Workstation di gestire le misure secondo una gerarchia organizzata in piani di misura, sessioni di misura e blocchi di misura. La combinazione degli identificatori associati a ciascuna di queste entità consentirà alla Instrument Workstation di identificare univocamente ciascuna misura durante tutte le operazioni di archiviazione e ricerca.

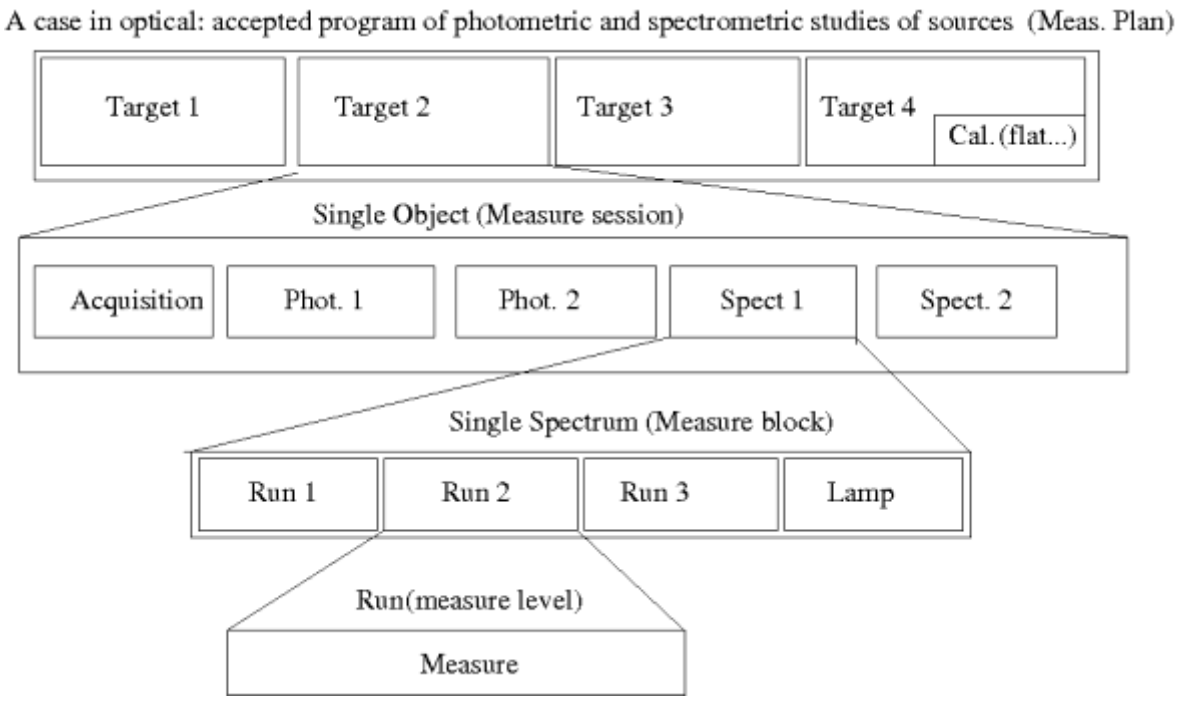


Figura 4-1Scenario di acquisizione dati nel caso di un telescopio ottico

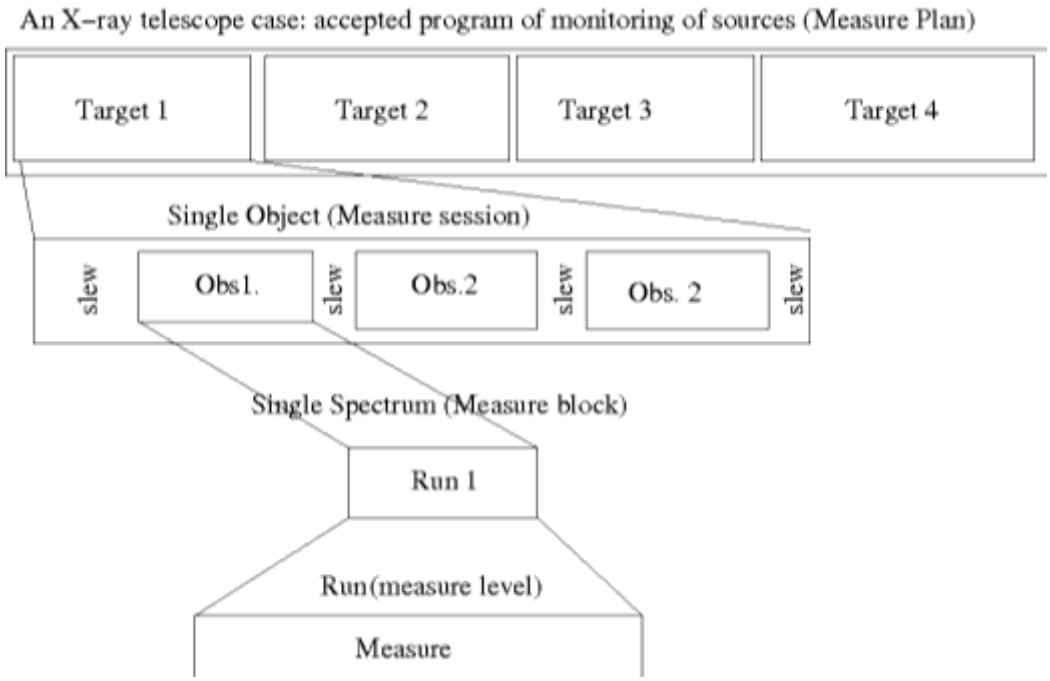


Figura 4-2 Scenario di acquisizione dati nel caso di un telescopio spaziale.

I casi d'uso principali relativi alla acquisizione, archiviazione, elaborazione e visualizzazione delle misure sono mostrati in Figura 4-3.

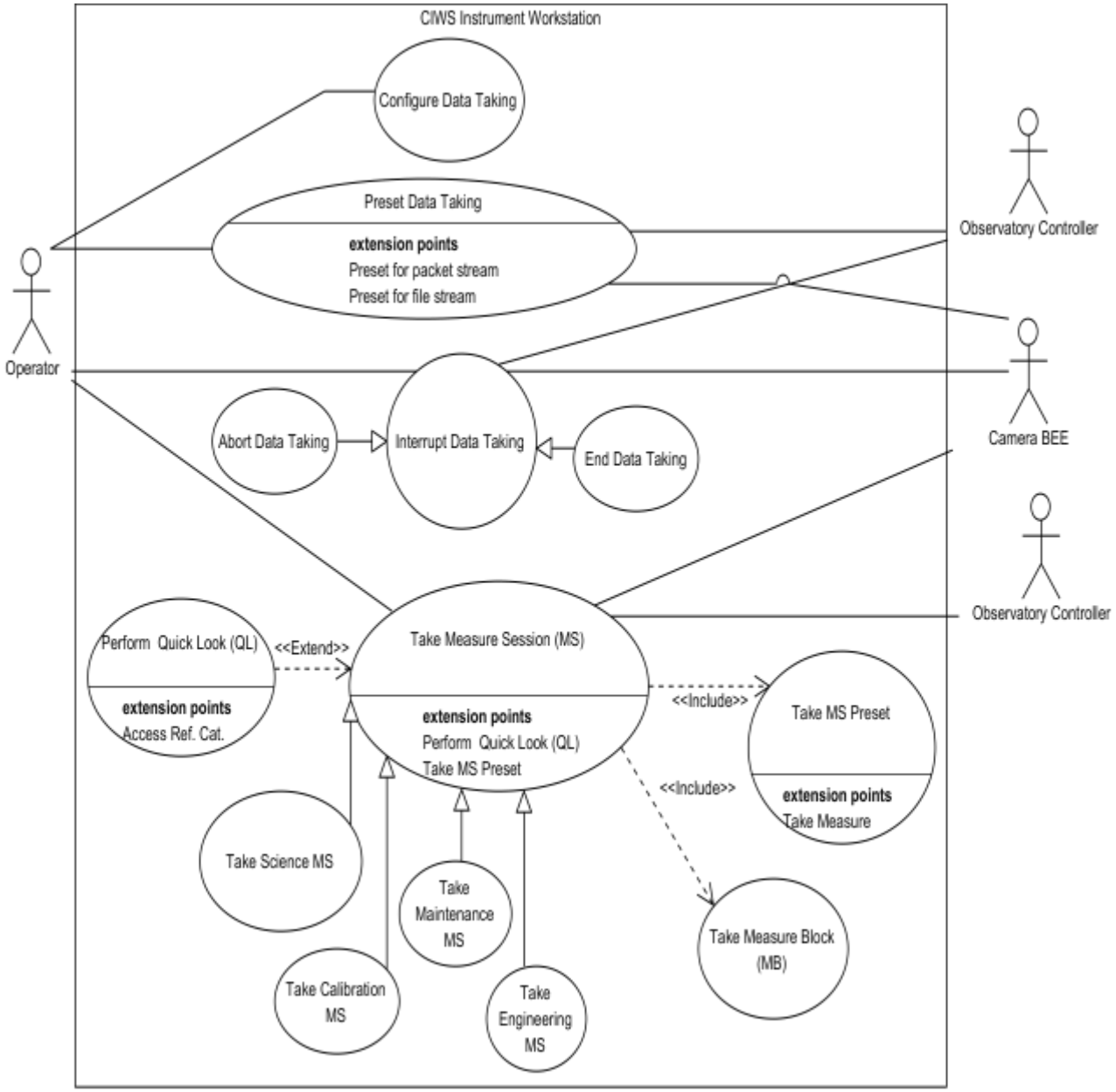


Figura 4-3 Diagramma di alto livello dei casi d'uso dell'acquisizione delle misure.

Prima di iniziare una nuova sessione di misura relativa a un determinato piano di misure, l'operatore configura il software e inizializza i servizi richiesti per l'acquisizione dei dati e per le eventuali comunicazioni con il sistema che coordina le operazioni a livello di sistema. La sessione di misura può essere di tipo scientifico, calibrazione, manutenzione o ingegneristico e, come detto, è costituita da uno o più blocchi di misure, eventualmente preceduti da una fase preliminare di "*Preset*" del telescopio.

I casi d'uso principali previsti per ciascun blocco di misure (MB) sono mostrati in Figura 4-4. Possono essere blocchi semplici costituiti da una successione di misure, oppure blocchi composti da blocchi semplici da eseguire dopo una operazione di offset del telescopio.

Nell'ambito di ciascuna misura si può distinguere fra l'insieme dei dati generati dallo strumento durante la sua configurazione o attesa (dati di modo *IDLE)* e quello dei dati generati durante l'acquisizione dal rivelatore (dati di modo *RUN*). Su ciascuno di questi insiemi si prevedono operazioni di trasformazione e di archiviazione locale o remota, corredata da eventuali dati ausiliari forniti dal sistema che coordina le operazioni a livello di sistema.

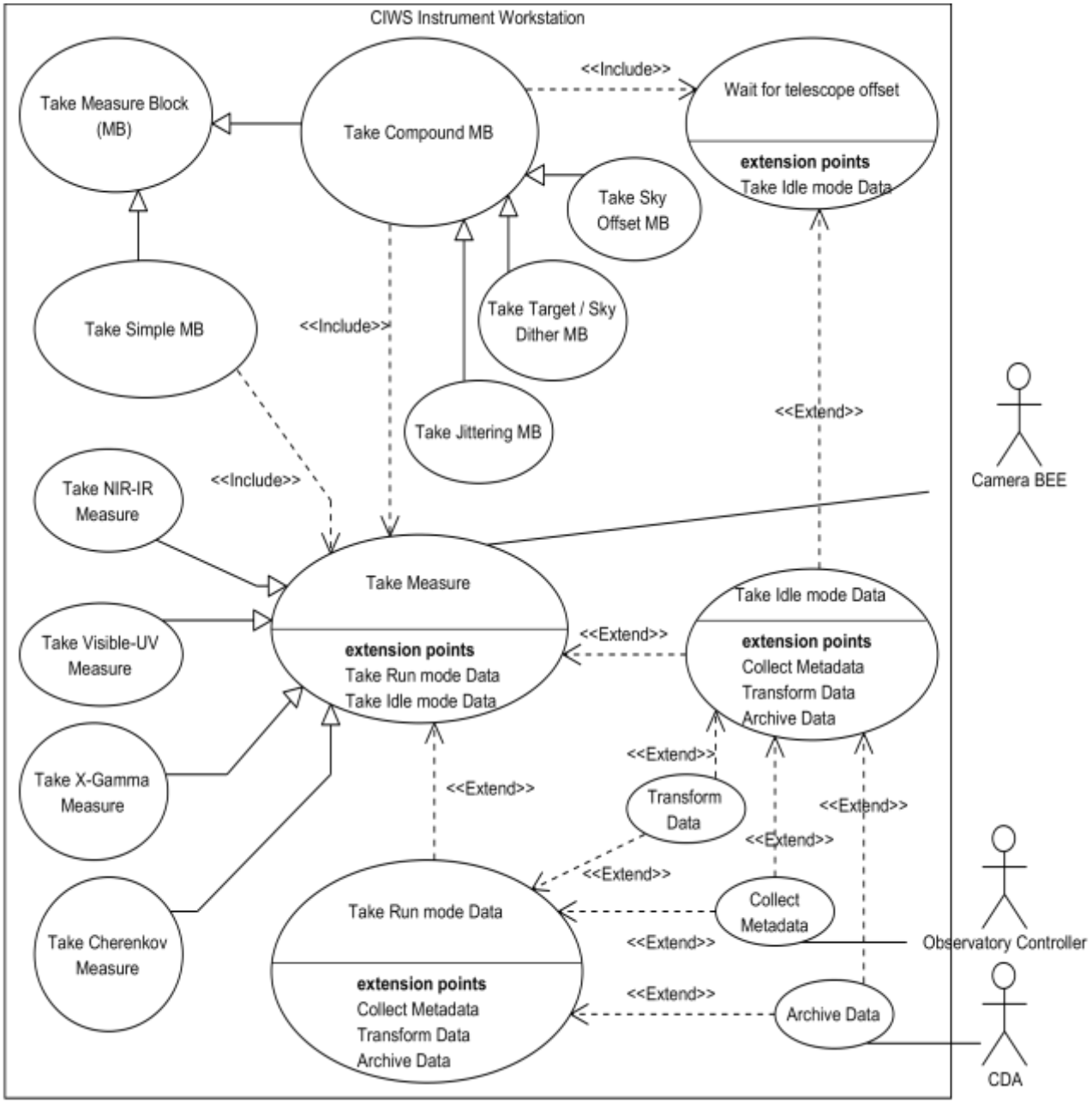


Figura 4-4 Diagramma dei casi d'uso dell'acquisizione di un blocco di misure

5. Analisi e progettazione

Il progetto CIWS si è ispirato allo standard ESA PSS per la documentazione ([RD.1], [RD.2]). ESA PSS fornisce un ottimo framework per la documentazione, ma non fornisce indicazioni specifiche per l'analisi, la progettazione e lo sviluppo del software. Per quanto riguarda il processo di sviluppo adottato, questo è già stato descritto nella sezione 2.3.

La progettazione del software si è ispirata al *4+1 Logical View Model*, che guida il processo dalla definizione dei requisiti fino alla definizione di diverse "viste" dell'architettura.

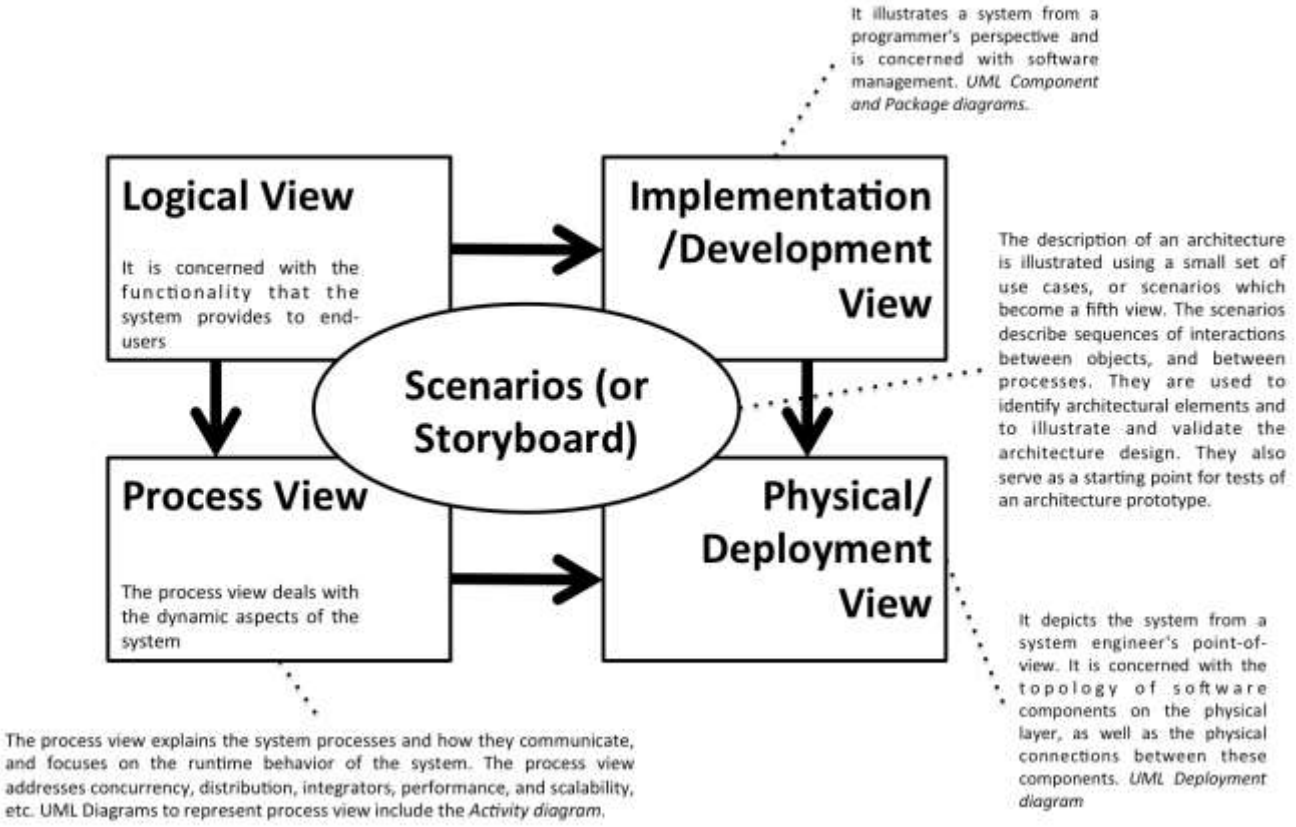


Figura 5-1 4+1 Logical View Model

Con riferimento alla figura sopra, la vista centrale è lo scenario (o Use Case model in un contesto *Unified Process* oppure una Storyboard in un contesto SCRUM, come quello adottato per il progetto CIWS). La definizione degli scenari, contenuta nell'URD del CIWS ([DD.1]) è il punto di partenza per la definizione delle altre viste, che invece sono dotate di una sequenza e precedenza logica:

- La *Logical View* (o vista logica) descrive gli aspetti funzionali del sistema, identificando i blocchi funzionali principale e le loro relazioni. Una descrizione completa del modello logico del CIWS è presente nella sezione 2 del CIWS SSD [DD.2], che è anche l'input fondamentale per la definizione dei requisiti funzionali e non funzionali (sezione 3 del CIWS SSD [DD.2] e DAS SSD [DD.3]). Per la descrizione logica relativa ai database utilizzati si è utilizzato anche il modello Entity/Relationship e il modello relazionale.
- La *Implementation/Deployment View* (o vista di implementazione), descrive la parte statica del sistema software realizzato e che può essere rappresentata da UML *Component*

Diagram. È presentata nella sezione 4 dei documenti di design ([DD.1], [DD2]). Descrive il sistema dal punto di vista dello sviluppatore e consente di soddisfare i requisiti utente e i requisiti software definiti nelle due viste precedenti.

- La *Process View* si focalizza sull'aspetto dinamico del sistema (ad esempio, come avviene la comunicazione tra processi).
- La *Physical/Deployment View* descrive come i diversi componenti e processi software sono allocati sui diversi nodi di elaborazione e la topologia dei diversi componenti software.

5.1 CIWS-FW

Come mostrato in Figura 5-2, Il CIWS-FW è composto da diversi componenti che possono essere così raggruppati:

- DPS: acquisizione (mediante DISCOS o MCS), elaborazione (utilizzando PacketLib e ProcessorLib) e visualizzazione (Quick-Look) dei dati. Il sistema fornisce anche componenti di supporto, quali un Control Software per la configurazione e l'esecuzione della pipeline ed una GUI per visualizzare i log della pipeline
- DAS: accesso , archiviazione e interrogazione dei dati archiviati.
- Reference Catalogues: cataloghi di riferimento, da utilizzare ad esempio nel Quick-Look
- Data Model: contiene la definizione XML dei data format, data type, data view del CIWS
- Common Libraries:
    - DPS: librerie comuni fornite dal DPS
    - DAS: librerie comuni fornite dal DAS
- External Libraries: dipendenze esterne.

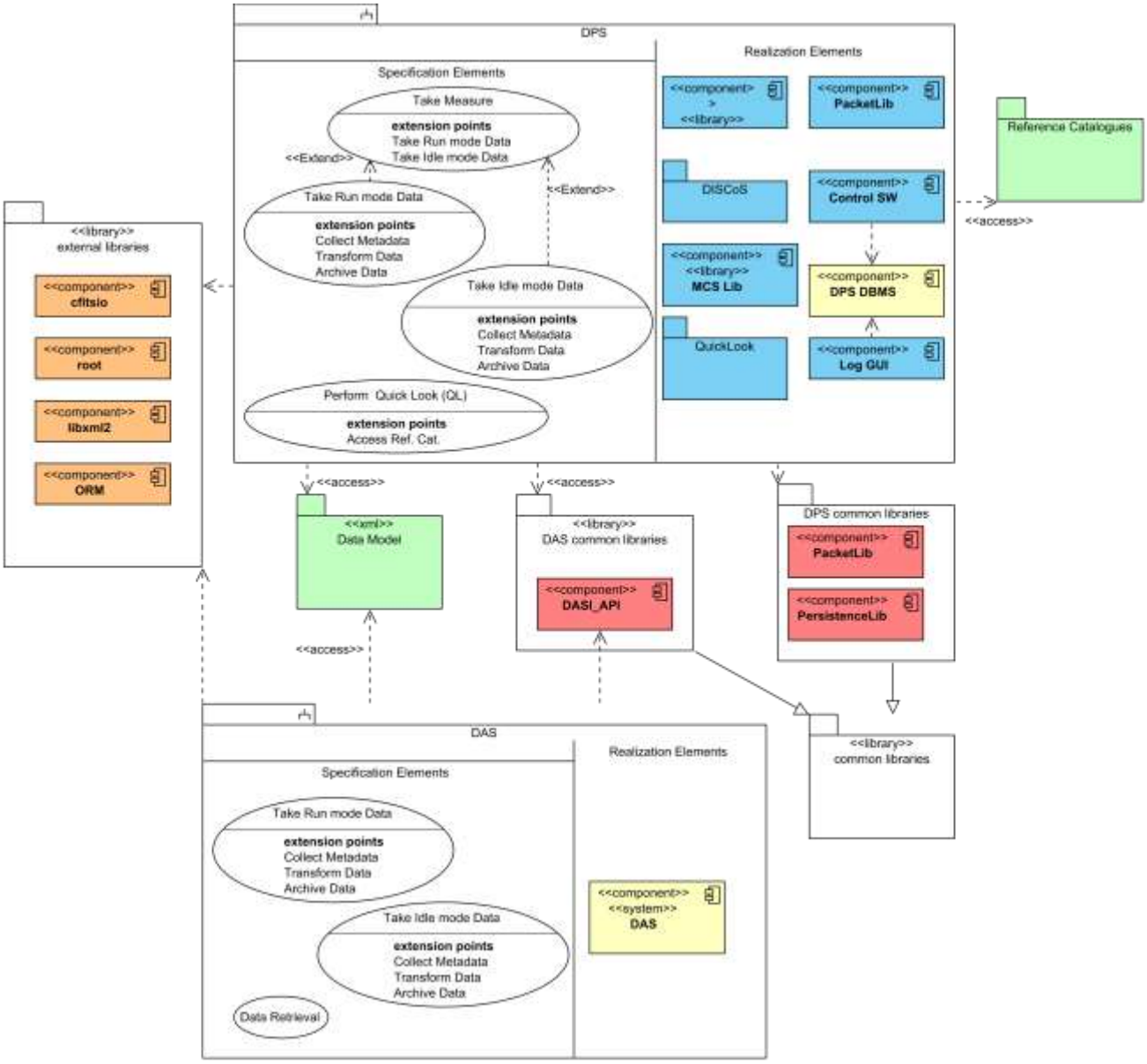


Figura 5-2 Architettura del CIWS-FW

5.1.1 DPS

Il Data Processing System (DPS) è un sistema software di acquisizione, archiviazione su file system, e display grafico dei dati durante l'acquisizione (modalità on-line) e dopo l'acquisizione (modalità off-line). Il DPS è integrato con il DAS (si veda la prossima sezione) per l'archiviazione permanente dei dati acquisiti in un database) e con i Reference Catalogues. Il DPS fornisce strumenti e interfacce di programmazione applicazioni (API) per:

- la definizione del *modello dei dati* (*data model*) di uno specifico progetto, cioè per la specifica delle strutture dati da acquisire e da visualizzare;

- un set di librerie e tools per l'acquisizione e l'archiviazione dei dati su file system, l'elaborazione dei dati raw e la conversione dei dati raw in formati diversi, quali il FITS
- l'interfaccia verso il DAS e i Reference Catalogues.

Il DPS è logicamente suddiviso in tre componenti principali:

A. *Data Acquisition (DAQ)*:
   a. Acquisizione di telemetria e telecomandi provenienti dallo strumento (the DL0 data format);
   b. Acquisizione di files provenienti dallo strumento (the DL0 data format)

B. *Data Transformation (DTR)*: la conversione del formato DL0 nel formato DL1

C. *Quick Look (QL)*: il display dei dati provenienti dallo strumento. In particular, il Quick Look dovrebbe essere in grado di:
   - processare e visualizzare i *dati scientifici* per effettuare verifiche ingegneristiche mediante *Viste* definite dall'utente per organizzare opportunamente i dati;
   - effettuare l'health assessment dello strumento utilizzando i *dati di house-keeping*;
   - visualizzare le mappe del cielo (sky maps) con sovrapposti i Reference Catalogues;

Il Quick Look può funzionare in modalità on-line (connesso al resto del DPS) oppure stand-alone in modalità off-line (accedendo ai dati archiviati).

L'utente del DPS deve per prima cosa configurare il sistema di acquisizione e archiviazione dei dati (DISCOS). Il secondo passo è quello di descrivere i *Source Packet* acquisiti dallo strumento (nel caso di dati organizzati tipo la telemetria satellitare) in un formato XML o testuale per la libreria *PacketLib*, una libreria C++ per il *routing*, il *decoding* e il *coding* della telemetria satellitare. PacketLib è uno dei componenti della *ProcessorLib*, una seconda libreria C++ configurabile mediante file di configurazione e che permette la traduzione del formato dati raw in formato di più alto livello, ad esempio il già citato FITS ([RD.4]). L'ultimo elemento configurabile del DPS è il Quick Look, che dispone di un linguaggio di descrizione dei diversi elementi che lo compongono.

Il linguaggio di definizione degli elementi del Quick Look

Come riportato nella seguente figura, la configurazione necessaria per utilizzare il Quick Look può essere divisa logicamente in parte statica e dinamica.

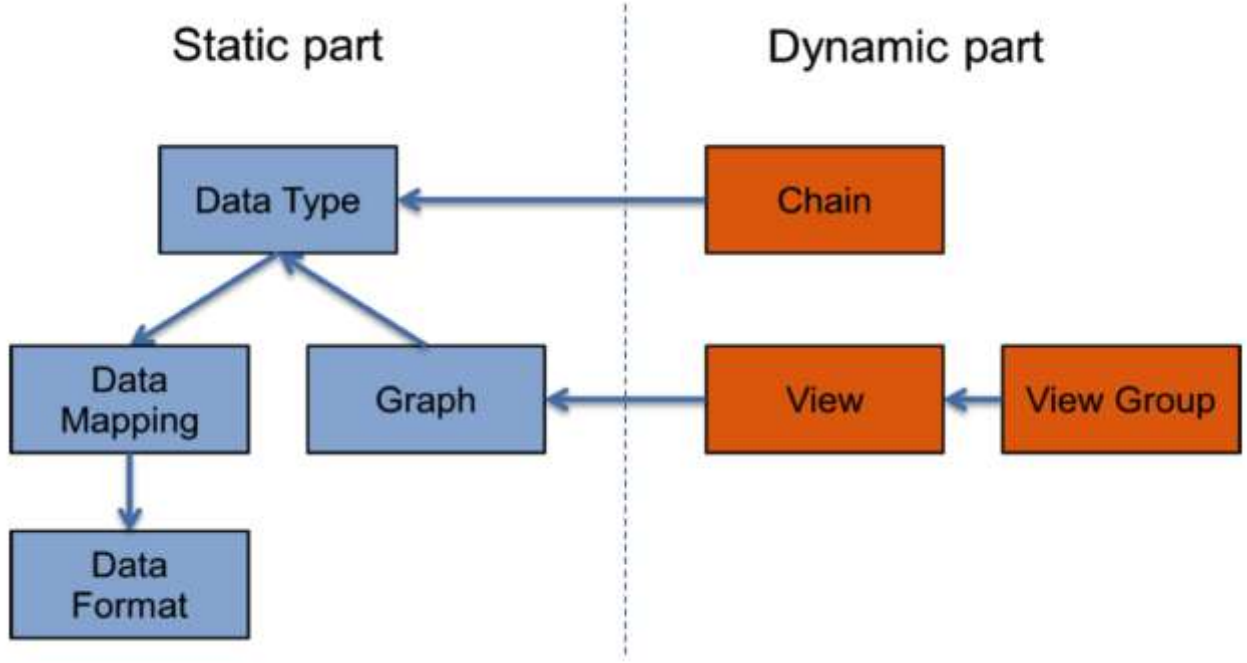


Figura 5-3 I diversi elementi del Quick Look descritti mediante XML

La parte statica è generalmente definita dal CIWS-FW *Developer* poiché il supporto di *data format* aggiuntivi richiede la conoscenza delle operazioni interne del framework e competenze da programmatore. Nel caso semplice in cui vengono usati *data format* già forniti con il framework anche un CIWS *User* può configurare l'intero Quick Look. I seguenti elementi vanno configurati in maniera statica:

- *Data Type*: rappresenta l'astrazione del *Data Format* e può essere visto come una tabella con una numero fisso di colonne.
- *Data Format*: descrive la struttura di un formato file. L'unico formato fornito con il Quick Look è il formato FITS. Altri data format possono essere aggiunti facilmente dai CIWS-FW Developers.
- *Data Mapping*: definisce come effettuare la corrispondenza tra un Data Type e un Data Format.
- *Graphs*: definisce come visualizzare i Data Types, ovvero i grafici da usare.

La parte dinamica, invece, viene aggiornata durante l'esecuzione del Quick Look e comprende:

- *Chain*: controlla quali dati vengono processati dal framework. Questo permette di abilitare/disabilitare un tipo di dato senza dover mettere mano alla definizione del data model.
- *View*: definisce quali grafici verranno visualizzati in una finestra e il layout.
- *ViewGroup*: definisce un gruppo di viste. Permette all'utente di automatizzare l'apertura di più View.

Architettura del DPS

Nella seguente figura è riportato lo schema generale dell'architettura del DPS, con le dipendenze e le relazioni esterne.

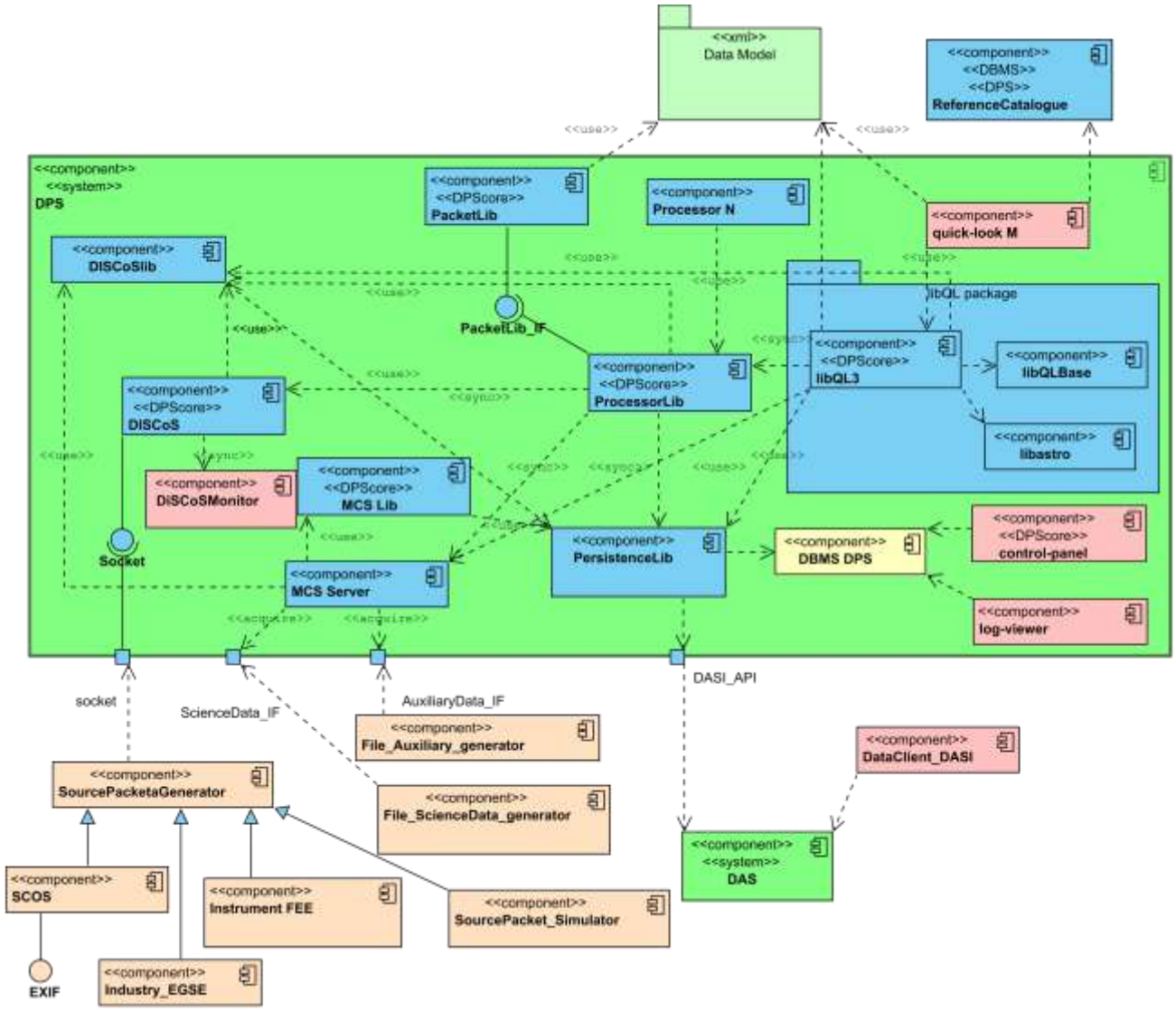


Figura 5-4 DPS: architettura generale

Il DPS, il DAS e i Reference Catalogues possono essere visti come sistemi indipendenti ma che possono essere collegati mediante opportune interfacce.

I componenti del sottosistema di acquisizione dati del DPS sono

- *DISCOS*: per l'acquisizione di un flusso di source packet, l'archiviazione su file system e i meccanismo di sincronismo con i processori (identificati con lo stereotipo <<sync>>) mediante la *DISCOSlib*)
- *MCS Server*, componente di acquisizione di files, sviluppato utilizzando la *MCSLib*

I *sistemi esterni* al CIWS sono evidenziati come componenti arancio nella figura, in particolare

- *Source Packet Generator*: genera i source packet per il sistema DISCOS: le sorgenti dati possono essere diverse, ad esempio l'EGSE fornito dall'industria, SCOS, la FEE dello strumento, un simulatore software di Source Packets;
- *File Generator*: il flusso dati può essere rappresentato da file (di dati scientifici o ausiliari) che sono acquisti dall'MCS server.

Alcune librerie di base consentono l'elaborazione e la conversione dei Source Packets in un diverso formato file:

- *PacketLib*: libreria C++ che permette la codifica, decodifica e routing dei source packet. Questa libreria è utilizzata dalla ProcessorLib
- *ProcessorLib*: libreria utilizzata per la scrittura dei *Processori* (convertitori da source packet a file di output in diverso formato). Fornisce l'interfaccia verso DISCOS e permette allo sviluppatore che utilizza il CIWS-FW di concentrarsi solo sulla scrittura dei *Processori* stessi, lasciando alla ProcessorLib il reperimento da DISCOS e il routing dei source packet, il meccanismo di sincronismo con DISCOS, l'interfaccia con il Quick Look, e l'interfaccia con la *PersistenceLib*. Il formato di output è invece lasciato a scelta dell'utente. Formati utilizzati dal CIWS-FW sono FITS e XML.
- *PersistenceLib*: libreria utilizzata dai diversi componenti del CIWS-FW per la memorizzazione nel database DPS dei path dei file di dati acquisiti e dello stato della pipeline. I dati sono memorizzati nel *DBMS DPS*
- *libQL3*: libreria utilizzata per lo sviluppo dei Quick Look

Altri componenti del sistema sono invece forniti di interfaccia grafica utente:

- *DISCOS Monitor*: un monitor che permette di controllare l'acquisizione dei source packets e il passaggio di questi ai vari processori
- *Control Panel*. Un componente che permette l'esecuzione della pipeline del DPS e la modifica dei parametri della pipeline stessa (mediante interfaccia web)
- *Log Viewer***:** interfaccia web che permette l'interrogazione dei dati memorizzati dalla PersistenceLib nel DBMS DPS
- *Quick Look*: componente per la visualizzazione grafica dei dati acquisiti, già descritti in questa sezione.

5.1.2 DAS

Il Data Access System (DAS) è un sistema software per l'archiviazione, il reperimento e la gestione di metadati e dati acquisiti ed elaborati dall'Instrument Workstation (dati di Livello 1) o prodotti dai successivi livelli di elaborazione. Il DAS fornisce strumenti e interfacce di programmazione applicazioni (API) per:

- la definizione del *modello dei dati* (*data model*) di uno specifico progetto, cioè per la specifica delle strutture dati da archiviare o reperire tramite il sistema, in termini di attributi dei metadati, relazioni con altre strutture dati, ed il formato e layout dei dati binari;
- l'archiviazione dei metadati e delle associazioni tra tipi di dati in un database relazionale, per consentirne l'*interrogazione* ed applicare la *semantica transazionale* alle operazioni di inserimento e reperimento dei dati;

- gestire la *persistenza* dei dati binari, da dati di piccole dimensioni a flussi di dati binari di grandi dimensioni, mantenendo la consistenza tra metadati e dati corrispondenti.

Il DAS è logicamente suddiviso in tre componenti principali:

- un linguaggio di definizione dei dati (*DAS DDL*) in formato XML, il quale fornisce una grammatica sintetica ed intuitiva che gli sviluppatori utilizzano per definire le strutture dati da archiviare o reperire in un progetto specifico;
- un'interfaccia di programmazione applicazioni (*DAS API*), che mappa automaticamente le strutture dati definite dall'utente, tramite il DDL, in classi e metodi di supporto per le operazioni con il database, e fornisce un linguaggio di interrogazione orientato agli oggetti;
- un componente di gestione dei dati (*DAS DMC*), il quale mappa i metadati e le relazioni tra i tipi definiti tramite il DDL in un database relazionale, con l'ausilio di un sistema di mappatura dal modello a oggetti a quello relazionale (*Object to Relational Mapping*, ORM), e memorizza i dati binari in un file system condiviso.

L'utente del DAS deve inizialmente specificare il modello dei dati di uno specifico progetto, tramite la creazione di un file DDL che include tutte le definizioni delle strutture dati. Il DAS elabora tale file e genera automaticamente le classi (nei linguaggi C++ e Python) corrispondenti ai tipi definiti nel DDL, aggiorna la libreria che implementa l'API del DAS e lo schema del database relazionale per supportare i nuovi tipi definiti dall'utente (Figura 5-5). Successivamente, l'utente del DAS passa alla codifica del programma client, nel quale utilizza l'API del DAS per inserire, interrogare o reperire i dati.

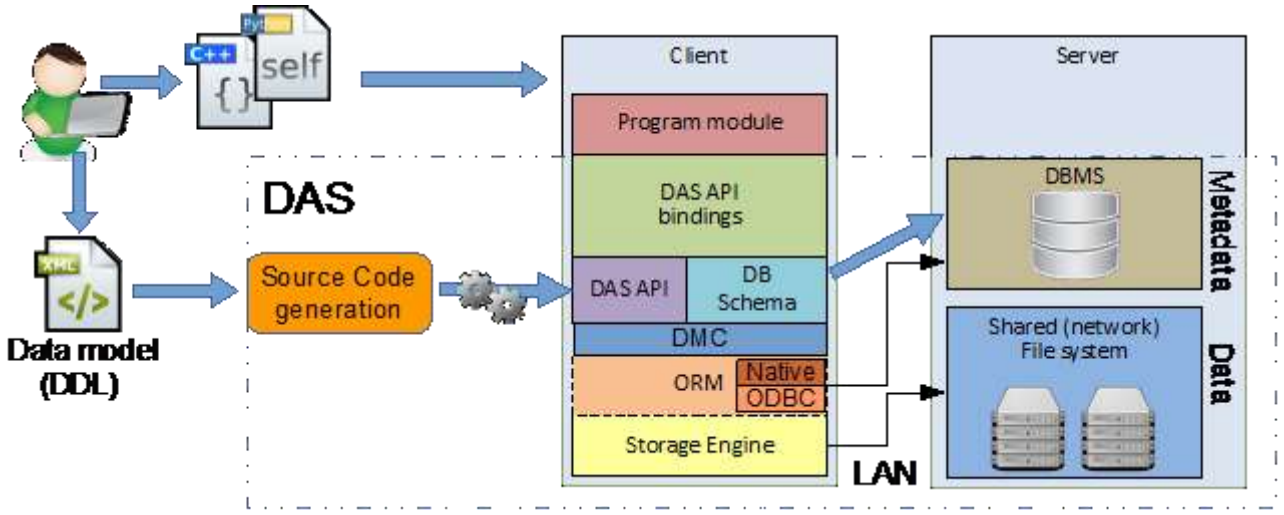


Figura 5-5 DAS: architettura generale

Il linguaggio di definizione dei dati

Il DDL fornisce una grammatica XML, formalizzata nel linguaggio XSD (XML Schema Definition language), per definire le strutture dati, o tipi, da archiviare tramite il DAS.

La definizione di un tipo nel DDL include due sezioni principali: la sezione *metadati* e la sezione *dati* (Figura 5-6). La prima definisce una lista di attributi (*keyword*) che descrivono i dati. La seconda definisce la struttura dei dati binari e può essere di due tipi: una tabella binaria oppure un'immagine. Tali sezioni sono logicamente analoghe all'*header block* e al *data block* dello standard FITS. Poiché la sezione dati è opzionale, è possibile definire tipi di dati che contengono solo metadati. Inoltre, un tipo di dato può essere associato ad altri tipi. L'associazione con un tipo di dato può inoltre indicare: i) una *molteplicità*, quando più istanze dello stesso tipo sono coinvolte nell'associazione; ii) il *tipo di relazione* (*shared, exclusive, extend*), il quale determina in che modo l'associazione viene mappata nello schema relazionale. Per poter definire nuovi tipi di dati come estensioni di tipi di dati esistenti, nella definizione di un tipo è possibile indicare il tipo padre (*ancestor*).

Come opzione predefinita, il DAS memorizza i metadati nel database relazionale di *back-end*; i dati vengono resi persistenti in modo trasparente in un formato binario (analogo alla serializzazione binaria dello standard VOTable), in un filesystem condiviso. Tuttavia, per dati binari di piccole dimensioni (piccoli array o immagini), nella definizione DDL di un tipo è possibile specificare la memorizzazione su BLOB (Binary Large OBject) nel database di *back-end*.

Architettura del DAS

La Figura 5-7 mostra i componenti principali dell'architettura del DAS. Il nucleo del sistema è sviluppato in linguaggio C++. Un componente è costituito dal sistema ORM, che consente la persistenza di istanze di classi C++ in un DBMS (DataBase Management System) relazionale, e gestisce automaticamente la conversione tra i tipi del C++ e quelli del linguaggio SQL. Nel DAS è stato utilizzato un sistema ORM open-source, chiamato ODB (www.codesynthesis.com), il quale attualmente supporta molteplici DBMS, inclusi MySQL, Oracle, PostreSQL ed SQLite. Per ciascun DBMS supportato, ODB si interfaccia ad esso tramite l'API nativa in C (quando disponibile) invece dell'API standard ODBC, consentendo una maggiore efficienza delle operazioni.

A partire dal file DDL creato dall'utente, il DAS genera automaticamente le classi C++ corrispondenti a ciascun tipo del DDL e aggiunge a ogni classe le direttive per il sistema ORM. Tramite le direttive, il sistema ORM genera il codice di supporto per l'interfacciamento al database relazionale. Il DMC implementa l'API del DAS, adottando alcune soluzioni presenti nella specifica JPA (Java Persistency API) che semplificano la sincronizzazione automatica di un dato in memoria e delle istanze ad esso correlate con le controparti rese persistenti nel DAS. Il Data Storage Engine implementa il meccanismo di serializzazione dei dati. Attualmente esso realizza due tipi di serializzazione dei dati binari: come file binari memorizzati su filesystem condiviso oppure come BLOB all'interno del DBMS. Tuttavia esso fornisce delle classi base che possono essere estese per supportare in futuro nuovi tipi di serializzazione (ad es. in formato FITS).

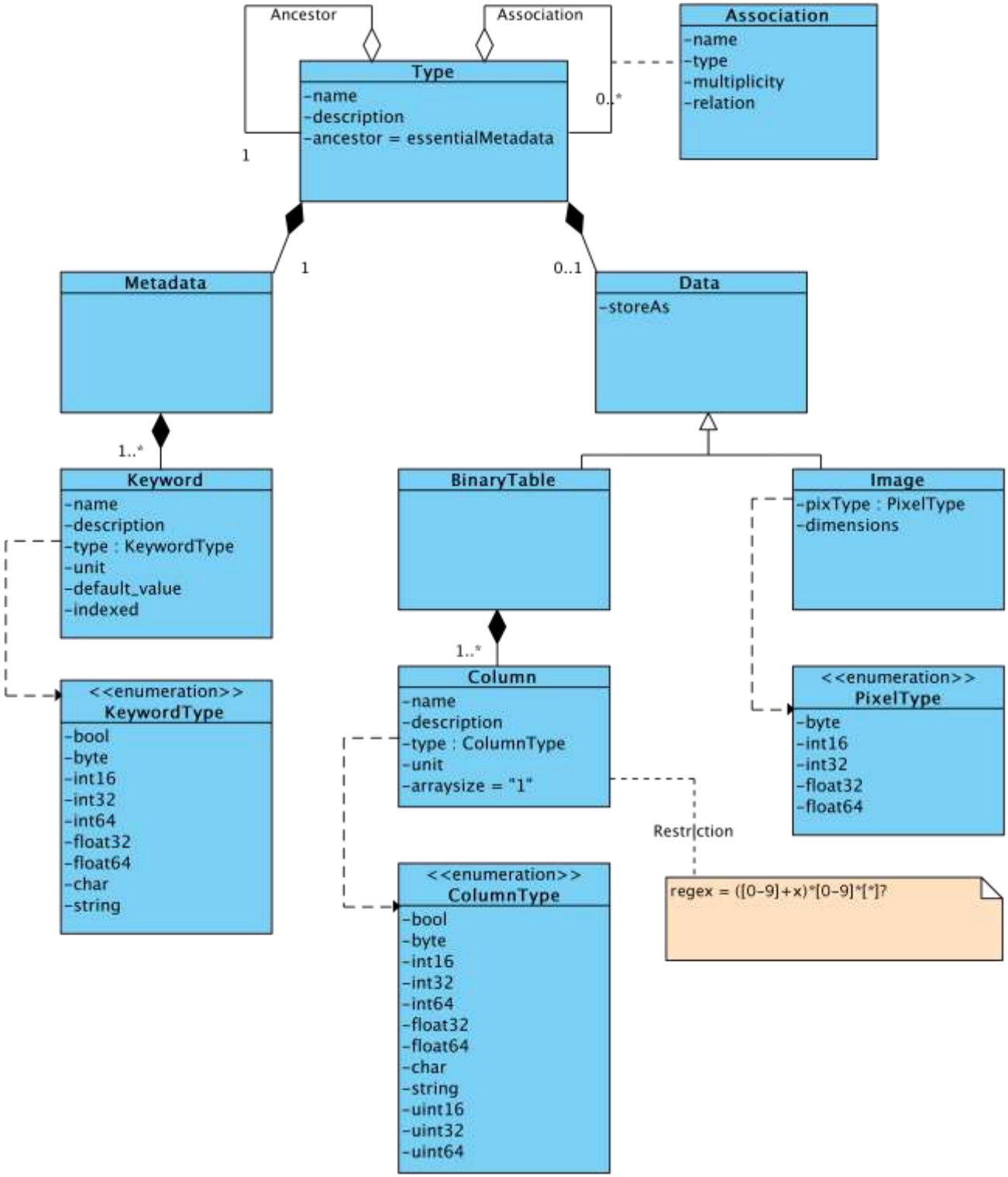


Figura 5-6 Elementi sintattici del linguaggio di definizione dei dati.

L'API principale del DAS, in linguaggio C++, fornisce sia un'interfaccia basata sui *template*, sia un'interfaccia polimorfica che consente l'inferenza dei tipi a tempo di esecuzione. Inoltre, il DAS

fornisce anche un *binding* in linguaggio Python dell'API, tramite l'uso di SWIG (Simplified Wrapper and Interface Generator).

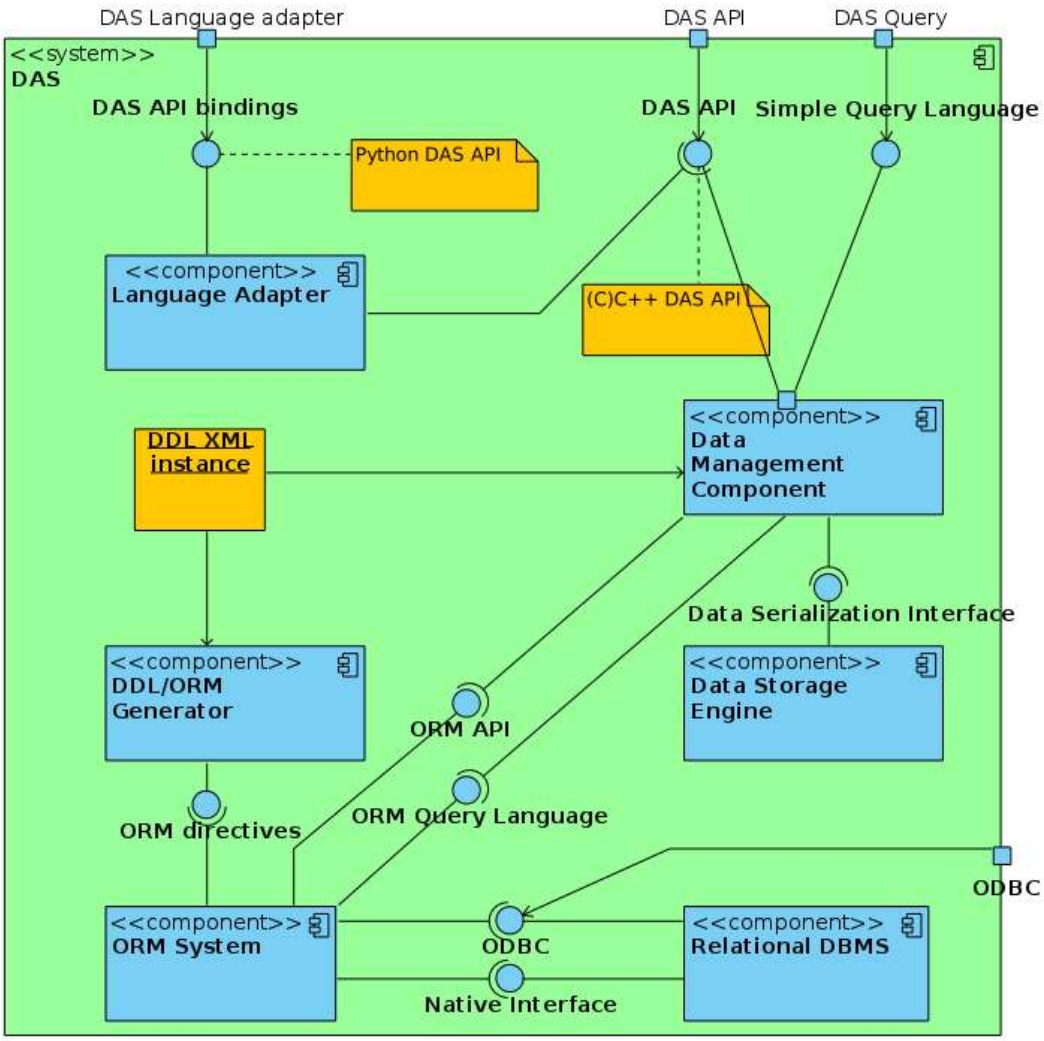


Figura 5-7 Architettura del DAS - component model

5.1.3 Reference Catalogues

L'uso di cataloghi astronomici, nel nostro caso ottico-infrarosso, sono importanti sia per poter simulare e/o pianificare osservazioni, che poter identificare gli oggetti rivelati. L'utilizzo avviene quindi sia in fase di visualizzazione, e.g. Quick Look, che di analisi dati via pipeline o anche interattivamente.

L'approccio usato per la creazione del catalogo dedicato CIWS è stato quello già largamente utilizzato da altri progetti in corso presso l'OATO.

Infatti il catalogo è inserito come tabella in un database relazionale MySQL con i vari oggetti indicizzati usando la pixelizzazione sferica HTM (Kunszt P. Z., Szalay A. S., & Thakar A. R. 2001).

In questo modo la ricerca di oggetti che si trovano in una certa regione di cielo può avvenire in modo molto rapido, tipicamente dell'ordine dei 10 millisec.

Il catalogo principale (Figura 5-8), distribuito con il relativo software di interrogazione, ha un limite in magnitudine di R=14 e contiene poco più di 20 milioni di oggetti.

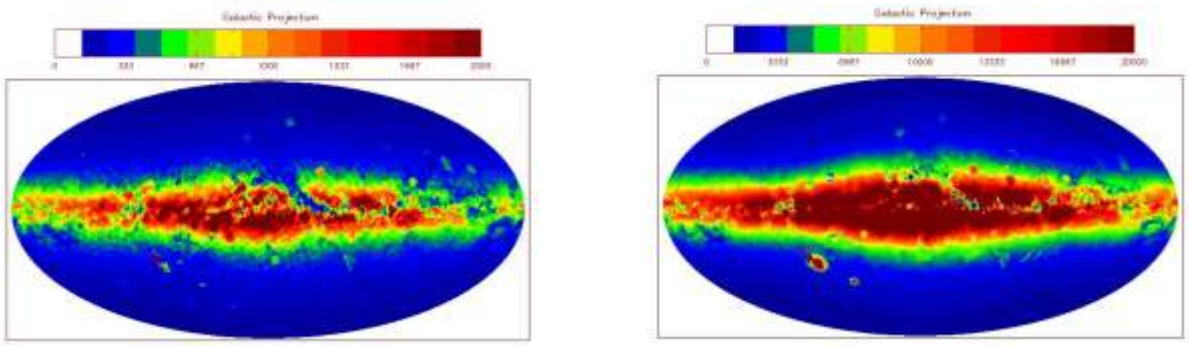


Figura 5-8 A sinistra il Catalogo di default con 20 milioni di oggetti e taglio R=14. A destra il Catalogo con taglio R=18 che contiene circa 330 Milioni di oggetti. Le mappe sono realizzate con pixelizzazione healpix con pixel size di circa ~0.85 gradi quadrati.

L'utente può però anche accedere remotamente a un catalogo con taglio R=18 che contiene circa 327 milioni di oggetti.

Quest'ultimo catalogo è uno dei molti ad accesso pubblico ospitati presso l'OATO il cui utilizzo da parte degli utenti CIWS può avvenire in modo semplice sia attraverso client MySQL, che con l'utilizzo di una libreria API.

Si fa notare che questa libreria, che a sua volta utilizza la libreria DIF ed è disponibile al sito "http://ross.iasfbo.inaf.it/CiwsCats/", è al momento utilizzabile solo per un numero limitato di azioni. Un suo aggiornamento è previsto non appena la struttura dell'archivio cataloghi dell'Osservatorio di Torino sarà finalizzata.

Il catalogo CIWS è stato costruito a partire dai cataloghi GSC2.3, 2MASS e WISE, contiene 11 magnitudini, B,V,R,I,J,H,K,W1,W2,W3,W4 (per i dettagli si fa riferimento alla documentazione CIWS). Si fa notare che per motivi di ottimizzazione, tutti i valori nella tabella MySQL sono convertiti ad interi. Ad esempio le coordinate sono in milliarcsec e le magnitudini sono moltiplicate per 1000. I valori di magnitudine mancanti sono marcati con il valore -32768.

In aggiunta ad un accesso diretto alla tabella con il client di MySQL o altro programma scritto in qualsiasi linguaggio, il catalogo CIWS viene fornito con un programma stand-alone (sviluppato in C++) che offre una serie di opzioni di query e che producono un'uscita ASCII che l'utente può usare a vari scopi.

La compilazione del codice sorgente dipende solo dalla libreria mysqlclient e permette ad un utente avanzato di aggiungere ulteriori opzioni. Il programma stand-alone si chiama "*getCIWScat*". In e qui di seguito vengono riportati alcuni esempi di utilizzo.

```
% getCIWScat
getCIWScat  Ver 0.1b, 05-Mar-2014, LN@IASF-INAF
Usage:
  getCIWScat [OPTIONS] RAcenter DECcenter side_arcmin [CatName]
or:
  getCIWScat [OPTIONS] RAcorner1 DECcorner1 RAcorner2 DECcorner2 [CatName]
with coordinates in degrees, or:
  getCIWScat [OPTIONS]  hh mm ss  +/-dd mm ss  side_arcmin  [CatName]
                     (RAcenter) (DECcenter)
Where OPTIONS are:
  -hcCeEinqrS [-d DB_name] [-m Mag_lim] [-o Order_By] [-O Order_By] [-s Server_name]
  -h: print this help
  -c: compact version: just print 'RA (deg) Dec (deg) R  H'
  -C: input data are center and radius of a circle
  -e: extended version: print more all the info read
  -E: use external server (Turin)
  -i: as -q but does not check existence of CatName
  -n: just print the number of objects resulting from the query
  -q: print the query without executing it
  -r: print the output in raw format as read from the DB
  -S: print RA and Dec in string rather than fractional degrees
  -d DB_name: send query to 'DB_name'
  -m Mag_lim: select only objects with (one or more) Mag < 'Mag_lim'
  -o Order_By: order output using 'Order_By' column (use -r to see names)
  -O Order_By: like -o but sort in Descending order
  -s Server_name: send query to 'Server_name'
and 'CatName' can be:
  CIWScat14 (def.) | CIWScat18 | (more to come ?) |
':' and fractional 'ss.s' in coordinates are allowed.
Default server and DB: localhost, PUBCats

% getCIWScat 10 20 30

#RA Dec B V R I J H K W1 W2 W3 W4
10.237224 +19.892600 14.61 13.99 13.66 13.34 12.90 12.59 12.52 -32.77 -32.77 -32.77 -32.77
10.145366 +19.989884 14.89 14.23 13.94 13.70 13.08 12.80 12.71 -32.77 -32.77 -32.77 -32.77
...

% getCIWScat 10 12 13 -20 30 40 10 ( or getCIWScat 10:12:13 -20:30:40 10 )

#RA Dec B V R I J H K W1 W2 W3 W4
153.106171 -20.501918 14.34 -32.77 12.81 12.21 11.34 10.74 10.63 -32.77 -32.77 -32.77 -32.77
153.088559 -20.464719 14.23 -32.77 13.27 13.02 12.57 12.25 12.23 -32.77 -32.77 -32.77 -32.77
...

% getCIWScat -c 10 -20 30

#RA Dec R H
10.039079 -20.048558 12.49 11.62
10.068512 -20.029344 13.93 12.54
...

% getCIWScat -c -S 10 -20 30

#RA Dec R H
10:02:20.68 -20:02:54.8 12.49 11.62
10:04:06.64 -20:01:45.6 13.93 12.54
...
```

Figura 5-9 Esempi di utilizzo del programma getCIWScat

6. Integrazione e test

Integrazione e testing si posizionano nel livello più basso del ciclo di vita di verifica della qualità del software, come da figura seguente.

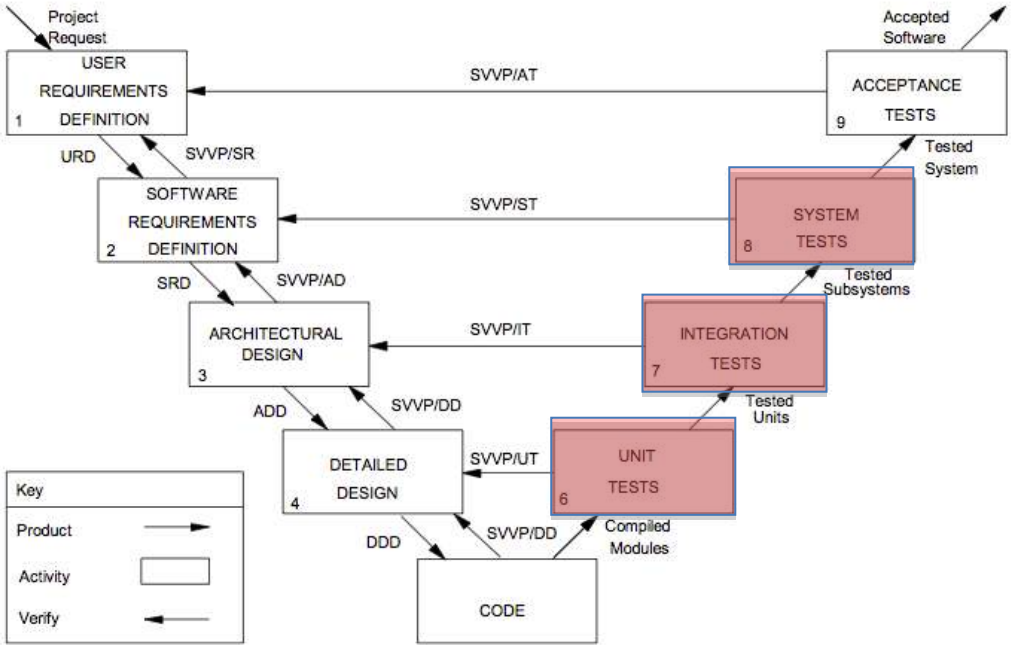


Figura 6-1 Software life cycle: verification approach

Con *Unit Testing* o component testing in ambito object-oriented ci si riferisce alla verifica delle funzionalità a livello di classi. Per il CIWS-FW lo unit-testing viene attualmente eseguito automaticamente appena effettuata la build di un componente.

I componenti testati a questo livello sono il DAS, PacketLib e libQLBase, con l'intento di automatizzare questo test anche per il resto dei componenti.

Come unit-test library è stata scelta quella di boost ed è stata automatizzata grazie al sistema di build *cmake*.

Lo scopo dell' *Integration Testing* invece è verificare che le interfacce tra componenti rispettino le specifiche di progettazione. Si è deciso di effettuare integrazioni iterative e su componenti separati, invece di effettuare una integrazione globale di tutti i componenti.

Sono stati svolti, in alcuni casi anche in parallelo, i seguenti Integration Test tra i componenti:

- Test connessione tra S21Processor e DISCoS tramite l'uso del Control Panel (si veda Sezioni 5 e 7.1)
  Componenti: DISCoS, PacketLib, ProcessorLib, Control Panel, S21Processor
- Test di connessione tra S21Processor, DISCoS e DAS (si veda Sezioni 5 e 7.1)
  Componenti: DISCoS, PacketLib, ProcessorLib, DAS, S21Processor
- Test di funzionamento del Quick Look offline (si veda Sezioni 5 e 7.1)
  Componenti: libQL package, quick-look ASTRI

- Test di funzionamento del Quick Look online con dati ricevuti da DISCoS (si veda Sezioni 5 e 7.1)
  Componenti: DISCoS, PacketLib, ProcessorLib, libQL3 package, quick-look ASTRI
- Test di funzionamento del Quick Look con le immagini e reference catalogue (si veda Sezioni 5 e 7.2)
  Componenti: Reference catalogues, libQL package, quick-look AGILE
- Comando remoto con ActiveMQ
  Componenti: ActiveMQ, Libstomp, DISCoS, Consumer, Producer
- Acquisizione file-by-file con MCS
  Componenti: MCS server, MCS lib

Sono stati svolti con successo anche due *System Test*:

- System Test del Prototipo ASTRI (Sezione 7.1)
- System Test del Prototipo AGILE (Sezione 7.2)

È stato svolto anche un Installation Testing del CIWS-FW sulla macchina IASFBO ciws02 con piattaforma target del CIWS-FW ovvero CentOS 6.4. La maggior parte delle dipendenze vengono installate da sorgenti e automatizzate tramite script di *bash*.

Il software del progetto è disponibile in vari repository git. Di seguito si riporta la struttura dei repository:

```
R W  CIWS-FW/DAS/DAS
R W  CIWS-FW/DPS/DISCoS/DISCoSConf
R W  CIWS-FW/DPS/DISCoS/DISCoSDoc
R W  CIWS-FW/DPS/DISCoS/DISCoSExt
R W  CIWS-FW/DPS/DISCoS/DISCoSMake
R W  CIWS-FW/DPS/DISCoS/DISCoSMon
R W  CIWS-FW/DPS/DISCoS/DISCoSPro
R W  CIWS-FW/DPS/DISCoS/DISCoSRcvArc
R W  CIWS-FW/DPS/DISCoS/DISCoSRcvScript
R W  CIWS-FW/DPS/DISCoS/DISCoSScript
R W  CIWS-FW/DPS/DISCoS/DISCoSSnd
R W  CIWS-FW/DPS/DISCoS/DISCoSlib
R W  CIWS-FW/DPS/FileServer
R W  CIWS-FW/DPS/PacketLib
R W  CIWS-FW/DPS/PersistenceLibC
R W  CIWS-FW/DPS/PersistenceLibCpp
R W  CIWS-FW/DPS/ProcessorLib
R W  CIWS-FW/DPS/QL/libQL3
R W  CIWS-FW/DPS/QL/libQLBase
R W  CIWS-FW/DPS/QL/libastro
R W  CIWS-FW/DPS/common_include
R W  CIWS-FW/Setup/CIWS-FW-Setup
R W  CIWS-FW/Setup/CIWS-FW-deps-Setup
R W  CIWS/IW1/QL_ASTRI
R W  CIWS/IW1/Setup/IW1-Setup
R W  CIWS/IW1/processorS21
R W  CIWS/IW2/QL_AGILE
R W  CIWS/IW2/Setup/QL_AGILE-Setup
R W  CIWS/IW2/libAGILE
```

Figura 6-2 Struttura dei repositories Git del CIWS-FW

Lo scaricamento dei repository viene fatto con *git-submodules*. L'installazione avviene utilizzando uno script bash. Di seguito se ne riporta un esempio, che compila i vari repository e li installa in una directory (e.g. local).

```
#!/usr/bin/env bash
CIWSFW_BUILDDIR=$PWD/build
### build configuration
BUILD_TYPE=Debug
NTHREADS=8
VERBOSE="VERBOSE=1"
clean=$1 # pass "clean" to this script in order to make clean before make.
# workaround for CentOS naming
if [ -e /usr/bin/cmake28 ]; then
  cmake_exe=cmake28
else
  cmake_exe=cmake
fi
### internal functions ###########################
function echo_red {
  echo -e "\e[00;31m$1\e[00m"
}
function cmake_install {
  trap exit ERR
  MODULE_REPOSITORY=$PWD/$1
  MODULE_BUILDDIR=${CIWSFW_BUILDDIR}/$1/$BUILD_TYPE
  pushd $PWD
  if [[ $clean == "clean" ]];
  then
    echo
    echo "Cleaning $1..."
    rm -rf ${CIWSFW_BUILDDIR}/$1
    echo "Clean complete."
  else
    mkdir -p ${MODULE_BUILDDIR}
    cd ${MODULE_BUILDDIR}
    echo
    echo_red "Building $1..."
    $cmake_exe -DCMAKE_BUILD_TYPE=$BUILD_TYPE \
          -DCMAKE_INSTALL_PREFIX:PATH=${CIWSFW_PREFIX} \
          -DBoost_NO_BOOST_CMAKE=ON \
           ${MODULE_REPOSITORY}
    make ${VERBOSE} -j${NTHREADS} ${*:2}
    echo_red "Build complete."
    echo_red "Installing $1..."
    $SUDO make install
    echo_red "Installation complete."
  fi
  popd
}
function make_install {
  trap exit ERR
  pushd $PWD
  cd $1
  if [[ $clean == "clean" ]]
  then
    echo
    echo "Cleaning $1..."
    make clean
    echo "Clean complete."
  else
    echo
    echo_red "Building $1..."
    make ${VERBOSE} -j${NTHREADS} ${*:2}
    echo_red "Build complete."
    echo_red "Installing $1..."
    $SUDO make install prefix=${CIWSFW_PREFIX}
    echo_red "Installation complete."
  fi
```

```
  popd
}
################################################
### check environmental variables
if [ -z "$CIWSFW_DEPS" ] || [ -z $(env | grep CIWSFW_DEPS) ]; then
  echo "You need to set the CIWSFW_DEPS environmental variable based on your
previous deps installation path."
  exit
fi
startpath=`pwd`
### load profiles
. profile
. ${CIWSFW_DEPS}/profile
### test if we need root privileges to install files
INFO=( $(stat -c"%a %G %U" -L $CIWSFW_PREFIX) )
PERM=${INFO[0]}
GROUP=${INFO[1]}
OWNER=${INFO[2]}
if [[ $OWNER == "root" ]]; then
  SUDO=sudo
fi
### build/install CIWSFW modules.
cd PacketLib
patch=../patches/0001-Remove-m64-CPPFLAG.patch
git apply --check $patch 2>/dev/null || git apply --reverse $patch
git apply $patch
cd ..
make_install PacketLib CPPFLAGS="-m32"
make_install common_include
make_install DISCoS/DISCoSlib CFLAGS="-m32"
make_install DISCoS/DISCoSConf CFLAGS="-m32"
make_install DISCoS/DISCoSExt CFLAGS="-m32"
make_install DISCoS/DISCoSMon CFLAGS="-m32"
make_install DISCoS/DISCoSRcvArc CFLAGS="-m32"
make_install DISCoS/DISCoSDoc CFLAGS="-m32"
make_install DISCoS/DISCoSPro CFLAGS="-m32"
make_install DISCoS/DISCoSScript CFLAGS="-m32"
make_install DISCoS/DISCoSSnd CFLAGS="-m32"
make_install ProcessorLib CPPFLAGS="-m32"
make_install PersistenceLibC
cmake_install libQLBase
cmake_install libastro
cmake_install libQL3
# install DAS component
trap exit ERR
MODULE_REPOSITORY=$PWD/DAS
MODULE_BUILDDIR=$MODULE_REPOSITORY
pushd $PWD
if [[ $clean == "clean" ]]; then
  echo
  echo "Cleaning DAS..."
  cd ${MODULE_BUILDDIR}
  git clean -df
  echo "Clean complete."
else
  mkdir -p ${MODULE_BUILDDIR}
  cd ${MODULE_BUILDDIR}
  echo
  echo_red "Building DAS..."
  cmake -DCMAKE_BUILD_TYPE=$BUILD_TYPE \
        -DCMAKE_INSTALL_PREFIX:PATH=${CIWSFW_PREFIX} \
         ${MODULE_REPOSITORY}
  make DAS_A ${VERBOSE} -j${NTHREADS}
  make DAS_SO ${VERBOSE} -j${NTHREADS}
  make ${VERBOSE} -j${NTHREADS}
  echo_red "Build complete."
  echo_red "Installing DAS..."
  $SUDO make install
  echo_red "Installation complete."
```

```
fi
popd
# install the profile
cd $startpath
if [[ $CIWSFW_PREFIX != "/" ]]; then
  cp profile $CIWSFW_PREFIX
fi
if [[ $clean == "clean" ]];
then
  echo_red "\nCIWS-FW cleaning successful."
else
  echo_red "\nCIWS-FW installation successful."
fi
```

Figura 6-3 Script per l'installazione (deployment) del software CIWS-FW

7. Proof of concept

7.1 Prototipo ASTRI

Utilizzando il CIWS-FW è stato implementato un prototipo della Instrument Workstation che soddisfa i requisiti software del progetto ASTRI. Questo prototipo rappresenta la "*Proof of Concept*" dei moduli CIWS-FW che realizzano l'acquisizione e archiviazione in formato RAW e FITS, l'acquisizione e archiviazione in formato RAW e DAS, i comandi remoti con Active MQ e l'acquisizione di files.

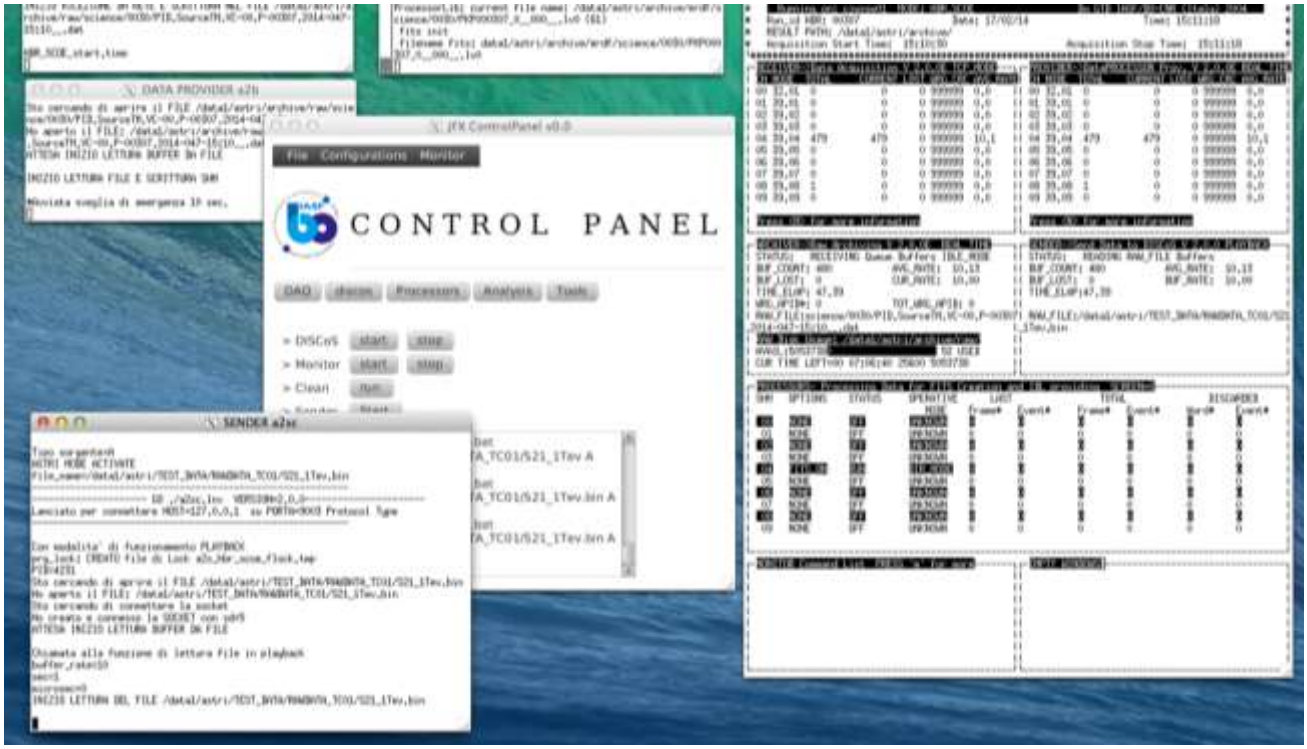


Figura 7-1GUI dell'operatore per il controllo dell'acquisizione e archiviazione del prototipo ASTRI

7.1.1 Acquisizione e archiviazione RAW e FITS

Questo modulo è in grado di acquisire un flusso di dati pacchetto per pacchetto grazie alle funzionalità offerte dalla *DISCoSLib*. È stato implementato il componente software che consente di acquisire e archiviare i dati ASTRI di tipo S21 [RD.7]. Il componente *ProcessorLib* è stato usato per sviluppare il processore che converte i dati di input come pacchetti di dati raw in real time (oppure file raw in off-line) in file FITS. Il test del prototipo è stato eseguito sfruttando le funzionalità della GUI di controllo del framework: Il *Control Panel* (mostrato nell'immagine sotto).

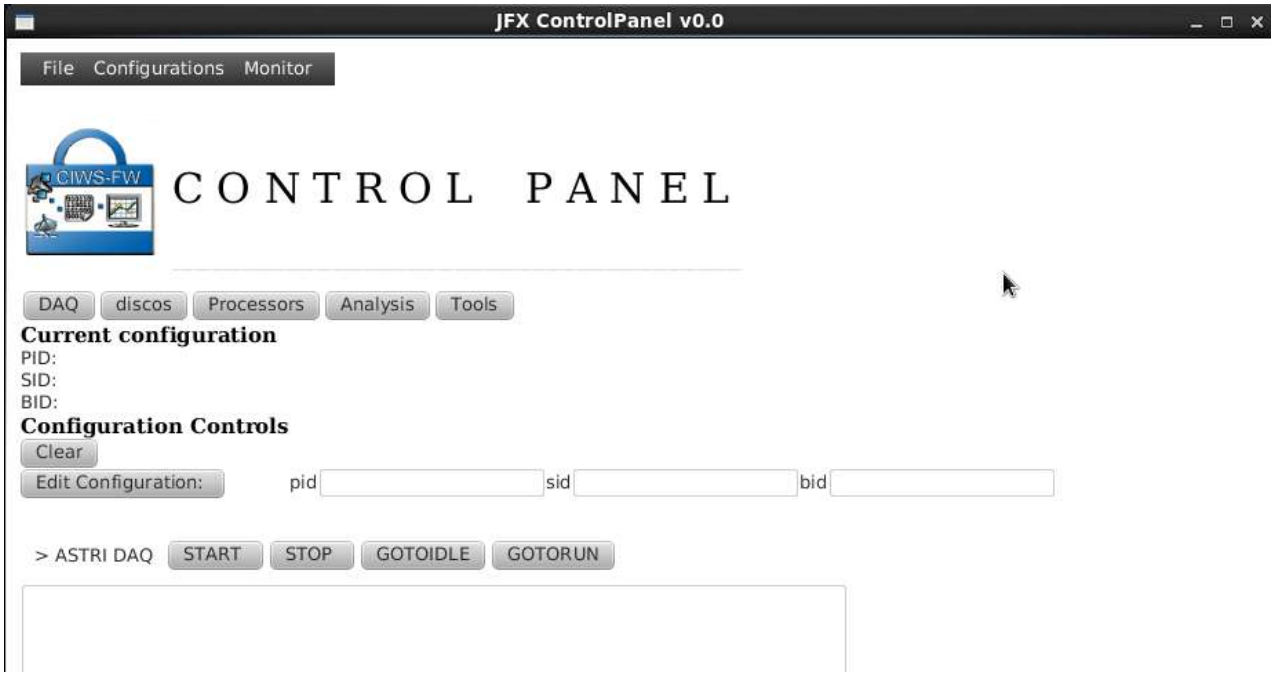


Figura 7-2Pannello di Configurazione e Controllo del Prototipo ASTRI

Dal Control Panel è stato eseguito come primo componente il Monitor (nell'immagine sotto) che visualizza lo stato del sistema di acquisizione.

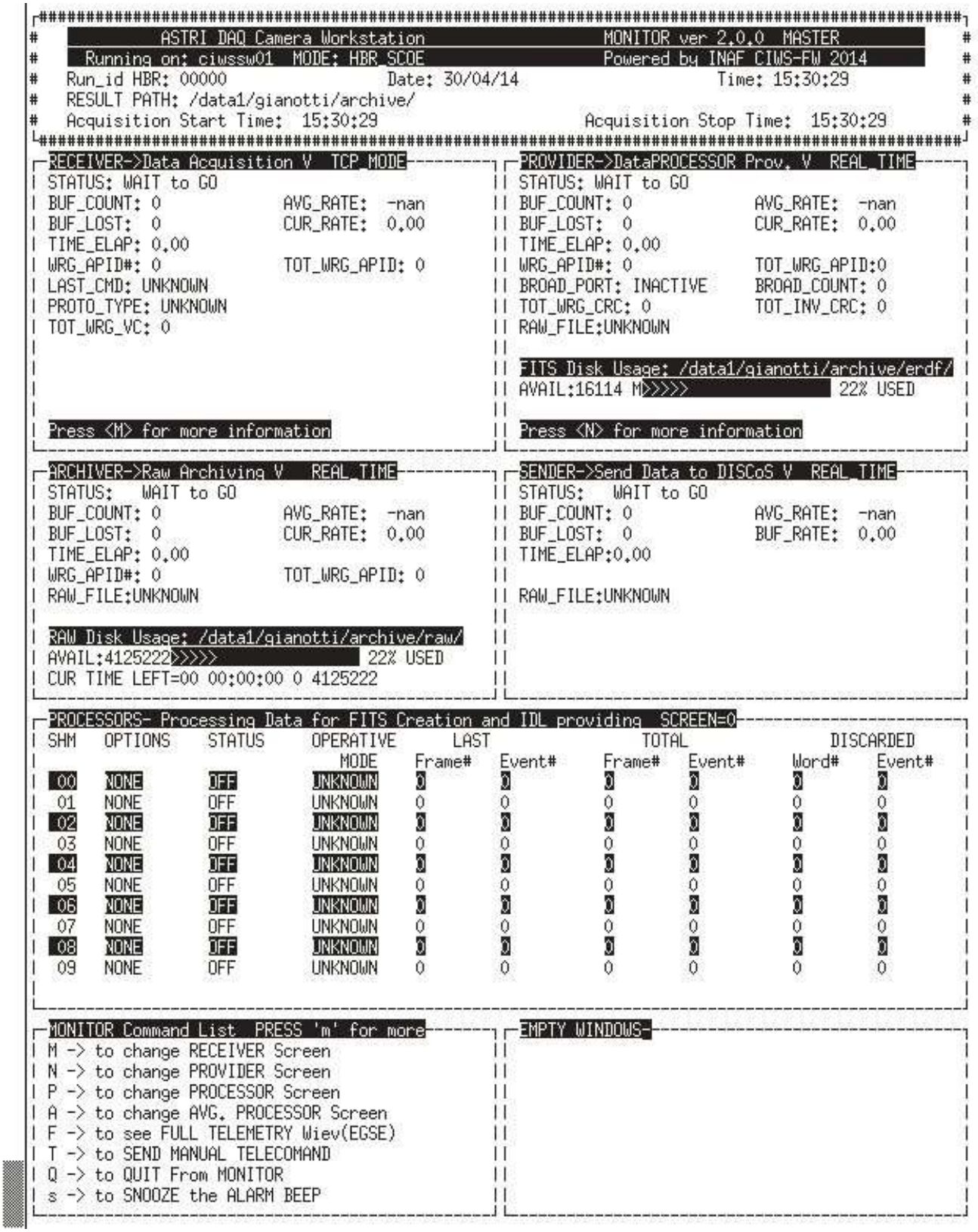


Figura 7-3 Pannello di monitor del prototipo ASTRI

Subito dopo sono stati eseguiti il *DISCoS* e il *processore S21* (converte i dati scientifici L0 raw in formato L0 FITS) come mostrano le finestre dell'immagine sottostante. L'esecuzione di *DISCoS* è tracciata su due finestre. Quella in alto visualizza il processo di archiviazione dei file RAW, mentre quella al centro mostra il processo di acquisizione dei dati dallo strumento.

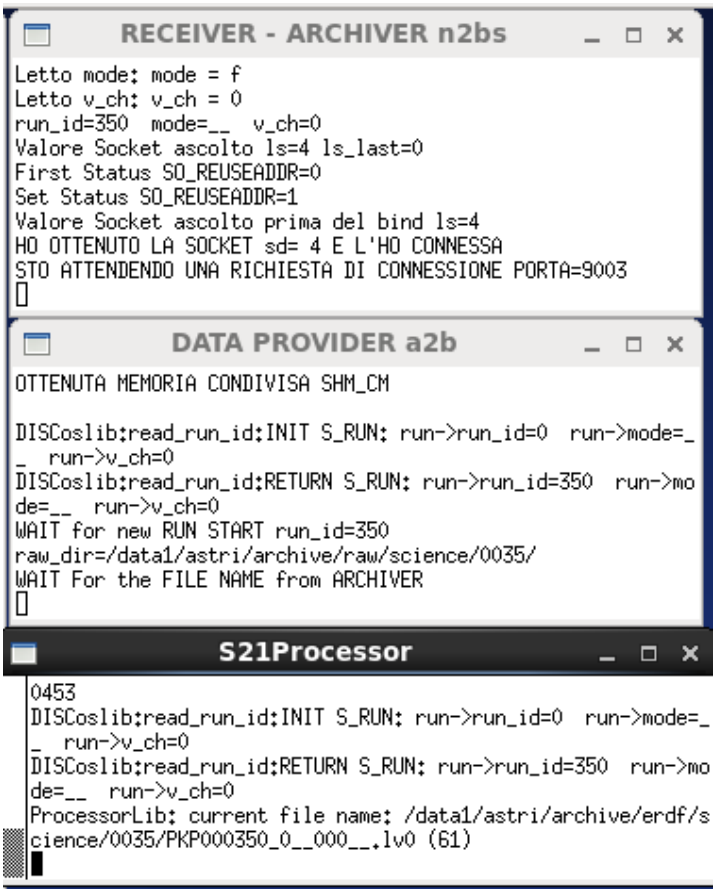


Figura 7-4 Pannelli di monitor dei componenti del Prototipo ASTRI

Il sistema di acquisizione è stato testato grazie all'ausilio di un simulatore dello strumento, il *Sender*, il quale può essere configurato in termini di velocità di invio pacchetti; ciò consente di eseguire sia test di correttezza che di performance. L'immagine sotto mostra l'interfaccia del Sender.

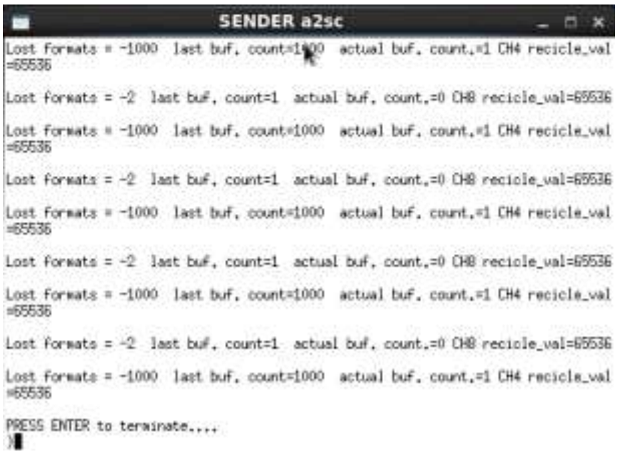


Figura 7-5 Pannello di monitor del Simulatore di Telemetria del Prototipo ASTRI

7.1.2 Acquisizione, archiviazione raw e DAS

In questo modulo di prototipo è rimasta invariata la parte di acquisizione e archiviazione raw, mentre è stata sostituita la parte di scrittura su file FITS con la scrittura su DAS mediante opportune API.

È stato creato il DDL per definire il tipo di dato S21 da archiviare. La struttura è simile a quella del file FITS. Il riquadro sottostante mostra una parte del DDL:

```
<type name="s21" description="ciws prototype data">
<metadata>
<keyword name="telescope" type="string" index="yes"/>
<keyword name="instrument" type="string" index="yes"/>
<keyword name="telescopeid" description="telescope identifier" type="int16" index="yes" />
...
</metadata>
<data storeAs="file">
<binaryTable>
<column name="datetime" type="string" description="UTC date time yy-mm-ddThh:hh:ss" />
<column name="seconds" type="int32" unit="s" description="nanoseconds fraction time" />
<column name="nanoseconds" type="int32" unit="ns" description="nanoseconds fraction time"
/>
<column name="gpsStatus" type="boolean" description="gps status" />
<column name="eventnumber" type="int32" description=""/>
<column name="flags" type="int32" />
<column name="packetlen" type="int16" description="packet lenght" />
<column name="tr_pdm" type="byte" description="triggered pdm" />
<column name="ssc" type="int16" description="source sequence counter" />
<column name="event_counter" type="int32" />
<column name="PDM1_HighGain" type="int16" unit="v" description="high gain values"
arraysize="64" />
<column name="PDM1_LowGain" type="int16" unit="v" description="low gain values"
arraysize="64" />
...
<column name="PDM37_HighGain" type="int16" unit="v" description="high gain values"
arraysize="64" />
<column name="PDM37_LowGain" type="int16" unit="v" description="low gain values"
arraysize="64" />
</binaryTable>
</data>
</type>
```

Al fine di ottenere delle performance ottimali sui tempi di esecuzione sono stati scritti più eventi nello stesso oggetto: è stato creato un oggetto *transient* ed è stato eseguito un *append* per ogni

pacchetto ricevuto. Raggiunto un numero di eventi prestabilito per l'oggetto, esso è stato resto persistente nel sistema DAS.

Il codice sorgente sotto mostra la fase di append:

```
bool CIWSProcessor::setValue(){
 ...
 short ssc = p->header->getFieldValue(3);
 das::Array<short> a_ssc(1);
 a_ssc(0) = ssc;
 das->append_column("ssc",a_ssc);
 …
 for (int i = 0; i < nblocks; i++){
 stringstream hi_cname;
 hi_cname << "PDM" << i+1 << "_HighGain";
 das::ColumnArray<short> high_gains(1);
 high_gains(0).resize(64);
 SDFBFBlock* current_block = sdf->getBlock(i);
 // the field 2 is the spare.
 int offset = 3; // the primer are of pdm id

 for (int j = 0; j< 64; j++)
 high_gains(0)(j) = current_block->getFieldValue( j + offset);
 das->append_column_array(hi_cname.str(), high_gains);
 …
 }
 return true;
}
```

Nella figura sotto è riportato il codice che rende persistente l'oggetto:

```
int* CIWSProcessor::initIntValueForOutput_close(){
 if(das){ // persist the das
 // create connection
 shared_ptr<das::tpl::Database> db = das::tpl::Database::create("ciws_prototype");
 // create transaction
 das::Transaction tr = db->begin();
 // write packets
 db->persist(das);
 tr.commit();
 }
 return 0;
}
```

7.1.3 Comando remoto con ActiveMQ

Il CIWS-FW permette anche di interfacciare sistemi esterni per ricevere comandi remoti. Allo scopo si è deciso di realizzare un prototipo usando Apache ActiveMQ, un sistema altamente scalabile e flessibile che permette di gestire code di messaggi. Con tale prototipo si è voluto dimostrare come un sistema generico può scambiare messaggi con DISCoS (siano essi comandi, telemetria o altro).

Nel caso specifico ci si è concentrati sulla parte di comandi ed è stato adattato dagli esempi un prototipo produttore/consumatore affinché un sistema generico possa generare un messaggio destinato a DISCoS; ActiveMQ gioca un ruolo di “postino” e ha il compito di instradare il messaggio da un mittente a un destinatario. Il processo consumatore (che fa parte di DISCoS), una volta ricevuto il messaggio (che è stato previamente depositato su una coda di messaggi) avrà il compito di trasformarlo e di trasmetterlo alla “local Shared Queue” di DISCoS stesso per la successiva esecuzione.

Essendo ActiveMQ un sistema pensato per gestire molte connessioni, si evita di dover generare un numero che può essere elevato di processi DISCoS che devono gestire connessioni esterne, rendendo il framework CIWS altamente flessibile.

Per il momento non ci si è preoccupati della sicurezza, quindi non si è preso in considerazione l'uso di autentificazione o l'uso di protocolli sicuri come *ssl/https*, ecc., ma viene usata una semplice connessione in cui i dati transitano in chiaro; in particolare così è come lavora per default la libreria *stomp* che è stata usata.

Stomp è una libreria che gestisce i messaggi con un protocollo di messaggistica orientato al testo (*Simple Text Oriented Messaging Protocol*). Anche se questa libreria è stata usata in una fase preliminare, è abbastanza sconsigliato l'uso (con CIWS/DISCoS) per la completa mancanza di “data typing”, dovendo l'utente gestire correttamente il protocollo interno di spedizione/ricezione.

7.1.4 Acquisizione file-by-file con MCS

Il CIWS-FW consente di eseguire l'acquisizione dati dallo strumento file by file. Abbiamo sviluppato un prototipo in grado di acquisire file in formato FITS, archiviarli e notificare la ricezione al componente *Persistence* del CIWS-FW. A questo scopo è stato implementato un *server multi-thread* minimale usando la libreria MCS. Questo offre il servizio di acquisizione file e di *logging* su database MySQL. Per eseguire i test è stato usato un client MCS scritto in C++ (ma si sarebbe potuto usare un qualsiasi altro linguaggio) in grado di collegarsi al server e inviare file FITS del tipo S21 di ASTRI. Il Server MCS ogni volta che riceve un file FITS legge l'*header* in cui sono riportate le caratteristiche del dato e, dopo le opportune verifiche, archivia il file nel path definito dall'utente in fase di configurazione. Contestualmente all'archiviazione, il Server MCS notifica la ricezione al Persistence mediante un'interfaccia in cui viene passata la configurazione (piano, sessione e blocco di misura) e il *path* completo in cui è stato archiviato il file. Tutte le configurazioni, sia per il Server che i Client, sono gestite da un file che viene letto a run time. Al lancio, il Server richiede i tre parametri *"pid, sid, bid"* e legge quindi la configurazione e lo stato dei file già acquisiti e presenti nel log del database. Il processo rimane attivo in background come *daemon*. L'output può essere indirizzato sia su *stdout* che su file. Di seguito è riportato l'output ottenuto durante l'esecuzione del prototipo in cui è stato ricevuto un file FITS. La colonna "*thr*" riporta l'ID del *thread* che ha effettuato l'azione: *MMM* indica il Main thread, *LLL* il Local thread e *nnn* è l'ID univoco del Client. Per terminarlo, si usa il comando "*killmcs*". Si veda il manuale MCS per ulteriori informazioni.

```
% myFileServer 20 30 10
 *** myFileServer Ver 0.1b, 07-03-2014, LN@IASF-INAF ***
Process started at: Tue Apr  8 11:48:37 2014

Will use table 'CiwsDB.DevicesLog_20_30_10'

My Customizable Server (MCS) ver. 0.3.3-alpha3

YYYYMMDD HHMMSS thr MESSAGE
-------- ------ --- ----------------------------------------------------
---------
20140408 114837 MMM Server                                  created
20140408 114837 MMM APPD set to: /home/nicastro/store
20140408 114837 MMM Creating new LocalThread...
20140408 114837 LLL LocalThread                             created
20140408 114837 LLL  *** myFileServer Ver 0.1b, 07-03-2014, LN@IASF-
INAF ***
20140408 114837 LLL initial: locPath=/home/nicastro/store/ciwsusr/
20140408 114837 LLL Start Local::run().
20140408 114837 LLL Configuration file: /home/nicastro/ciwsfiles.conf
20140408 114837 LLL : no unsent images found.
```

```
20140408 114837 MMM MCS Server running on 192.167.166.10:9313


-- Qui di seguito il logging del Server in ricezione file:

20140408 115013 MMM Creating new UserThread on ID=0...
20140408 115013 000 UserThread                          created
20140408 115013 000  >Welcome at luna.iasfbo.inaf.it running
myFileServer ver. 0.1a
20140408 115013 000 < CID
20140408 115013 000  >Client ID: |0|, chunk size: |1638400|
20140408 115013 000 < USR ciwsusr
20140408 115013 000  >Welcome user ciwsusr.
20140408 115013 000 < *** PASSWORD ***
20140408 115013 000  >OK.
20140408 115013 000 < DBN CiwsDB
20140408 115013 000  >Database changed to CiwsDB
20140408 115013 000 < CON
20140408 115013 000  >Login successfull (CiwsDB)
20140408 115013 000 < DEVICE BATMAN_IMG
20140408 115013 000 < PUT IMG091I013.fits.gz 1
20140408 115013 000  >OK, receiving file IMG091I013.fits.gz
20140408 115013 000  >File received: IMG091I013.fits.gz
20140408 115013 000 < PDATA 143
20140408 115013 000  >OK, receiving data
20140408 115013 000  >End of data.
20140408 115013 000 FILENAME: 'IMG091I013.fits' DATAMODE: '321608'
20140408 115013 000 SUBMODE: '002' RUN_ID: '00000006'
20140408 115013 000 < NFR IMG091I013.fits.gz
20140408 115013 000 New file ready from |BATMAN_IMG|
20140408 115013 000  >NEW_FILE_READY message received:
IMG091I013.fits.gz
20140408 115013 000 hk_exec: DevicesLog_20_30_10 updated with ID=1
20140408 115013 000 post_exec: BATMAN_IMG 1 IMG091I013.fits.gz
20140408 115013 000 hk_postexec: copied to
/CiwsFiles/ImgsDBArchive/BATMAN_IMG/321608002/
20140408 115013 000 persist FILE 12 1 ...
20140408 115013 000 < BYE
20140408 115013 000  >Bye.
20140408 115013 000 Client thread is terminating...
20140408 115013 000 UserThread                          destroyed


-- Qui di seguito il logging del Client in spedizione file (al momento
sono
   5 le keyword lette dall'header e utilizzate per il logging):

% myFileClient IMG091I013.fits.gz
IMG091I013.fits.gz
ffname=IMG091I013.fits.gz
GetInfoHeader: There are 142 keywords.
GetInfoHeader: Transferred 5 kwds

GetInfoHeader: /home/nicastro/BatmanData/DirImages/IMG091I013.fits.gz
processed.
File: IMG091I013
TEL_ID:  BATMAN_IMG
DATAMODE: 321608
SUBMODE:  002
RUN_ID: 00000006
FILENAME:  IMG091I013.fits
OK, receiving file IMG091I013.fits.gz
File received: IMG091I013.fits.gz
OK, receiving data
End of data.
NEW_FILE_READY message received: IMG091I013.fits.gz

Done.
```

7.2 Prototipo AGILE

Il prototipo AGILE consiste essenzialmente nel rifacimento del Quick Look di AGILE utilizzando i file XML per descrivere Data Model, Data Format, Chain e Viste del Quick Look. Il focus principale di questo prototipo è quindi sul Quick Look. Il motivo di questa scelta è derivato dal fatto che il Quick Look è uno dei componenti CIWS più configurabili ed è quello che gli utenti finali essenzialmente vedono ed utilizzano. La scelta di lavorare sul Quick Look di AGILE è derivata dal fatto che forniva una copertura totale di tutti i requirement di un generico Quick Look del CIWS.

Nel seguito si riporta come esempio la configurazione del data type 3914. La configurazione del Quick Look Component per quanto riguarda il data type 3914 è piuttosto semplice mediante i files XML. Un CIWS User può svolgere facilmente questo passo. Per i FITS è possibile, inoltre, usare lo script di conversione *fits2xml* per generare gli XML a partire dalle informazioni presenti negli header del data format. Il convertitore si trova nella libreria libQLBase all'interno della cartella scripts.

```
<?xml version="1.0" encoding="UTF-8"?>
<ddl>
  <type id="3014_packets" name="PACKETS">
    <data>
      <binaryTable>
        <column name="PAKTNUMB" type="int32" />
        ...
        <column name="MMPAGEID" type="int16" />
        <column name="MMPKTCNT" type="int16" />
        <column name="APIDSEQC" type="byte" />
        <column name="OBTMSETT" type="int32" unit="us" />
        <column name="OBTSECTT" type="float64" unit="s" />
        <column name="OBTTIMFL" type="byte" />
        <column name="PLOPMODE" type="byte" />
        <column name="PKTSTRID" type="byte" />
        <column name="REDUNDID" type="byte" />
        <column name="CRCVALUE" type="int16" />
        <column name="TIME" type="float64" unit="us" />
        <column name="NEVENTS" type="int16" unit="event" />
      </binaryTable>
    </data>
  </type>
  <type id="3014_packets" name="EVENTS">
    <data>
      <binaryTable>
        <column name="PAKTNUMB" type="int32" />
        <column name="EVNUMBER" type="int16" />
        <column name="SSID" type="byte" />
        <column name="MSGVALI" type="byte" />
        <column name="TIMEDAY" type="int16" unit="d" />
        <column name="TIMEMSEC" type="int32" unit="ms" />
        <column name="Q1" type="float32" />
        <column name="Q2" type="float32" />
        <column name="Q3" type="float32" />
        <column name="Q4" type="float32" />
        <column name="TIME" type="float64" unit="ms" />
      </binaryTable>
    </data>
  </type>
</ddl>
```

The 3914 Data Format (3914_format.xml)

Nel data format 3914 sono presenti due tabelle in relazione 1-to-many, con *foreign key* PACKTNUMB.

Per il data type si è deciso di definire solo un sottoinsieme dell'intero data format. Si vuole visualizzare soltanto i quaternioni Q1, Q2, Q3 e Q4 usando il Quick Look.

```
<?xml version="1.0" encoding="UTF-8"?>
<type id="3914_0" DV="100">
  <field name="SSID" type="byte" />
  <field name="MSGVALI" type="byte" />
  <field name="Q1" type="float" />
  <field name="Q2" type="float" />
  <field name="Q3" type="float" />
  <field name="Q4" type="float" />
  <field name="TIME" type="double" unit="ms" />
</type>
```

The 3914_0 Data Type (3914_0_type.xml)

Una volta definiti il Data Type e il Data Format, è stato creato un XML per definire come mappare i field del Data Type all'interno del Data Format. Per il formato FITS il mapping è piuttosto immediato data la relazione 1:1 tra field e colonna. Basta definire una serie di link. La grammatica dell'attributo **ref** per i FITS è fissata come **table/column**.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping type="3914_0" inputFormat="fits" query="select 3914_events.* from 3014_events
inner join 3014_packets ON 3914_packets.PAKTNUMB == 3914_events.PAKTNUMB">
  <link field="SSID" ref="3914_events/SSID" />
  <link field="MSGVALI" ref="3914_events/MSGVALI" />
  <link field="Q1" ref="3914_events/Q1" />
  <link field="Q2" ref="3914_events/Q2" />
  <link field="Q3" ref="3914_events/Q3" />
  <link field="Q4" ref="3914_events/Q4" />
  <link field="TIME" ref="3914_events/TIME" />
</mapping>
```

The 3914_0 data mapping (3914_0_mapping.xml)

Dall'esempio precedente si può notare come il Quick Look component gestisce gli *inner join* di tabelle in relazione *1-many* a livello di input dei dati: basta definire la query sul mapping xml.

Si definisce quindi come visualizzare i dati descrivendo i grafici usando un XML per i graphs. Nell'esempio si vogliono visualizzare i quaternioni e il campo MSGVALI usando grafici temporali.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphs>
<!--#########################################################################-->
<!-- File: PDHU.param -->
  <graph id="3914_0_MSGVALI" type="temporal" name="3914_0:MSGVALI" title="3914_0: MSGVALI">
    <xaxis ref="3914_0/MSGVALI" label="VALIDITY" />
  </graph>
  <graph id="3914_0_Q1" type="temporal" name="QUAT1" title="3914_0: QUAT1">
    <xaxis ref="3914_0/Q1" label="quat" />
  </graph>
  <graph id="3914_0_Q2" type="temporal" name="QUAT2" title="3914_0: QUAT2">
    <xaxis ref="3914_0/Q2" label="quat" />
  </graph>
  <graph id="3914_0_Q3" type="temporal" name="QUAT3" title="3914_0: QUAT3">
    <xaxis ref="3914_0/Q3" label="quat" />
  </graph>
  <graph id="3914_0_Q4" type="temporal" name="QUAT4" title="3914_0: QUAT4">
    <xaxis ref="3914_0/Q4" label="quat" />
  </graph>
  ...
</graphs>
```

The 3914_0 graphs (PDHU.xml)

Per finire è necessario definire una Chain tramite XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<chains>
  ...
```

```
  <chain name="3914_0" type="3914_0" uniqueKey="3914" procChannel="14">
    <config DK="6000" DT="100" DS="100" qlrows="10" connect="yes"/>
  </chain>
  ...
</chains>
```

chains.conf

In Figura 7-6 è mostrato il risultato ottenuto con il Quick Look configurato come illustrato sopra.

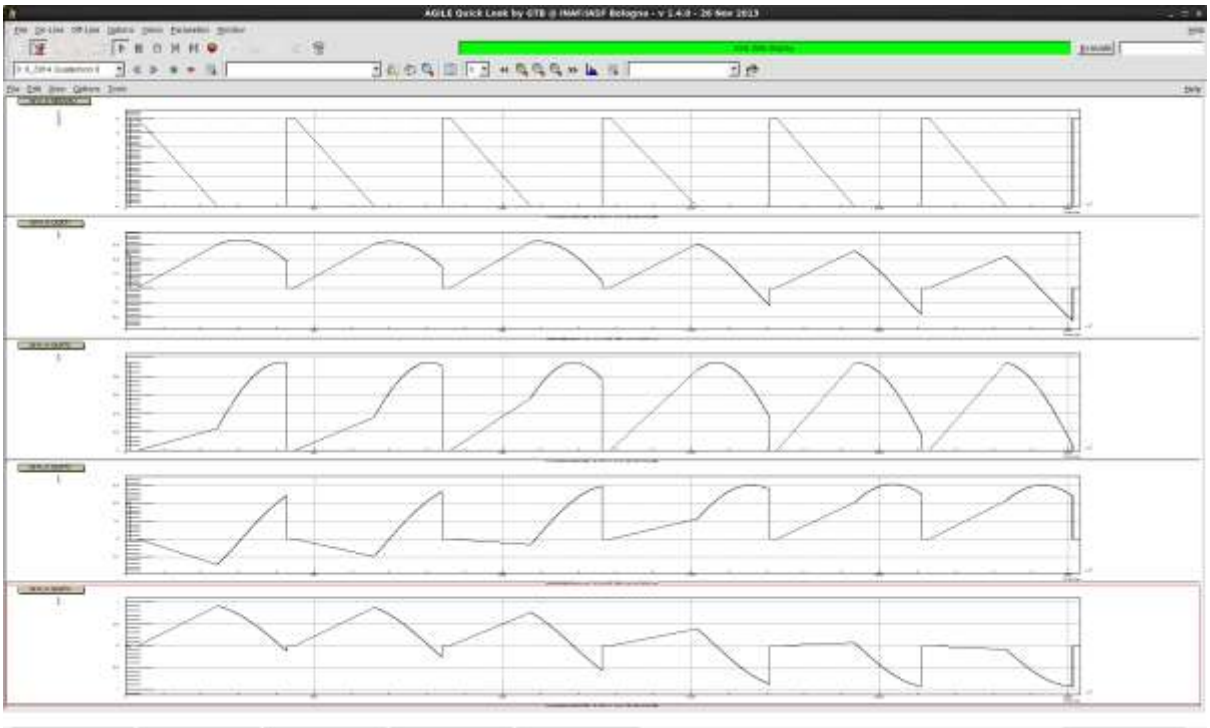


Figura 7-6 Una vista con diversi Pads sui dati contenuti nel data type 3914

La struttura architetturale del Quick Look consente al CIWS-FW Developer di ricreare o estendere le viste mediante la semplice estensione della classe *QLViewParameter*.

Con il Prototipo questa funzionalità è stata utilizzata per creare una vista personalizzata del modello 3d del satellite ricavata in base ai dati sui quaternioni contenuti nel data type 3914 (Figura 7-7).

Con il Prototipo è stata verificata anche la capacità del Quick Look di leggere immagini FITS e sovrapporvi i dati dei Reference Catalogues (Figura 7-8).

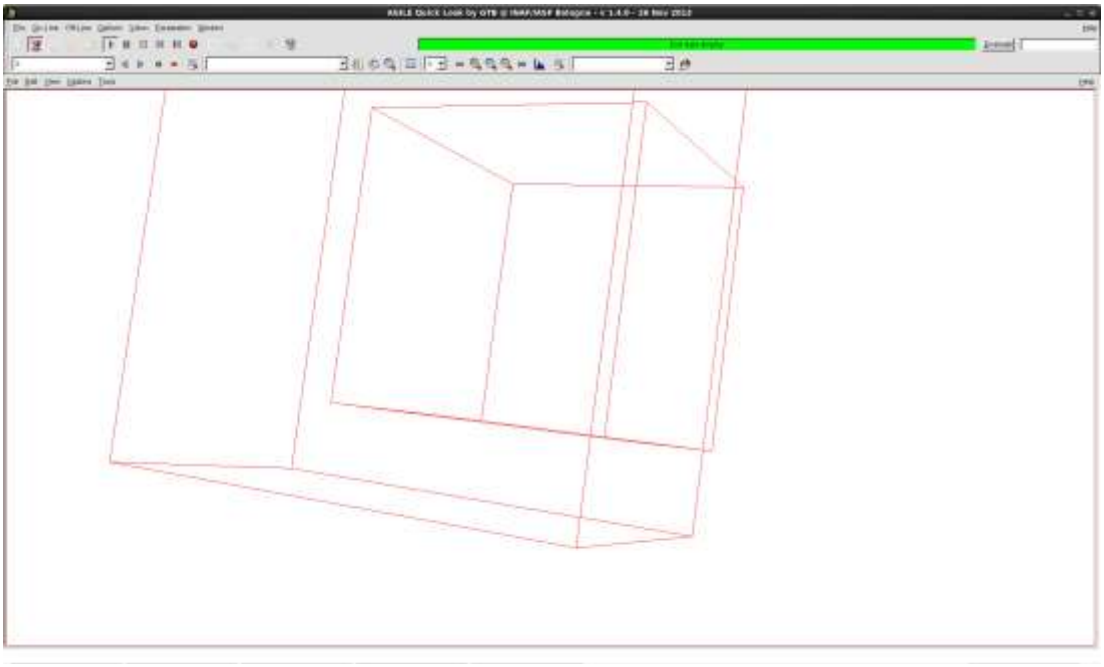


Figura 7-7 Un modello 3D semplificato di AGILE che ne rappresenta l'assetto in base ai quaternioni

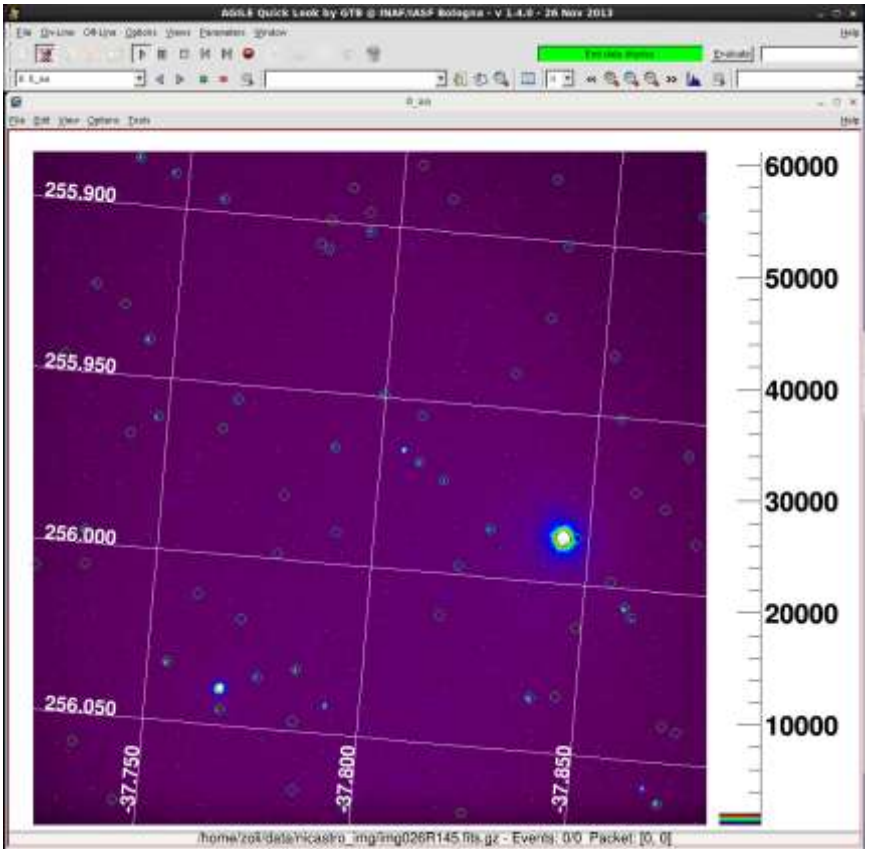


Figura 7-8 Sovrapposizione di mappe da Reference Catalogues