

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II



Dottorato in Ingegneria Aerospaziale, Navale  
e della Qualità

Indirizzo Aerospaziale

XIX ciclo

**Archiviazione on-line di dati provenienti  
da sonde relative all'esplorazione del  
Sistema Solare**

Francesco Carraro

Coordinatore del corso di Dottorato:  
Antonio Moccia, Dip. di Scienza e Ingegneria dello Spazio  
Università degli Studi di Napoli "Federico II"

Relatori:  
Luigi Colangeli, Vito Mennella  
INAF - Osservatorio Astronomico di Capodimonte



<b>INTRODUZIONE .....</b>	<b>1</b>
<b>PARTE I.....</b>	<b>5</b>
<b>IL CONTESTO INTERNAZIONALE E LO STANDARD PDS .....</b>	<b>5</b>
<b>CAPITOLO 1.....</b>	<b>7</b>
<b>ESPLORAZIONE DEL SISTEMA SOLARE .....</b>	<b>7</b>
1.1) SCENARIO INTERNAZIONALE .....	7
1.2) UNA STRATEGIA PER L'ESPLORAZIONE PLANETARIA .....	10
1.2.1) La progettazione .....	10
1.2.2) Le 4 "grandi aree" .....	12
1.3) LE MISSIONI .....	14
1.3.1) Pianeti giganti .....	14
1.3.2) Marte .....	14
1.3.3) Pianeti interni .....	15
1.3.4) Corpi minori .....	18
1.4) ARCHIVIAZIONE DEI DATI .....	20
<b>CAPITOLO 2.....</b>	<b>27</b>
<b>LO STANDARD PDS.....</b>	<b>27</b>
2.1) COS'È IL PDS .....	27
2.1.1) La struttura .....	27
2.1.2) La filosofia del PDS .....	29
2.2) STRUTTURA DI UN ARCHIVIO PDS .....	32
2.2.1) Struttura generale .....	32
2.2.2) Volume description file .....	33
2.2.3) Meta-data file .....	36
2.2.4) Data file .....	37
Caratteristiche generali .....	37
File dati e label - introduzione .....	39
Struttura di una label .....	40
Definizione della struttura dei dati .....	41
ODL – Object Data Language .....	42
<b>CAPITOLO 3.....</b>	<b>45</b>
<b>IL FORMATO DEI DATI DI VIMS-V .....</b>	<b>45</b>
3.1) I "CUBI" .....	45
3.1.1) L'oggetto QUBE .....	45
3.1.1.1) Il core .....	46

3.1.1.2) I suffix plane .....	47
3.1.1.3) La scrittura dei dati in un file .....	48
3.2) I CUBI DI VIMS.....	50
3.2.1) La label presente nei cubi .....	50
3.2.1.1) Parte iniziale della label .....	51
3.2.1.5) .....	56
<b>PARTE II .....</b>	<b>59</b>
<b>L'ARCHIVIO ON-LINE DI VIMS-V .....</b>	<b>59</b>
<b>CAPITOLO 4.....</b>	<b>61</b>
<b>LA CREAZIONE DEL DATABASE DI VIMS-V.....</b>	<b>61</b>
4.1) I PARAMETRI PER LE RICERCHE .....	61
4.1.1) I criteri per la scelta .....	61
4.1.2) I parametri scelti .....	64
4.2) MYSQL.....	65
4.3) LA PROCEDURA DI INSERIMENTO DEI DATI NEL DATABASE.....	68
4.3.1) inserimento_vims .....	70
4.3.2) uno_prepara_filesCubeListingNuovi.....	71
4.3.3) Preparazione dei file “cube_listing_final” .....	73
4.3.3.1) due_prepara_cube_listing_final .....	73
4.3.3.2) due_prepara_cube_listing_final.pl .....	74
4.3.4) tre_crea_elenco_cube_listing_final.....	76
4.3.5) inserimentoUpdateVims.php .....	77
<b>CAPITOLO 5.....</b>	<b>81</b>
<b>VIMS_PREVIEW_CREATION .....</b>	<b>81</b>
5.1) I CRITERI PER LA CREAZIONE DELLE IMMAGINI .....	82
5.2) LA STRUTTURA DEL PROGRAMMA .....	83
5.2.1) VIMS_Preview_Creation.pro .....	84
5.2.2) VIMS_Preview_Creation_eventcb1.pro .....	87
5.2.3) VIMS_Preview_Creation_eventcb2.pro.....	89
5.2.3.1) L'avvio della procedura.....	89
5.2.3.2) Costruzione delle cartelle per le immagini .....	90
5.2.3.3) Creazione dei file di controllo .....	92
5.2.3.4) La scelta della riga per i profili.....	95
5.2.3.5) La creazione dei profili di riga .....	96
5.2.3.6) La creazione delle immagini RGB .....	97
5.2.3.7) La fase finale .....	98
<b>CAPITOLO 6.....</b>	<b>101</b>

## **IL SISTEMA DI RICERCA DEI DATI..... 101**

6.1) LE POSSIBILITÀ DI UTILIZZO.....	102
6.2) I LINGUAGGI UTILIZZATI.....	103
6.3) LA HOME PAGE.....	104
6.3.1) Strumento non selezionato.....	105
6.3.2) Strumento selezionato.....	107
6.3.2.1) Area pubblica.....	107
6.3.2.2) Area riservata.....	111
6.4) INDEX_JAVASCRIPT.JS.....	113
6.4.1) Verifica del browser usato dall'utente.....	113
6.4.2) Gestione dei <i>layer</i> e dei rispettivi menu.....	115
6.4.3) Gestione delle date.....	116
6.4.4) Altre funzioni.....	117
6.5) GLI SCRIPT DI RICERCA.....	118
6.5.1) Ricerca per singolo parametro.....	119
6.5.2) Ricerca con più parametri.....	120
6.5.3) Ricerca per intervallo temporale.....	121
6.5.4) Ricerca per sessione.....	121
6.6) LE TABELLE CON I RISULTATI.....	122
6.6.1) Rows Table.....	123
6.6.2) Pictures Table.....	126
6.6.3) Download Table.....	128
6.7) LA PAGINA DI QUICK-LOOK.....	130

## **CAPITOLO 7..... 133**

### **VIMS READER..... 133**

7.1) CONSIDERAZIONI PRELIMINARI.....	133
7.1.1) La scelta di utilizzare Java.....	133
7.2) LA STRUTTURA DEL PROGRAMMA.....	136
7.3) IL CARICAMENTO DELL'APPLET.....	138
7.3.1) La richiesta al server.....	138
7.3.2) Il metodo <code>init()</code> .....	138
7.4) L'OGGETTO CUBO.....	140
7.4.1) Il costruttore.....	141
7.4.2) Lettura dei valori delle keyword.....	142
7.4.3) Lettura del cubo.....	143
7.4.4) Lettura del file con le $\lambda$ .....	145
7.4.5) I metodi per il recupero dei dati.....	146
7.4.5.1) profilo di riga dallo spettro.....	147
7.4.5.2) profilo di colonna.....	148
7.4.5.3) immagine monocromatica.....	149
7.4.5.4) spettro.....	150

7.4.5.5) altri .....	150
7.5) L'INTERFACCIA GRAFICA .....	151
7.6) VIEW PANEL .....	153
7.6.1) ViewWindowBase.java .....	153
7.6.2) SpectrumDraw.java .....	154
7.6.3) ImageDraw.java .....	156
7.6.4) ProfileWindowBase.java .....	157
7.6.5) VisProfileDraw.java .....	158
7.6.6) IRProfileDraw.java .....	162
<b>CONCLUSIONI.....</b>	<b>163</b>
<b>APPENDICE A .....</b>	<b>167</b>
<b>LA MISSIONE CASSINI-HUYGENS E LO STRUMENTO VIMS-V</b> <b>.....</b>	<b>167</b>
A.1) LA MISSIONE CASSINI-HUYGENS .....	167
A.1.1) La nascita della missione .....	167
A.1.2) Le fasi della missione .....	168
A.2) GLI OBIETTIVI SCIENTIFICI .....	171
A.2.1) Atmosfera di Saturno.....	171
A.2.2) Anelli di Saturno.....	173
A.2.3) Titano .....	173
A.2.4) Satelliti ghiacciati .....	175
A.3) VIMS-V .....	178
A.3.1) Caratteristiche tecniche.....	178
A.3.2) Sistema ottico .....	179
A.3.3) Telescopio Shafer .....	181
A.3.4) Spettrometro Offner.....	182
A.3.5) Il reticolo .....	184
A.4.6) La CCD.....	186
<b>BIBLIOGRAFIA .....</b>	<b>189</b>

# Introduzione

Il lavoro svolto per l'attività di dottorato che verrà descritto nella presente tesi si inserisce all'interno di un contesto italiano ed internazionale ricco di missioni volte all'esplorazione del Sistema Solare. Il settore della ricerca, soprattutto nel campo dell'esplorazione spaziale, è stato da sempre caratterizzato da una forte propensione alla collaborazione, tra enti e persone, a livello internazionale. Questo carattere di cooperazione si è, col tempo, rafforzato, soprattutto grazie alla nascita ed allo sviluppo di internet, il quale ha reso infinitamente più facili e veloci le comunicazioni e lo scambio di dati ed informazioni.

Proprio l'esistenza del web, con le possibilità da esso offerte di connettersi in ogni istante ad un qualunque server del mondo, ha dato modo alla comunità scientifica di cominciare ad immaginare, in una prima fase, e realizzare, poi, delle banche dati, facilmente accessibili, che permettessero, a chiunque fosse interessato ad un determinato progetto di ricerca, di reperire tutto il materiale utile per il proprio lavoro. L'esperienza accumulata col susseguirsi delle missioni, inoltre, ha permesso di far comprendere a tutti quanto importante ed utile, al fine del ritorno scientifico delle missioni stesse, fosse l'opportunità di poter disporre di simili strumenti di raccolta e distribuzione dei dati.

Si può, dunque, nell'ambito di un tale scenario, comprendere l'importanza che può avere una banca dati il cui scopo sia quello di fornire un aiuto alle attività di ricerca della comunità italiana dei planetologi.

Lo scopo del lavoro qui presentato è stato proprio quello di progettare e realizzare un archivio dati, accessibile attraverso la rete, il quale fornisca un servizio di ricerca dei dati stessi che consenta di discriminare facilmente, all'interno della mole di file relativi ad un

determinato strumento, quelli utili per un determinato lavoro dai restanti. Al fine di fornire un servizio efficiente e realmente utile, sono stati predisposti, per ogni strumento, differenti criteri di ricerca e tre diverse modalità di visualizzazione dei risultati.

Oltre a ciò è stato messo a disposizione dei ricercatori un software che consente di aprire i file di dati e di visualizzarne il contenuto, in maniera tale da permettere all'utente di conoscere esattamente il contenuto del file stesso e di valutarne, quindi, l'interesse da un punto di vista scientifico.

La presente tesi è stata suddivisa in due sezioni principali. Nella prima, contenente i primi tre capitoli, vengono affrontati tre argomenti preliminari alla descrizione di quanto realizzato: innanzitutto viene delineato il contesto internazionale all'interno del quale inquadrare il progetto di una banca dati dedicata alla comunità scientifica italiana. Oltre a ciò, viene dato conto dell'utilità di questo lavoro, in termini di ritorno scientifico, presentando esempi di missioni che, grazie ad un simile strumento, hanno dato origine a numerose pubblicazioni ed altre dalle quali, al contrario, non essendo state investite risorse per la creazione di archivi dati, non si è avuto un altrettanto importante contributo.

Al fine di non appesantire la stesura della tesi e di non ripetere più volte gli stessi concetti, si è deciso di utilizzare lo strumento VIMS-V, a bordo della missione Cassini, come esempio per la spiegazione del funzionamento dell'intero archivio.

Nel capitolo due è stato illustrato, in maniera esauriente, lo standard PDS utilizzato per l'archiviazione dei dati di tutte le missioni relative all'esplorazione del Sistema Solare. Nel terzo capitolo, infine, si è provveduto a mostrare come tale standard sia stato utilizzato nel caso di VIMS-V.

La seconda parte della tesi è stata, invece, dedicata alla spiegazione di tutto il lavoro di progettazione e realizzazione dell'archivio vero e proprio. Tale resoconto è stato suddiviso in quattro parti, esposte in altrettanti capitoli. Nel capitolo quattro viene descritto tutto il processo di inserimento dei dati nel database,



operazione eseguita da una serie di script che vengono illustrati uno per uno.

Nel capitolo cinque si parla del programma realizzato allo scopo di creare delle immagini di preview utilizzate durante alcune fasi del processo di ricerca per mostrare agli utenti i risultati delle interrogazioni del database da essi impostate.

Il capitolo sei è dedicato alla descrizione di tutti i processi di ricerca messi a disposizione degli utenti e delle possibili tabelle risultato che gli utenti stessi possono utilizzare per visualizzare i risultati da essi ottenuti.

Il capitolo sette, infine, mostra l'applet realizzata allo scopo di fornire la possibilità ai ricercatori di visualizzare in tempo reale gli spettri e le immagini monocromatiche contenute all'interno dei file acquisiti da VIMS-V.



## Parte I

# Il contesto internazionale e lo standard PDS

Nella prima parte della tesi viene fornita una descrizione del contesto all'interno del quale inquadrare il lavoro svolto durante i tre anni nel corso dei quali l'attività è stata portata avanti e viene fornita una esauriente descrizione dello standard PDS, utilizzato per l'archiviazione dei dati. Questa parte della tesi può essere suddivisa in due sezioni: la prima riguardante l'ambito nel quale inserire il lavoro svolto, la seconda riguardante lo standard PDS in generale e la sua applicazione nel caso di VIMS-V, del quale viene fornita una descrizione, insieme alla missione nella quale è impiegato, in appendice.

Il primo argomento trattato, all'interno del capitolo 1, è una descrizione dell'utilità, all'interno del contesto internazionale ed italiano della ricerca, del lavoro di realizzazione di un archivio online di dati che abbia un compito di supporto al lavoro di ricerca della comunità dei planetologi italiani.

Nei due capitoli seguenti, viene presentata una esauriente descrizione dello standard PDS e della sua applicazione ai dati di VIMS-V, il quale viene preso ad esempio per la descrizione del funzionamento dell'intero sistema realizzato. Una simile scelta è stata fatta in virtù della somiglianza, in linea generale, del funzionamento dei processi di ricerca messi a disposizione degli utenti. Descrivere ciò che è stato realizzato per tutti gli strumenti avrebbe, quindi, rappresentato una inutile ripetizione degli stessi

concetti. Nel capitolo due, quindi, viene descritto lo standard in tutti i suoi aspetti, dalla filosofia generale che ne è alla base fino alla struttura completa di un archivio. Il tutto in un ambito generale, poiché tale standard è nato e si è evoluto per poter essere utilizzato in tutte le missioni di esplorazione del Sistema Solare, a prescindere dal tipo di missione.

Nel capitolo tre, infine, viene descritto come lo standard PDS sia stato utilizzato nel caso dello strumento VIMS-V.

# Capitolo 1

## Esplorazione del Sistema Solare

### 1.1) Scenario internazionale

Negli ultimi anni il Sistema Solare è stato esplorato estensivamente e, conseguentemente, la conoscenza che ne abbiamo si è estesa sia verso l'interno sempre più vicino al Sole, la nostra stella, sia verso l'esterno, fino ai confini del nostro sistema e oltre, grazie alle sonde *Voyager*. Al contempo la conoscenza del Sole è giunta al punto che è oggi possibile analizzare, senza soluzione di continuità, la stella dal suo interno alle regioni più esterne dell'atmosfera, dove accelera il vento solare che investe tutti i pianeti.

L'ESA ha formulato un suo programma, precedentemente denominato "*Horizon 2000 plus*", ed ora ribattezzato "*Cosmic Vision*", il quale prevede una coordinata esplorazione del Sistema Solare, che prevede un esame del corpo centrale, mediante la missione "*Solar Orbiter*"; lo studio del vento solare, grazie alla missione "*Double Star*"; una innovativa analisi delle interazioni tra vento solare ed oggetti planetari e, infine, una dettagliata indagine dei pianeti, sia dal punto di vista geologico che geofisico. A questo ultimo gruppo appartengono le missioni "*Cassini-Huygens*", "*Mars Express*", "*Venus Express*", "*Rosetta*" e "*BepiColombo*".

La NASA ha anch'essa lanciato un programma di esplorazione del Sistema Solare che si muove su due direttrici parallele: da un lato è presente un discorso di comprensione

dell'ambiente planetario nel quale l'uomo dovrebbe potersi muovere senza subire danni; dall'altro si ritiene necessario comprendere i complessi meccanismi e le relazioni che legano tra loro i diversi corpi che sono presenti nel Sistema Solare. Così è nato da un lato il complesso approccio che si è espresso nel programma “*Living with a Star (ILWS)*” al quale si può idealmente accostare il programma di esplorazione di Marte, e dall'altro il programma “*Origins*” che si propone di studiare quale sia l'origine del nostro Sistema, degli oggetti che lo costituiscono, e della vita che su almeno uno di essi si è instaurata.

La prossimità Terra-Sole rende l'investigazione della nostra stella unica non solo nell'ambito dell'astrofisica, ma anche per la stretta connessione fisica esistente tra i due corpi. Il Sole interagisce con l'ambiente terrestre sia dal punto di vista energetico, sia attraverso l'influenza che esercita sull'atmosfera e la magnetosfera terrestre alle quali è connesso tramite il vento solare. Le perturbazioni di origine solare che si propagano nell'eliosfera hanno, infatti, notevole impatto sul pianeta Terra e soprattutto sul funzionamento dei sistemi tecnologici più sofisticati. Gli effetti sono particolarmente critici sul funzionamento di satelliti, di sistemi di navigazione satellitare e di trasmissione di energia ed informazione al suolo, nonché sulla salute degli uomini nello spazio. L'impatto è quindi notevole su programmi spaziali, esplorazione spaziale, reti di telecomunicazioni e navigazione satellitare, difesa.

Queste considerazioni hanno portato all'abbandono del vecchio concetto che vede la planetologia come una scienza e sé, separata dalla fisica Solare: un pianeta si evolve all'interno di un preciso ambiente di radiazione, subisce la prolungata interazione con il vento solare, che in assenza di campo magnetico, ne erode l'atmosfera, subisce mutamenti climatici in parte derivanti dalle variazioni della attività solare. Tuttavia i complessi meccanismi che danno luogo a questi fenomeni non sono di facile comprensione e spesso richiedono un approccio integrato: così negli ultimi anni tutte le missioni planetarie più importanti hanno caricato a bordo strumenti in grado di misurare campi magnetici, plasm, e atomi neutri.

Nello stesso tempo si sta meglio comprendendo come l'evoluzione climatica di un pianeta sia condizionata da meccanismi diversi, che dipendono non solo dalla posizione del pianeta rispetto al Sole ma anche dalla composizione della sua atmosfera, che a sua volta dipende anche dalla evoluzione interna. Così fenomeni vulcanici distribuiti su larga parte della superficie di un pianeta e di lunga durata possono contribuire a generare una atmosfera ricca in gas e particelle, che successivamente influirà sulla evoluzione in temperatura del pianeta stesso. È noto, infatti, che la presenza di una atmosfera ricca in gas serra può contribuire ad innalzare la temperatura di un pianeta molto al di là di quella di corpo nero. Esempio di ciò è il pianeta Venere.

Accanto allo studio del Sole e dei pianeti del Sistema Solare si colloca, inoltre, l'indagine relativa ai corpi minori, per i quali il processo di elaborazione della materia che si è prodotto nelle fasi di formazione ed evoluzione dei pianeti si è interrotto. I corpi che sono presenti nella fascia asteroidale presentano una variegata struttura superficiale e composizione chimica, che conserva traccia di una complessa evoluzione chimica e dinamica collisionale. Lo studio di tali oggetti può darci informazioni sul tempo in cui hanno cominciato ad innescarsi i processi di differenziazione di oggetti planetari.

I corpi minori, inoltre, hanno anche un altro ruolo nella evoluzione dei corpi planetari: processi collisionali estesi hanno modificato le superfici planetarie, hanno ceduto calore nel corso di processi da impatto, hanno forse portato grandi quantità di volatili ai pianeti interni, contribuendo, probabilmente, alla formazione degli oceani terrestri. Sembra ormai dimostrato che processi da impatto hanno interagito con la evoluzione della vita sulla Terra e non si può escludere che possano interagire con l'evoluzione della nostra specie anche in futuro. Una sfida per i prossimi anni sarà riuscire ad identificare il luogo di provenienza di meteoriti, quali le Eucriti, che sono tra i materiali più antichi differenziati. Proprio allo scopo di investigare la natura di tali corpi è stata proposta dalla NASA la missione "*Dawn*".

Allo scopo di studiare la natura dei materiali più primitivi presenti all'interno del Sistema Solare è invece necessario spingersi più lontano, là ove le temperature sono abbastanza basse da aver permesso ai volatili di preservarsi. E' per questo motivo che l'ESA ha investito una risorse nello studio delle comete e, dopo il notevole successo della missione "*Giotto*", ha messo a punto una ambiziosa missione destinata all'osservazione di corpi di piccole dimensioni, la missione "*Rosetta*", partita nel marzo del 2004 e destinata ad incontrare la cometa "67/P Churyumov-Gerasimenko" nel 2014.

## 1.2) Una strategia per l'esplorazione planetaria

Il primo passo che è necessario compiere nel momento in cui si vuole intraprendere una impresa impegnativa come l'esplorazione del Sistema Solare è quello di definire una strategia che renda sostenibile una tale azione. I problemi da affrontare sono, infatti, molteplici.

### 1.2.1) La progettazione

Il primo problema è costituito dalla necessità di raggiungere corpi molto lontani e da ciò consegue l'esigenza di dover realizzare delle sonde che siano adatte a sopravvivere in un ambiente fortemente ostile quale è quello dello spazio interplanetario. Ogni sonda ospita a bordo strumenti di diversa natura, destinati a differenti tipi di analisi e con diverse finalità scientifiche e, ovviamente, per ognuno di essi si pone il problema di dover sopportare lo stress del lancio e di arrivare in perfetta efficienza all'appuntamento con il proprio obiettivo, il quale può essere raggiunto dopo "solo" qualche mese o addirittura dopo alcuni anni. Tutto ciò richiede una lunga fase di progettazione nella quale vanno tenuti in conto i fattori che potrebbero determinare cedimenti o perdite di efficienza da parte



dello strumento durante il tragitto. Risulta ovvio che tali considerazioni sono fortemente influenzate dal percorso che dovrà compiere la sonda nel suo viaggio verso il pianeta o il corpo che è destinata ad osservare. Si devono, infatti, valutare con molta attenzione le condizioni di temperatura e di ambiente di radiazione che caratterizzano le diverse regioni del Sistema Solare.

Un viaggio, o una parte di viaggio, nella direzione della parte più interna del Sistema Solare pone, ad esempio, dei problemi di surriscaldamento delle componenti esposte alla radiazione solare maggiori di quanto faccia un viaggio nella direzione della parte esterna del Sistema Solare stesso. Si va, infatti, dalle torride regioni interne, ove il problema è dissipare l'eccessiva radiazione termica e schermarsi dal bombardamento dei raggi cosmici solari, alle gelide regioni esterne, ove acquisire energia sufficiente è il problema chiave. Al tempo stesso, a parità di percorso, anche una diversa esposizione delle superficie della sonda alla radiazione solare causa notevoli escursioni termiche. Questa circostanza impone di dover utilizzare materiali e tecnologie che siano in grado di resistere a condizioni estreme di temperatura.

Ciò implica il dover utilizzare materiali che siano in grado di adattarsi a situazioni in cui la temperatura sia molto alta o molto bassa, ma può anche voler dire individuare materiali che abbiano la capacità di adattarsi ad entrambe le situazioni appena descritte. Tale sistema di adattamento passivo, tuttavia, non è l'unico sistema possibile per il controllo della temperatura. Sistemi attivi che utilizzano dei *cryocooler* per mantenere la temperatura sempre in un intervallo accettabile di valori sono frequentemente utilizzati.

A ciò è da aggiungere, come accennato in precedenza, la necessità di dover tener conto del notevole stress che ogni apparecchiatura subisce al momento del lancio, quando il razzo, con tutti i suoi componenti ed il suo carico, è sottoposto ad imponenti sollecitazioni meccaniche dovute all'accelerazione causata dalla spinta dei razzi ed alle inevitabili conseguenti vibrazioni.

È proprio in conseguenza di queste considerazioni che, durante la fase di studio e progettazione degli strumenti e delle

sonde, una grande quantità di tempo è spesa nella realizzazione dei modelli termici e meccanici, i quali sono sottoposti a severi test proprio per verificarne le capacità di sopportazione delle condizioni alle quali saranno sottoposte durante il loro ciclo di vita.

Un tale notevole sforzo richiede la partecipazione sia della comunità scientifica che dell'industria. I planetologi devono stabilire le esigenze scientifiche sia della missione nel suo insieme sia dei singoli strumenti, fornendo al tempo stesso un quadro il più possibile dettagliato delle condizioni ambientali nelle quali le sonde con i loro strumenti si troveranno a dover sopravvivere durante tutte le fasi della missione. È poi compito dell'industria proporre soluzioni e progettare strumenti ed apparecchiature che soddisfino tutti i requisiti tecnici necessari affinché quanto realizzato sia poi adatto a lavorare nelle condizioni ambientali previste. Tutto ciò può, ovviamente, avvenire solo in presenza di una stretta collaborazione tra ricercatori ed industria che si concretizzi in un continuo scambio di informazioni sia durante la fase di progettazione vera e propria, sia durante le successive sessioni di test. Proprio quello di instaurare una tale intensa collaborazione con l'industria è stato uno dei primi sforzi compiuti dalla comunità scientifica ed il buon esito di tale sforzo, visibile nel successo delle missioni più recenti, quali "*Mars Express*", "*Venus Express*" e "*Rosetta*" o delle meno recenti, quale la missione "*Cassini*", costituisce uno dei più importanti traguardi di tale comunità.

### **1.2.2) Le 4 "grandi aree"**

I pianeti e gli altri corpi che formano il Sistema Solare hanno, in generale, caratteristiche fisiche e chimiche differenti gli uni dagli altri. Notevoli sono le differenze riscontrabili confrontando le dimensioni dei vari corpi e le rispettive morfologie; una importante differenza è data, inoltre, dalla presenza o meno di attività sismica e vulcanica. Anche per ciò che riguarda l'atmosfera si hanno notevoli e vistose diversità: quasi tutti i pianeti e alcuni satelliti sono caratterizzati dalla presenza di atmosfera, seppure con notevoli

discrepanze passando da un corpo all'altro, mentre altri ne sono totalmente privi.

Pur in un simile quadro di difformità è comunque possibile individuare, nell'ambito dell'intero Sistema Solare, pochi gruppi di corpi che presentano delle caratteristiche comuni che li differenziano in maniera sostanziale dai corpi appartenenti agli altri gruppi.

Tali gruppi sono in numero di 4 ed individuano altrettante "grandi aree" di esplorazione del Sistema Solare. La suddivisione generalmente accettata da tutti è la seguente:

*Pianeti giganti:* Giove, Saturno ed i loro satelliti;

*Marte;*

*Pianeti interni:* Mercurio, Venere, Terra e Luna;

*Corpi minori:* comete, asteroidi, NEO;

Dall'osservazione dei 4 gruppi si possono facilmente cogliere significative differenze innanzitutto nelle caratteristiche principali dei corpi: al primo gruppo appartengono i pianeti giganti e gassosi mentre gli altri sono o pianeti in prevalenza rocciosi, con o senza atmosfera, o, sostanzialmente, delle vere e proprie rocce più o meno stazionarie in una qualche regione del Sistema Solare. La decisione di studiare un tipo di corpo piuttosto che un altro condiziona, quindi, la scelta degli strumenti che dovranno essere montati a bordo di una sonda.

È chiaro poi che le condizioni di irraggiamento solare al quale è sottoposta una navicella spaziale variano in base all'oggetto selezionato per lo studio. Mentre per i corpi interni le condizioni sono tali da poter usare dei pannelli solari per alimentare la navicella stessa e, addirittura, da doversi preoccupare di fornire alla suddetta un'efficace protezione dalla radiazione proveniente dal Sole, spostandosi verso l'esterno del Sistema Solare, e quindi andando ad osservare i pianeti giganti o i corpi minori presenti all'esterno del sistema, l'irraggiamento scende notevolmente ed il problema si sposta a come fornire sufficiente energia allo *spacecraft* per il corretto funzionamento suo e di tutti gli strumenti in esso alloggiati.

## 1.3) Le missioni

### 1.3.1) Pianeti giganti

Il settore dei pianeti giganti è, tra tutti, quello che al momento è meno presente nei piani futuri delle agenzie spaziali. Ciò avviene, ovviamente, non per mancanza di interesse al riguardo ma perché dopo un lungo viaggio di 7 anni, nel 2004, finalmente, la sonda Cassini ha raggiunto Saturno cominciando a mandare una notevole mole di dati che terrà impegnati i ricercatori per molto tempo. La missione, che ha una durata nominale di 4 anni, verrà descritta maggiormente in appendice.

### 1.3.2) Marte

L'esplorazione di Marte rappresenta uno degli obiettivi di maggiore rilievo nei programmi delle agenzie spaziali mondiali. Infatti, Marte è il pianeta "terrestre" con le maggiori similitudini con la nostra Terra, ma, al tempo stesso, la sua storia evolutiva ha dato luogo a sostanziali differenze sia geologiche, che chimiche e fisiche. L'esplorazione di Marte ha importanti implicazioni in varie branche della moderna esplorazione del Sistema Solare e rappresenta un punto di riferimento essenziale per l'approfondimento scientifico della formazione e dell'evoluzione dei "pianeti terrestri".

Pertanto tale esplorazione risponde ad una sempre crescente domanda di conoscenze nel settore dell'esplorazione del Sistema Solare, con specifico riferimento agli studi atmosferici e climatici, alle analisi geochimiche e mineralogiche, alla ricerca di serbatoi sub-superficiali di acqua e di tracce di forme di vita estinta o ancora esistente. In una prospettiva a più lungo termine, Marte rappresenta, dopo le imprese lunari, l'obiettivo di riferimento per l'esplorazione planetaria umana.

A dimostrazione del fatto che Marte è da sempre uno degli obiettivi principali nell'ambito dell'esplorazione del Sistema Solare

---

basti ricordare le già numerose missioni che lo hanno visto quale obiettivo di studio. Tra queste vale la pena citare le varie missioni della serie “*Mars*”, le sonde “*Viking*” che restituirono numerose foto, la famosa missione “*Mars Pathfinder*” che nel 1997 portò per la prima volta un piccolo rover a calcare la superficie del pianeta rosso, e i recenti “*Spirit*” ed “*Opportunity*”. In ambito europeo, la missione “*Mars Express*”, missione tuttora in corso, rappresenta il primo passo concreto dell'ESA per una partecipazione diretta all'esplorazione di Marte. In prospettiva futura, il programma “*Aurora*” offrirà l'opportunità di uno sviluppo focalizzato su Marte e sensibile alle ricerche esobiologiche.

**Mars Express:** La sonda, al momento del lancio, era composta dal modulo *Mars Express Orbiter* e dal lander *Beagle-2*, progettato per studiare la geologia del pianeta e l'eventuale presenza di vita. Si sperava che il lander potesse fornire informazioni definitive sulla capacità del pianeta di supportare forme di vita nel passato ma purtroppo è andato perso durante la fase di discesa. Molti degli strumenti sono la copia di strumenti persi durante la fallita missione Russa “*Mars 96*”, tra questi vanno citati i due strumenti italiani: lo spettrometro di Fourier, *PFS* ed il radar *MARSIS*. Gli obiettivi scientifici principali della missione riguardano la ricerca di acqua nel sottosuolo ma anche la determinazione della natura delle strutture mineralogiche presenti sulla superficie del pianeta e il ruolo avuto, in passato, dall'acqua presente sulla superficie, nella loro formazione. La missione si occupa, inoltre, dello studio dell'atmosfera, della sua interazione con la superficie del pianeta e con l'ambiente esterno.

### 1.3.3) Pianeti interni

I pianeti terrestri forniscono una opportunità unica per studiare i processi che hanno portato alla formazione di mondi abitabili quali la Terra. Mercurio e la Luna, infatti, conservano il ricordo fossile di eventi passati che hanno avuto luogo anche sulla Terra, ma che su di essa sono stati mascherati dalla successiva

evoluzione. Essi, al contrario, preservano il ricordo del bombardamento primordiale che chiuse il processo di formazione planetaria. Comprendere le differenze e le analogie tra questi due oggetti ci permetterà di avere una più chiara idea delle prime fasi del processo di formazione planetaria.

Terra e Venere, che per molti versi possono essere considerati pianeti gemelli, forniscono un laboratorio naturale per lo studio delle atmosfere planetarie e della loro evoluzione. Al momento attuale l'esplorazione dei pianeti interni si basa su 3 missioni ESA, "*SMART-1*", "*Venus Express*" e "*Bepi Colombo*". La prima missione ha avuto termine da poco tempo, la seconda è in corso, mentre la terza è in fase di preparazione, il lancio è previsto per il 2013.

**SMART-1:** lanciata nel 2003, è una missione lunare volta a provare per la prima volta la propulsione ionica per un viaggio interplanetario. Tra gli obiettivi scientifici della missione vale la pena citare un nuovo e dettagliato studio della superficie e della sua topografia, la ricerca di acqua e una conferma, o eventualmente una smentita, della teoria secondo la quale la Luna si sarebbe formata in seguito alla collisione di un enorme asteroide con la Terra.

**Venus Express.** La prima fase dell'esplorazione venusiana tra il 1962 ed il 1985 con "*Venera*", "*Pioneer*", "*Venus*" e "*Vega*", ha fornito una descrizione generale delle condizioni chimico-fisiche dell'atmosfera e della superficie. Ciò ha anche generato una serie di quesiti, ad oggi insoluti, relativi ai processi fisici che determinano tali condizioni. La missione radar della NASA "*Magellano*" ha permesso di studiare la complessa geologia venusiana evidenziando la presenza di un esteso vulcanismo, che ha forse influenzato l'evoluzione della atmosfera stessa. Tuttavia restano ancora misteriosi molti aspetti della evoluzione dell'atmosfera stessa e di come sia stato possibile generare un così marcato effetto serra, a causa del quale le condizioni di temperatura della superficie di Venere sono assai superiori a quelle che avrebbe un pianeta privo di atmosfera alla stessa distanza di Venere dal Sole. Infatti l'intensità dell'effetto serra su Venere è talmente notevole da portare la

temperatura della superficie a più di 700 K. E' chiaro che lo studio della atmosfera di Venere e la comprensione dei meccanismi fisico-chimici che ne hanno determinato l'evoluzione porterà ad una assai miglior comprensione di quanto l'effetto serra possa influire sulla evoluzione dell'atmosfera del nostro pianeta. La missione ha, tra gli scopi principali, lo studio della composizione dell'atmosfera, delle strutture in essa presenti e della loro evoluzione. Di rilievo, inoltre, la determinazione del bilancio radiativo del pianeta e lo studio della superficie di Venere.

Tra i punti di forza della missione sono lo spettrometro ad immagini *VIRTIS*, il cui gemello è già integrato a bordo della missione "*Rosetta*" e lo spettrometro di Fourier denominato *PFS*, gemello di quello a bordo della navicella "*Mars Express*".

**BepiColombo.** Mercurio è il pianeta più vicino al Sole e in quanto tale è importante per definire e verificare i modelli di formazione del Sistema Solare. In particolare Mercurio, Venere, Terra e Marte costituiscono la famiglia dei pianeti terrestri e le informazioni su ciascuno di essi sono essenziali per tracciare l'origine e l'evoluzione dell'intero gruppo. Inoltre la conoscenza dell'origine e dell'evoluzione dei pianeti terrestri gioca un ruolo importante nel capire le condizioni in cui si è formata la vita sulla Terra, e quelle in cui si potrebbe formare in altri sistemi planetari.

Gli obiettivi di questa missione sono: lo studio delle formazioni presenti sul pianeta, la sua geologia, e la sua composizione. Verrà, inoltre, analizzato il suo campo magnetico e, fatto di notevole rilievo, verrà eseguita un test volto a verificare la teoria della relatività generale di Einstein.

La missione BepiColombo è di gran lunga la più complessa mai programmata dall'ESA ed è costituita da 3 satelliti indipendenti:

Il *Mercury Planetary Orbiter* (MPO), un modulo stabilizzato a 3 assi, in grado di puntare sempre al nadir e che ruoterà intorno al pianeta in un'orbita piuttosto bassa. Esso sarà dedicato al *remote sensing*, al *radio science* e all'osservazione degli asteroidi.

Il *Mercury Magnetospheric Orbiter* (MMO), un modulo che ruoterà intorno al proprio asse orbitando intorno a Mercurio in un'orbita particolarmente eccentrica, sarà dedicato alla misura di campi, onde e particelle. Esso verrà trasportato in prossimità di Mercurio da un modulo di servizio, in pratica un'altra navicella che resterà in orbita probabilmente ospitando un telescopio per la rivelazione dei NEO.

Il *Mercury Surface Element* (MSE), un lander che eseguirà misure sulla superficie di Mercurio.

La missione "*Bepi Colombo*" è fondamentale poiché fornisce dati chiave a 3 diverse comunità, quella solare, quella planetologica e quella che studia la fisica di base.

### **1.3.4) Corpi minori**

Gli oggetti primitivi del Sistema Solare sono quelli che hanno subito una alterazione limitata. E' infatti impossibile supporre che la materia primordiale da cui il Sistema Solare si è formato possa essere rimasta inalterata dall'epoca della sua formazione. Tuttavia, nei 4.6 miliardi di anni in cui il Sistema Solare si è evoluto possono essere intervenute delle modificazioni, anche se limitate, che possono essere termiche, quando generate dalla interazione con la radiazione solare, dovute al decadimento radioattivo degli elementi di lunga vita in essi presenti, oppure possono essere il risultato di una complessa evoluzione collisionale.

I corpi meno alterati sono quelli di dimensioni piccole, (da qualche chilometro a qualche centinaio di chilometri) e si trovano prevalentemente nella regione oltre l'orbita di Marte. A questa classe di oggetti appartengono sia gli asteroidi propriamente detti che le comete, fino ai corpi ghiacciati che ruotano lontano nella Kuiper Belt o nella Nube di Oort. A questo gruppo di oggetti vanno aggiunti i piccoli satelliti dei pianeti giganti, forse catturati, e la polvere interplanetaria.



Numerose sono le domande che la comunità scientifica si pone riguardo a questi corpi. Ci si domanda, per esempio, quale sia il legame tra l'evoluzione della fascia asteroidale e quella dei pianeti terrestri e quando abbia avuto luogo tale evoluzione; quale sia la distribuzione degli oggetti nelle zone esterne del sistema planetario e quale la loro relazione con le comete di corto e lungo periodo e se esista una differenza di composizione chimica tra queste due popolazioni.

Attualmente 2 missioni riguardano questa area di interesse, la missione “*Rosetta*”, in corso, e la missione “*Dawn*”, in avanzata fase di preparazione.

**Rosetta:** è una missione ESA lanciata nel marzo del 2004. È destinata ad incontrare, nel 2014 la cometa 67/P Churyumov-Gerasimenko, al termine di un viaggio che prevede 3 fly-by della Terra ed uno di Marte. Tra gli obiettivi scientifici della missione ci sono la determinazione della morfologia e della composizione del nucleo, lo studio della composizione e delle proprietà fisiche dei materiali volatili e lo studio dell'evoluzione dell'attività cometaria. La sonda porta anche con se un lander che avrà il non facile compito di posarsi sulla superficie del nucleo per una serie di analisi in situ.

Tra gli strumenti presenti a bordo della missione vanno citati gli italiani *VIRTIS*, spettrometro ad immagini per lo studio spettroscopico del nucleo e delle emissioni, e *GIADA*, per la raccolta e lo studio delle proprietà delle particelle provenienti dal nucleo della cometa.

**Dawn:** si tratta di una missione NASA-ASI la quale ha come obiettivo quello di raggiungere gli asteroidi Vesta, da ottobre 2011 fino a febbraio o maggio 2012, e Cerere, da agosto o settembre 2015 fino a gennaio 2016. Gli obiettivi scientifici sono la raccolta di informazioni sulle condizioni che regnavano durante le fasi iniziali dell'evoluzione del Sistema Solare e la caratterizzazione dei “mattoni” a partire dai quali si sono formati i pianeti terrestri. Poiché lo scopo principale di Dawn è studiare il ruolo delle dimensioni e della presenza di acqua nel determinare l'evoluzione di un pianeta, è stato scelto di osservare da vicino due dei proto-pianeti più grandi,

rimasti quasi intatti dalla loro formazione ma che presentano caratteristiche differenti a causa dei processi molto diversi che cui sono andati incontro. Da notare, in questa missione che è formalmente italo-americana, la presenza di *VIR*, una versione semplificata dello spettroscopio *VIRTIS* a bordo di “*Rosetta*”.

## **1.4) Archiviazione dei dati**

Uno dei fattori cruciali nel determinare il successo scientifico di una missione di esplorazione del sistema planetario è la capacità di estrarre valore e significato scientifico dalla mole spesso enorme di dati. Tale capacità dipende in modo sostanziale dalla possibilità di organizzare e finanziare in modo adeguato i programmi dedicati alla archiviazione ed alla analisi dei dati, possibilmente fin dalla concezione della missione stessa.

Un esempio di quanto un corretto approccio alla trattazione dei dati sia vitale per il successo di una missione può venire dalla missione NASA “*NEAR*”, per la quale era stato espressamente studiato e finanziato un programma separato di archiviazione ed analisi dati e che ha portato alla elaborazione di dati in misura ben superiore a quanto inizialmente stimato, in tempi molto brevi rispetto ai tempi di svolgimento della missione e con una grossa ricaduta anche in termini del ritorno d'immagine nei riguardi della opinione pubblica, anche non specialista del campo. Altre missioni del passato, come il “*Lunar Prospector*”, o le missioni di dimostrazione tecnologica, quali ad esempio “*Clementine*” e “*Deep Space 1*”, dimostrano invece il caso opposto: per tali missioni infatti non era stato previsto né finanziato un programma separato di archiviazione ed analisi dati, ed in conseguenza di ciò i dati provenienti da queste missioni sono stati scarsamente studiati ed interpretati.

Ogni missione spaziale produce una quantità enorme di dati. Il termine “dato” in questo contesto indica non solo l'informazione puramente scientifica (un'immagine multi-spettrale, una mappa di temperatura o altro), ma anche tutte le informazioni collaterali che

permettono l'interpretazione corretta e il pieno utilizzo del dato scientifico, ad esempio i dati di calibrazione, le informazioni sull'assetto della sonda o quelle sullo stato dello strumento.

Accade abbastanza spesso che dei dati siano re-interpretati ex-novo a distanza di anni, come è accaduto per le immagini della cometa Halley prese da "*Giotto*" o quelle di Mercurio riprese da "*Mariner 10*". L'informazione contenuta nei dati deve essere quindi non solo archiviata in modo opportuno per garantire una corretta e sicura conservazione, ma deve anche essere organizzata in modo da essere facilmente reperibile, consultabile, utilizzabile e confrontabile con altri dati.

Tutto questo pone enormi problemi di diversi tipi. Un primo aspetto di fondamentale importanza riguarda la mole di dati che vengono generati dai singoli strumenti. Valga per tutti l'esempio della missione Cassini: a partire dal luglio 2004 è iniziato il tour del sistema di Saturno che ha una durata nominale di 4 anni nel corso dei quali sono pianificate osservazioni in modo abbastanza continuativo. Per dare un'idea della quantità di dati generati da questa missione basti pensare che durante l'incontro di "*Cassini*" con Giove il solo strumento *VIMS* ha generato una quantità di dati pari a tutta la missione "*Galileo*".

Per capire meglio quanto questo aspetto sia di fondamentale importanza è necessario analizzare il problema anche da un'altra prospettiva. Occorre, infatti, tenere presente che ogni missione ospita a bordo diversi strumenti, di natura differente, volti ad analizzare, da punti di vista diversi, lo stesso target, in maniera da averne un quadro il più possibile completo e dettagliato.

Per ognuno degli strumenti è necessario un lungo lavoro a terra, prima che lo stesso sia alloggiato a bordo della navicella, al fine di ottimizzarne le prestazioni e di conoscerne nella maniera migliore il funzionamento e gli eventuali problemi che influenzano i dati da esso prodotti.

Oltre a ciò, anche durante la fase iniziale del volo, prima, cioè, che lo spacecraft giunga a destinazione, nella fase denominata

*commissioning*, si effettuano numerose acquisizioni di dati allo scopo di verificare se ognuno degli strumenti a bordo opera nella maniera corretta e di rilevare, se presenti, eventuali anomalie nel funzionamento di ciascuno.

Una volta completata la suddetta fase, durante il tragitto, gli strumenti non rimangono mai completamente inattivi fino all'obiettivo prefissato. Al fine di ridurre la quantità di carburante da caricare sulle navicelle, con conseguente notevole aumento di peso e di difficoltà per portarle in orbita, i percorsi per raggiungere il target prevedono sempre dei passaggi ravvicinati ad uno o più dei pianeti del Sistema Solare. Tali passaggi sono effettuati con lo scopo di sfruttare l'*effetto fionda* per "rubare" un po' del momento dei pianeti e ricavarne un'ulteriore spinta da conferire alla sonda nella giusta direzione. È facile capire che ogni volta che un'opportunità del genere si presenta è interesse di tutti attivare ogni strumento per effettuare delle acquisizioni da utilizzare sia come ulteriore verifica dello stato di salute dello strumento stesso, sia a fini di pura ricerca.

Oltre alle occasioni appena citate, nel corso del viaggio, possono presentarsi altre opportunità di interesse costituite da qualcuno dei corpi minori che, come detto in precedenza, costituiscono motivo di forte interesse per la comunità dei planetologi. È quindi naturale che anche in occasione di tali eventi tutte le possibili osservazioni siano eseguite.

Una volta raggiunta la destinazione, infine, inizia la fase più importante della missione, l'osservazione del *target* principale, per il quale la sonda è stata pensata e realizzata. Tale fase può avere una durata di qualche mese, come capita, ad esempio, per la missione "*Rosetta*", ma può anche durare anni, come nel caso della missione "*Cassini*". Alla durata nominale della missione può poi, generalmente, aggiungersi quella che viene definita "missione estesa", la cui durata può eguagliare quella della missione nominale. Per ogni strumento, durante tale periodo, sono pianificate osservazioni pressoché continuative, al fine di massimizzare il ritorno scientifico della missione e fornire ai ricercatori il maggior quantitativo possibile di informazioni.

Quanto detto fino ad ora dovrebbe rendere chiaro quanto sia rilevante la questione della mole dei dati che si accumulano nel corso della durata di una missione. Gestire questa grande quantità di dati significa, in ultima analisi, tenere sotto controllo il notevole numero di file nei quali i dati sono immagazzinati, siano essi osservazioni vere e proprie, dati di housekeeping, informazioni sulla geometria delle osservazioni o pacchetti di telemetria con le informazioni sul funzionamento dello strumento.

Ogni ricercatore che si trovi a voler lavorare su un argomento specifico, nel momento in cui si appresta ad organizzare il proprio lavoro, può trovarsi spiazzato di fronte all'imponente numero di file tra i quali ricercare le informazioni che gli sono necessarie. È necessario, inoltre, tener conto del fatto che, anche dopo aver discriminato i file con le osservazioni di interesse da resto, è molto probabile che il ricercatore si trovi comunque di fronte ad un non trascurabile numero di acquisizioni riguardanti lo stesso particolare ma effettuate in condizioni diverse e, quindi, non necessariamente tutte adatte al tipo di analisi che si vuole portare avanti.

Questo discorso, così come portato avanti fino ad ora, è relativo alla situazione che si prospetta ad un ricercatore che voglia occuparsi di studiare i dati, relativi ad un target ben preciso, e provenienti da **un solo** strumento. Tale ottica, però, è fortemente riduttiva in quanto trascura uno dei punti principali già discussi e che è importante sottolineare perché di fondamentale importanza: ogni sonda contiene un certo numero di strumenti diversi, ognuno in grado di osservare determinate caratteristiche, affinché tutti insieme forniscano un quadro il più possibile completo della natura e del comportamento del target. Trascurare questo aspetto può costituire una grave limitazione nel lavoro di ricerca. Informazioni derivanti da analisi di tipo differente possono rappresentare i pezzi di un puzzle che messi insieme sono in grado di fornire un quadro completo e preciso di ciò che si sta osservando.

È facile capire, a questo punto, che non è possibile pensare che ogni singolo membro della comunità scientifica debba mettere in

pieci e gestire un archivio, dalle notevoli dimensioni in termini di spazio su disco, al quale attingere per il proprio lavoro.

Alla luce di quanto detto risulta chiara l'importanza di un sistema di archiviazione dei dati che possa essere sfruttato quale strumento per trovare tutte le informazioni necessarie per il proprio lavoro senza avere la necessità di dover usare una considerevole parte del proprio tempo per eseguire in proprio una tale attività.

Un simile sistema deve avere alcune caratteristiche ben precise per poter essere veramente utile alla comunità scientifica:

- essere facilmente accessibile;
- essere facilmente utilizzabile;
- consentire ricerche relativamente veloci;
- consentire ricerche mirate, basate sulle caratteristiche fondamentali dello strumento i cui dati si stanno ricercando;
- fornire una risposta in grado di permettere all'utente di stabilire quali e quanti dei dati ritrovati dal sistema al termine della ricerca siano effettivamente interessanti;
- consentire, al termine della suddetta ricerca, di procurarsi i dati ritenuti di interesse al fine di poterli analizzare in un secondo momento.

Non è certo difficile, date le caratteristiche appena descritte, comprendere come la rete internet sia il "luogo" migliore per creare un tale sistema di archiviazione e ricerca dati. Se fino a qualche anno fa, infatti, la scarsa ampiezza di banda delle connessioni e la scarsa diffusione dei collegamenti al *web* potevano costituire un valido impedimento ad un progetto del genere, al giorno d'oggi tali problemi sono senz'ombra di dubbio superati e, anzi, la facilità di comunicazione da un capo all'altro del mondo e la rapidità degli scambi di dati offerte dalle connessioni a banda larga costituiscono le colonne portanti della ricerca, intesa, oggi come non mai, come

collaborazione tra persone ed enti situati in posizioni geograficamente anche molto lontane.

In un'ottica quale quella appena descritta la comunità dei planetologi italiani ha ritenuto opportuno che in Italia fossero sviluppate le conoscenze e le strutture necessarie ad archiviare e gestire anche in proprio e in loco i dati provenienti da missioni che la vedono pesantemente coinvolta e che richiedono anni di lavoro.

Per realizzare un tale proposito, evidentemente, è stato necessario un notevole sforzo progettuale e organizzativo. Uno dei requisiti fondamentali chiari fin dall'inizio, infatti, era che al momento della progettazione e della creazione di un tale archivio, lo schema di funzionamento generale e, con esso, le caratteristiche tecniche e di gestione, doveva essere concepito in maniera tale da non essere condizionato in maniera eccessiva dalle peculiarità di un qualche strumento (ad esempio, il primo per il quale l'archivio sarebbe stato operativo) ma doveva essere tale da potersi adattare a classi di strumenti differenti. Ciò al fine di poter dare vita ad uno strumento utile alla più ampia platea possibile e non solo ad un ristretto gruppo.





## Capitolo 2

### Lo standard PDS

#### 2.1) Cos'è il PDS

##### 2.1.1) La struttura

Il *Planetary Data System* è un sistema ideato e realizzato allo scopo di fornire un archivio, basato su uno standard unico, per la ricerca di tutti i dati relativi all'esplorazione del Sistema Solare.

Tale sistema, sviluppato negli Stati Uniti dalla NASA, è articolato in una serie di siti, dedicati ognuno ad una branca particolare dell'esplorazione planetaria, detti *discipline nodes*. In ognuno di essi sono fisicamente conservati i dati relativi alla disciplina nella quale il ciascun nodo è specializzato. Essi sono la struttura portante del PDS e sono strutture permanenti. Per ogni *discipline node* possono esistere delle sotto strutture, chiamate *data node*, le quali hanno compito di supporto nell'attività di raccolta, documentazione e archiviazione dei dati. I *data node* possono, al contrario dei *discipline node*, avere un'esistenza limitata nel tempo e venire disabilitati una volta che hanno esaurito un compito specifico, quale può essere quello di gestire un determinato set di dati fino al termine del processo di archiviazione.

Attualmente i *discipline node* esistenti sono i seguenti:

- *Atmospheres*, per i dati sulle atmosfere planetarie;
- *Geosciences*, per i dati sulle superficie e gli strati interni dei pianeti;
- *Planetary Plasma Interactions*, per i dati sulle magnetosfere e le interazioni di queste ultime con il plasma dello spazio interplanetario;
- *Rings*, per i dati sugli anelli dei pianeti;
- *Small Bodies*, per i dati su comete, asteroidi e polveri interplanetarie;
- *Imaging*, specializzato in immagini digitali ed elaborazioni;
- *Navigation and Ancillary Information Facility*, specializzato nella generazione delle informazioni geometriche relative alle osservazioni.

Ogni *discipline node* è responsabile, inoltre, del ripristino e dell'archiviazione dei vecchi dati, oltre che del processo di archiviazione di quelli riguardanti le nuove missioni.

Esiste poi un altro nodo, chiamato *Engineering Node*, il quale è a capo di tutta la struttura. Esso, inoltre, provvede a gestire e migliorare lo standard comune a tutti i nodi, quali la documentazione e la formattazione dei dati.

Tutti i nodi, sia i *discipline* che l'*Engineering*, oltre alle attività fin qui descritte, sono impegnati sia nell'opera di rendere tutti i dati accessibili alla comunità scientifica, sia in un attivo e costante lavoro di scambio di informazioni con i responsabili delle missioni in corso affinché tutti i dati che saranno poi archiviati abbiano il necessario corredo di informazioni ausiliarie e la necessaria documentazione.

## 2.1.2) La filosofia del PDS

La filosofia che anima la struttura del PDS, che ha ispirato la creazione dello standard e che ne costituisce il punto di forza è quella di guardare alle generazioni future: la ricerca di un sistema che permetta a chi farà ricerca fra 10 o 20 anni di poter avere pieno accesso non solo ai dati, ma a tutto l'insieme di informazioni ausiliarie e geometriche che permettono di utilizzare pienamente i dati per le proprie analisi e senza le quali una corretta interpretazione degli stessi sarebbe impossibile. Se si vuole che questo accada non basta limitarsi, quindi, a “salvare i bit” da qualche parte e mettere tutto da parte lasciando a chi verrà il compito di riordinare l'insieme delle informazioni, è necessario creare uno standard che si basi su una struttura definita ed organizzata nella quale siano presenti tutti gli elementi appena menzionati. Proprio questa è stata l'idea di partenza che ha portato alla definizione dello standard PDS.

Ogni archivio, per essere definito corretto e ben formato, deve possedere i seguenti dati:

- **dati “raw”**: con ciò si intende il flusso di informazioni giunte con la telemetria o i file derivati da una prima elaborazione che sia servita esclusivamente a rimuovere le anomalie note dello strumento o derivanti da fattori meccanici. È necessario che l'attuazione di tali misure non alteri in alcun modo i dati generando controversie o incertezze in una fase successiva;
- **dati finali**: sono i dati ottenuti al termine di tutto il processo di elaborazione durante il quale tali dati sono stati sottoposti a diversi stadi di riduzione e interpretazione.
- **Dati di calibrazione**: comprende tutto l'insieme di dati quali *dark current*, *flat field* o qualsiasi altro tipo di dato usato per ottenere i dati finali.

- **Informazioni ausiliarie:** informazioni geometriche, effemeridi e tutto ciò che è stato usato nelle fasi di riduzione e interpretazione dei dati.
- **Specifiche dello strumento:** caratteristiche dello strumento, senza queste informazioni ogni interpretazione è inutile.
- **Documentazione:** documenti di testo che descrivono i processi di riduzione e interpretazione dei dati, informazioni per l'interpretazione del contenuto dei file.

Un punto che va particolarmente sottolineato all'interno di un simile discorso riguarda lo sforzo che è stato fatto affinché al termine del processo di elaborazione che porta alla scrittura e all'archiviazione dei dati nella loro forma finale, sia i dati veri e propri che le informazioni ausiliarie sono contenute in file formattati in maniera tale da essere facilmente letti da *tool* che possono essere appositamente generati.

A tal riguardo è necessario specificare che nonostante il PDS sia uno standard, ed in quanto tale abbia delle regole ben definite per la creazione dei file di dati, esso deve essere abbastanza versatile da potersi adattare a descrivere dati provenienti da strumenti con caratteristiche anche molto differenti e destinati ad osservare *target* di natura molto differente: basti pensare a titolo di esempio alla differenza esistente tra uno spettroscopio ad immagini ed un radar. Ben poco hanno in comune due strumenti del genere eppure i dati di entrambi devono poter essere catalogati ed archiviati secondo il medesimo standard.

È facile comprendere come sia praticamente impossibile creare dei *tool* che siano in grado di aprire indistintamente file provenienti da uno qualsiasi dei possibili strumenti utilizzati nelle missioni. C'è di più: anche restringendo il campo ad una sola categoria di strumenti ci si accorge, in realtà, ben presto che risulta comunque molto difficile, anche se non impossibile, realizzare un software che permetta di effettuare l'analisi dei dati acquisiti con tutti gli strumenti appartenenti alla classe selezionata. A titolo di esempio,

allo scopo di spiegare meglio questo tipo di difficoltà, si vuole qui citare l'esempio costituito da due spettrometri ad immagine, *VIMS* di "Cassini" e *VIRTIS* di "Rosetta", strumenti sui dati dei quali si è avuto modo di lavorare durante il periodo nel corso del quale si è svolta l'attività di ricerca di dottorato. *VIRTIS* è uno strumento nato sulla scorta dell'esperienza acquisita grazie a *VIMS* e, in quanto evoluzione risulta essere più complesso sia sotto l'aspetto ingegneristico sia dal punto di vista della complessità dei file di dati prodotti. Tale diversità ha portato all'esigenza di avere due software che contemplassero due modalità di lettura dei file significativamente differenti.

Per concludere questo discorso va, infine, ribadito con forza che, nonostante ciò che è stato fin qui detto riguardo allo sforzo di rendere i dati facilmente leggibili da *tool* specializzati, lo scopo primario dello standard PDS, quello che ne costituisce al di là di ogni altra considerazione la ragion d'essere, è rendere i dati usufruibili da chi farà ricerca in futuro. Ciò significa che, più di ogni altro, il traguardo ultimo è fare in modo che i dati generati possano essere compresi anche da chi un domani vorrà accedere alle informazioni registrate negli archivi pur non facendo direttamente parte del team dal quale tali informazioni sono state generate.

In un tale contesto la leggibilità dei dati da parte di appositi software deve essere vista come un *plus valore* che viene attribuito ai dati stessi al termine di un lungo processo di elaborazione.

## 2.2) Struttura di un archivio PDS

### 2.2.1) Struttura generale

Un archivio PDS è costituito da un insieme di file, organizzati in una struttura gerarchica ben precisa, nel quale ogni livello fornisce informazioni su una scala più ampia rispetto al livello sottostante. Di seguito si riporta una descrizione molto generale dei suddetti livelli:

- **Data file:** è il livello più basso, quello dei dati veri e propri. Ad ogni file di dati deve corrispondere una *label* che lo descriva, sia essa *attached*, cioè facente parte del file di dati, sia, invece, *detached*, cioè, posta in un file separato.
- **Meta-data file:** è il livello intermedio. Di questo livello fanno parte i file che descrivono i parametri e le condizioni al contorno dell'osservazione: ad esempio, lo strumento, il sito di osservazione, il tipo di dati raccolti. Sono quelli che vengono chiamati *catalog file*.
- **Volume description file:** sono i file contenuti in ciò che è definito un "volume", sia esso un file compresso, un DVD o altro. Descrivono in maniera dettagliata l'organizzazione dei file del volume stesso: file di dati, documentazione, eventuale software, ecc.

Un altro concetto fondamentale nell'ambito dello standard PDS è quello di *data set*. Tale entità descrive tutto l'insieme di cartelle e file necessario per comprendere ed utilizzare i dati. Tali dati sono tutti quelli relativi ad una certa missione, ad una fase della missione stessa, ad un certo strumento, ecc. Un *data set* può essere interamente compreso all'interno di un volume, oppure può essere suddiviso in più volumi.

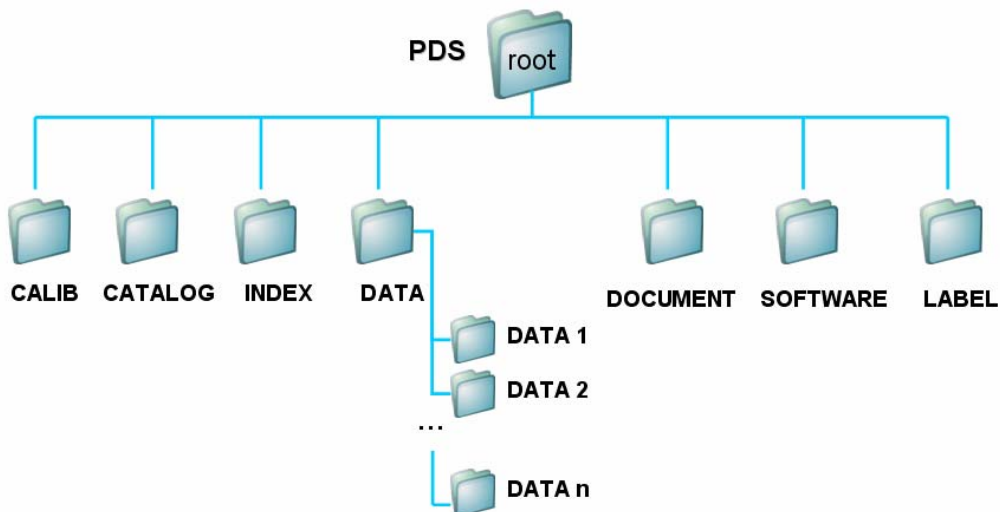


Figura 3. 1 Struttura di un archivio PDS che contiene tutte le cartelle obbligatorie e le principali opzionali.

### 2.2.2) Volume description file

Un volume è un'entità definita come un insieme auto-consistente di file che deve necessariamente contenere alcuni elementi, fondamentali per la comprensione dei dati, mentre altri sono opzionali. La relazione tra tale entità ed i supporti fisici può essere di tre tipi: nel caso in cui il volume è interamente contenuto in un unico supporto fisico si ha una relazione di tipo 1:1; se il volume, invece, deve essere suddiviso su più supporti si ha una relazione di tipo 1:N; se, infine, si ha il caso contrario, cioè è un unico supporto ad ospitare più volumi, si ha una relazione del tipo N:1.

Come accennato in precedenza i file descrittivi del volume sono quelli definiti come di livello più alto. Essi descrivono tutto ciò che è presente nel volume: dati, label, software, documentazione, ecc.

All'interno di ogni volume alcune cartelle ed alcuni file devono obbligatoriamente essere presenti affinché il volume sia ben formato. Tutti i file possono essere raggruppati in quattro categorie:

- “TXT”: sono file di testo o ASCII;
- “LBL”: sono file label, contenenti la descrizione di un oggetto PDS;
- “CAT”: sono file scritti utilizzando il cosiddetto “*Object Data Language*” (ODL), che sarà descritto più avanti, e contengono un catalogo degli oggetti PDS;
- “TAB”: sono file tabella.

L'elenco completo di cartelle e file obbligatori comprende i seguenti elementi:

- **Root Directory**
  - AAREADME.TXT: un'introduzione generale sul volume;
  - VOLDESC.CAT: un file ODL contenente un oggetto “VOLUME” che descrive il volume.
- **INDEX sub-directory**: lo standard PDS richiede una presenza minima di indici relativi alle cartelle ed ai file presenti. Ogni altro file di indici utile per l'utilizzo dei dati deve essere incluso in questa cartella.
  - INDEXINFO.TXT: descrizione generale del contenuto della cartella INDEX;
  - CUMINDEX.TAB, CUMINDEX.LBL: se il data set è costituito da più di un volume è necessaria la presenza di un indice cumulativo. Questi due file sono, rispettivamente, la tabella contenente il suddetto indice cumulativo e la label che lo descrive.



- INDEX.TAB, INDEX.LBL: contengono, rispettivamente, la tabella nella quale sono riportati tutti gli oggetti contenuti nel volume e la label che la descrive.
- **CATALOG sub-directory**: contiene i file catalogo che descrivono tutti gli elementi principali per la comprensione della missione e dei dati da essa prodotti: la missione, gli strumenti, i target, il personale del team, ecc.
  - CATINFO.TXT: una descrizione del contenuto della cartella CATALOG;
  - CATALOG.CAT: contiene un oggetto CATALOG il quale, a sua volta, raccoglie in sé tutti gli oggetti CATALOG prodotti per documentare i dati (MISSION, INSTRUMENT, DATA\_SET\_COLLECTION, ecc).
- **DOCUMENT sub-directory**: contiene la documentazione, in formato testo, che descrive ogni aspetto della missione: la sonda, gli strumenti, le tecniche di osservazione, le procedure di riduzione ed interpretazione, ecc.

Esistono poi delle directory che non devono essere obbligatoriamente presenti all'interno di un *data set* ma che, se presenti, contengono informazioni o software molto utili per lavorare con i dati. Tali cartelle sono le seguenti:

- **SOFTWARE sub-directory**: contiene il software, eventualmente rilasciato dal team che si occupa di produrre i dati, che può essere utilizzato per eseguire una più o meno approfondita analisi dei dati.
- **CALIB sub-directory**: contiene tutto ciò che è stato usato dal team responsabile per la produzione dei dati per effettuare la calibrazione dei dati: ciò significa dati e procedure seguite.

- **GEOMETRY sub-directory**: qui sono contenute tutte le informazioni relative alla geometria dell'osservazione: posizione rispetto al target, sistema di riferimento utilizzato per l'interpretazione della posizione stessa, angolo di fase, distanza dal target, ecc.

### 2.2.3) Meta-data file

I file *catalog*, come accennato in precedenza, sono file scritti seguendo le regole dell'*Object Data Language*, che verrà spiegato più avanti, e sono utilizzati per fornire informazioni riguardanti il formato, l'origine, la locazione corrente e il tipo di dati archiviati.

Di seguito viene riportata una lista contenente tutti gli oggetti *catalog* che sono richiesti per soddisfare le esigenze di completezza di informazioni di un *data set*.

- **File Data Set Catalog** : deve essere presente uno di questi oggetti per ogni valore associato alla keyword *DATA\_SET\_ID* nei file del livello sottostante, quello dei dati veri e propri. Oltre a descrivere il contenuto generale del *data set* questo oggetto include informazioni riguardanti la durata della missione, i programmi di osservazione condotti e la persona o il gruppo responsabile della produzione dei dati.
- **File Data Set Collection Catalog** : occasionalmente può risultare utile raccogliere tutti i dati contenuti in alcuni *data set*, in quanto in relazione tra loro per un qualche motivo, in un'unica collezione. La *collection* si presenta, all'interno dello standard PDS, come un'unica entità la quale, in talune occasioni, può risultare di grande utilità ai ricercatori che si trovino nella condizione di dover recuperare tutti i dati contenuti nella *collection* stessa. Oltre alla descrizione del contenuto della *collection*, questo oggetto contiene informazioni riguardanti il

periodo temporale durante il quale sono state eseguite le osservazioni che hanno generato i dati, puntatori ai file che descrivono lo stato dello strumento che ha effettuato le suddette acquisizioni e la descrizione sia della missione che dei programmi di osservazione condotti.

- **File Instrument Catalog** : fornisce una descrizione delle caratteristiche dello strumento utilizzato per l'acquisizione dei dati e dei riferimenti alla letteratura scientifica in proposito.
- **File Instrument Host Catalog** : è richiesto obbligatoriamente all'interno del primo *data set* costituito con i dati di un determinato strumento. Descrive la sonda che ospita lo strumento.
- **File Mission Catalog** : descrive gli obiettivi scientifici della missione, i programmi di osservazione stabiliti ed identifica le persone chiave e le istituzioni coinvolte nella missione stessa.

## 2.2.4) Data file

### Caratteristiche generali

Tutti i dati veri e propri sono contenuti all'interno della cartella DATA, la quale, a sua volta, può contenere delle sottocartelle nelle quali sono suddivisi i dati in base ad un qualche criterio (ad esempio, dati osservati in differenti bande spettrali ma acquisiti contemporaneamente).

Lo standard PDS impone ben poche severe restrizioni all'organizzazione fisica e logica dei file di dati, sebbene risulti chiaro, per chi è solito lavorare in questo ambito, come taluni formati siano più facilmente gestibili di altri.

I file possono essere scritti sia in formato testo che binario, possono essere del tipo *fixed record length* o *variable record length*; l'ordine di scrittura dei bit, il cosiddetto *endianness*, può essere uno qualsiasi dei possibili. Ciò significa che, in teoria, ogni possibile tipo di struttura dei *record*, le unità base di dati dai quali sono formati i file, può essere utilizzato.

Tuttavia, come è stato più volte sottolineato in vari punti di questo capitolo, lo scopo primario dello standard PDS è quello di rendere i dati utilizzabili dalla comunità scientifica ora ed in futuro. Ciò si traduce, nella pratica, con una spiccata tendenza ad utilizzare un formato in particolare piuttosto che altri. Le caratteristiche di tale formato sono le seguenti:

- Record di tipo *fixed length*;
- Un record logico per ogni record fisico;
- Standard di archiviazione commerciali (ASCII per i file di testo, IEEE per i file binari).

Anche per ciò che riguarda i file, che, come già accennato, possono essere scritti sia in formato testo che in formato binario, esistono delle convenzioni:

**File di testo:** qualunque sia il loro contenuto, sia esso una tabella, una label o documentazione, l'unica limitazione è data dall'obbligo di terminare ogni record con la coppia di caratteri *carriage return* (carattere 13 ASCII) e *line feed* (carattere 10 ASCII). Ciò al fine di assicurare che tali file siano leggibili dalla maggior parte dei sistemi operativi, nel peggiore dei casi con il trascurabile inconveniente di visualizzare qualche carattere bizzarro su sistemi, quali Unix, che non necessitano di tale coppia.

È consuetudine, come riportato sopra, utilizzare il formato *fixed length record* anche per questo tipo di file, soprattutto nel caso di tabelle e file label. Nel caso, invece, di file di documentazione non viene richiesto di aderire a tale standard, essendo più pratico scrivere documenti nel formato a *variable record length* oltre che per il fatto

di essere, tali documenti, destinati alla lettura umana e non da parte di software specializzati.

**File binari:** sono generalmente scritti nel formato citato nella tabellina riportata sopra in quanto di più facile gestione. Pur non essendo questo un obbligo l'uso del formato *variable* viene fortemente scoraggiato. Anche per ciò che riguarda il formato di sistema di scrittura dei bit non esistono vincoli, viene però richiesta una documentazione che permetta di comprendere il formato dei dati.

### **File dati e label - introduzione**

I dati con le loro label costituiscono il cuore di un archivio PDS. Chiunque si trovi a fare ricerca e necessiti di dover elaborare una serie di dati non necessita solo dei dati veri e propri ma anche di tutta quella serie di informazioni accessorie che descrivono il formato ed il contenuto dei file di dati.

Nello standard PDS ogni file di dati è sempre accompagnato da una corrispondente label, sia essa compresa nello stesso file che contiene i dati, e detta, in questo caso, *attached*, sia essa, invece, contenuta in un file separato, e detta *detached*.

Una label è composta da una serie di campi che si susseguono a formare una descrizione completa e dettagliata di tutto ciò che riguarda i dati ai quali essa è associata. In una label sono, infatti, presenti informazioni riguardanti lo stato di funzionamento dello strumento, gli istanti di inizio e fine acquisizione, il target osservato, la geometria dell'osservazione, la struttura dei dati all'interno del file. Una caratteristica fondamentale delle label è che, essendo scritte in *ODL*, sono facilmente comprensibili dagli esseri umani e, al tempo stesso, sono leggibili da parte di *tool* specializzati.

## Struttura di una label

Anche se non esistono, come già ribadito in precedenza, delle regole rigide all'interno dello standard PDS, chi si occupa di creare i *data set* è incoraggiato a seguire determinati modi di procedere piuttosto che altri. Ciò vale anche per la formazione della label che deve descrivere il contenuto di un determinato file di dati. Le keyword che la compongono possono essere raggruppate in insiemi i cui appartenenti descrivono un particolare aspetto dei dati:

- **standard identifiers:** indicano quale versione dello standard PDS è stata usata durante la creazione del *data set*;
- **file characteristics:** descrivono le caratteristiche del file contenente i dati, quali tipo di record utilizzato (*fixed* o *variable*), lunghezza del singolo record, numero di record che costituiscono il file, lunghezza della label e lunghezza della parte con i dati;
- **data pointers:** sono i puntatori ai dati veri e propri. Nel caso di *attached* label la parte binaria con i dati comincia al termine della label. se si ha una *detached* label i dati si trovano in un altro file indicato in questa sezione;
- **identification parameters:** è l'insieme di tutte quelle informazioni che permettono di mettere in relazione il file di dati con altri simili o collegati e con la descrizione nei file *catalog*. Si tratta di parametri quali il target osservato, la modalità di acquisizione dei dati, lo stato di funzionamento dello strumento, la missione e la fase della missione stessa durante la quale l'acquisizione è avvenuta, il *data set* di appartenenza ed altre ancora;
- **observational parameters:** è l'insieme delle informazioni che descrivono l'osservazione, in particolare i parametri geometrici che definiscono la posizione della sonda rispetto al *target*, l'angolo di fase tra sonda, *target*

e Sole, ad esempio, il sistema di coordinate utilizzato (dipendente dal tipo di *target*), ecc.

- **data structure definition:** comprende tutte le informazioni necessarie a capire come sono stati registrati i dati all'interno del file, il tipo di dato (intero, a virgola mobile, ecc), l'esistenza di eventuali informazioni ausiliarie accorpate insieme ai dati veri e propri, l'organizzazione dei dati stessi (vettori, matrici N-dimensionali, ecc).

### Definizione della struttura dei dati

La descrizione della struttura logica e fisica dei dati presenti nel file è riportata nelle label attraverso l'uso di alcuni oggetti, definiti dallo standard, delimitati dalle *keyword OBJECT, END\_OBJECT*. Tali oggetti possono anche essere nidificati, se necessario.

Allo scopo di promuovere una certa uniformità nelle descrizioni dei dati e per rendere possibile la creazione di *tool* che abbiano un minimo di valenza generale, pur con tutte le difficoltà di cui si è parlato nella parte iniziale di questo capitolo, lo standard PDS prevede una serie di oggetti già definiti e che, almeno nelle intenzioni, dovrebbero essere sufficienti a coprire tutti i possibili casi possibili. Proprio per raggiungere lo scopo di non rendere necessaria la definizione di nuove entità, infatti, tutti i suddetti oggetti sono stati sviluppati in maniera da essere abbastanza flessibili e generici.

Ogni oggetto definito nello standard PDS contiene un numero minimo di *keyword* che ne definiscono le proprietà sia fisiche, quali lunghezza o collocazione dei byte, che logiche, quali unità di misura o nomi dei campi contenuti.

## ODL – Object Data Language

È un tipo di linguaggio, estremamente semplice, sviluppato dalla NASA al solo scopo di utilizzarlo per descrivere gli oggetti e le strutture presenti in un archivio PDS.

Le unità base di questo linguaggio, cioè, le righe hanno la seguente sintassi:

*<keyword > = <value>*

Un elenco completo di tutte le *keyword* definite e disponibili per essere utilizzate in un archivio è presente in un apposito dizionario, il *Planetary Science Data Dictionary* (PSDD), disponibile on-line. Al fine di mantenere la più volte citata versatilità dello standard e per consentire allo standard stesso di mantenersi aggiornato seguendo l'evoluzione tecnica degli strumenti esiste un protocollo di aggiornamento delle *keyword* e dei valori per esse consentiti grazie al quale è possibile far fronte alle esigenze che si presentano quando si ha a che fare con nuove missioni che hanno incorporate soluzioni tecniche o modi operativi non previsti in precedenza dallo standard.

Una delle caratteristiche che le *keyword* dovrebbero avere, anche se, come ampiamente sottolineato, ciò non è obbligatorio, è quello di essere descrittive anche se lunghe, al fine di migliorare la leggibilità delle label.

Un insieme coerente di *keyword* può essere raggruppato in un oggetto, entità già citata più volte. Gli oggetti sono una parte particolarmente importante all'interno dello standard PDS, in quanto consentono di descrivere degli oggetti fisici trattandoli, nell'ambito di una label, come delle entità logiche a sé stanti, in una qualche analogia con gli oggetti così come sono concepiti e trattati nella programmazione orientata agli oggetti, appunto. Ogni oggetto, infatti, altro non è se non una descrizione di una struttura, più o meno complessa, di dati, la quale contiene una descrizione sia logica che fisica dei dati.



Gli oggetti definiti dal PDS devono essere utilizzati all'interno dei file *label* e *catalog* e sono, grazie alla loro genericità, in grado di descrivere un'ampia gamma di dati. Sta poi a chi è deputato ad utilizzarli per creare gli archivi saperli utilizzare, mediante l'impiego delle *keyword* e dei loro valori, per descrivere correttamente i caratteri più specifici dei dati stessi.

La struttura generale di un oggetto è la seguente:

```
OBJECT      = nome_oggetto
            keyword_1 = valore
            ...
            keyword_n = valore
END_OBJECT = nome_oggetto
```

Ogni oggetto può, inoltre, contenere altri oggetti al proprio interno, secondo alcune relazioni di “parentela” definite, relazioni le quali fanno sì che, per esempio, un oggetto TABLE possa contenere un oggetto COLUMN ma non un oggetto IMAGE.

Poiché, come già anticipato, l'utilizzo degli oggetti è riservato alla descrizione del formato logico e fisico dei file di dati, ogni oggetto deve contenere delle *keyword* che descrivano specifiche caratteristiche:

- il tipo di dato e la dimensione in byte del dato stesso;
- la posizione dei campi all'interno del record;
- il significato logico dei campi;
- ogni altra informazione specifica che riguardi l'oggetto e ne agevoli la comprensione.



## Capitolo 3

### Il formato dei dati di VIMS-V

In questo capitolo verrà presentato il formato PDS così come applicato allo strumento *VIMS-V*. Poiché lo scopo del lavoro descritto nella presente tesi è relativo al formato finale dei dati e non riguarda direttamente il funzionamento dello strumento si tralascia qui la descrizione dello strumento e della missione all'interno della quale opera. La conoscenza sia dello svolgimento della missione che del funzionamento, ovvero dei modi operativi e delle caratteristiche tecniche, dello strumento sono, però, utili nella fase preliminare del lavoro, quella di progettazione dell'archivio. Per tale motivo si è pensato, quindi, di inserire in appendice una parte riguardante sia lo svolgimento e le finalità scientifiche della missione *Cassini*, sia una sommaria descrizione tecnica dello strumento *VIMS-V*.

#### 3.1) I “cubi”

##### 3.1.1) L'oggetto QUBE

Un oggetto *cube* consiste in un array tridimensionale nel quale sono contenuti i dati acquisiti durante un'osservazione. Questo tipo di formato è spesso utilizzato per gli spettrometri ad immagine, qual è *VIMS* appunto, e consiste in una successione di *array* bidimensionali, posti in sequenza uno dietro l'altro. Ogni elemento dei suddetti *array* corrisponde ad un pixel della camera CCD che è

utilizzata per eseguire le acquisizioni, per cui le due dimensioni dell'*array* corrispondono, rispettivamente, alle coordinate spettrali, poiché lungo di essa si succedono colonne di pixel sui quali cade la luce di una determinata lunghezza d'onda  $\lambda$ , e alle coordinate spaziali, dato che lungo di essa si succedono righe di pixel sui quali cade la luce proveniente dalla stessa regione spaziale del target. La terza dimensione, lungo la quale sono allineati gli *array* bidimensionali, corrisponde alle coordinate temporali, visto che lungo di essa si susseguono *array* acquisiti in momenti successivi.

Ogni file *qube* è descritto da una *label* che fornisce tutte le informazioni necessarie a comprenderne la struttura, sia fisica che logica, e permette di sviluppare dei software che siano in grado di leggere il contenuto del file stesso e di interpretarne correttamente il contenuto. In particolar modo va tenuto presente che l'oggetto *qube*, oltre a contenere i dati veri e propri, memorizzati in una regione detta *core*, contiene anche informazioni ausiliarie in regioni dette *suffix plane*. Entrambe queste regioni sono descritte nella *label* in maniera dettagliata.

### **3.1.1.1) Il core**

Alcune *keyword*, in particolare, sono significative a tal riguardo. Innanzitutto la *keyword* AXES, che specifica il numero degli assi; AXIS\_NAME che, come si deduce dal nome, fornisce il nome degli assi e, fondamentale, l'ordine seguito durante l'operazione di scrittura dell'*array* nel file; CORE\_ITEM\_TYPE e CORE\_ITEM\_BYTES, che descrivono il tipo di dato usato per scrivere il file; CORE\_NAME e CORE\_UNIT, le quali forniscono il nome e le unità di misura relative ai dati. I nomi utilizzati per i 3 assi di un oggetto *qube*, generalmente, sono: *bands* per l'asse spettrale; *samples* per l'asse spaziale; *lines* per l'asse temporale. Di seguito viene raffigurata la struttura generale del *core* di un oggetto *qube*.

Ovviamente si tratta di una raffigurazione logica, che non può rappresentare ciò che si trova effettivamente in un file ma che

ben rappresenta il modo corretto di visualizzare mentalmente i dati così come sono organizzati nel file.

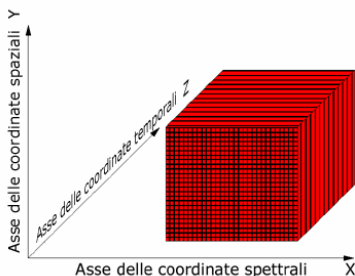


Figura 3. 1 Rappresentazione grafica di un oggetto QUBE.

### 3.1.1.2) I suffix plane

Ognuno dei tre assi in un oggetto *cube* può, eventualmente, includere dei dati ausiliari che estendono la lunghezza degli assi. Concettualmente ciò può essere descritto come un certo numero di regioni che sono disposte ai margini del *core* in altrettante zone ben precise. La regione che estende la dimensione spettrale è chiamata *backplane*, quella che estende la dimensione spaziale è chiamata *sideplane* e, infine, quella che estende la dimensione temporale è detta *bottomplane*. Tale situazione è mostrata in figura 3.2.

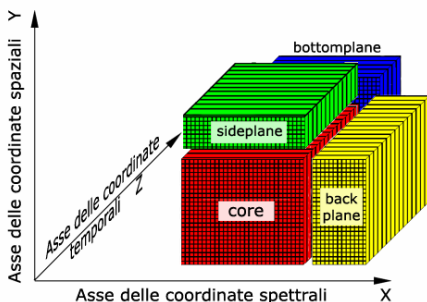


Figura 3. 2 Rappresentazione grafica di un core con i relativi suffix plane.

In maniera del tutto simile a quanto avviene per il *core*, anche i *suffix plane* sono descritti da una serie di *keyword*, in una zona appositamente dedicata della *label*, che comprendono il numero dei piani contenuti lungo ognuno degli assi ed informazioni riguardanti il tipo ed il formato dei dati utilizzati per scrivere tali regioni.

### **3.1.1.3) La scrittura dei dati in un file**

Uno dei problemi che si pongono all'attenzione di chi voglia occuparsi della comprensione del formato PDS è quello relativo alle modalità di scrittura degli oggetti *cube* in un file. Più precisamente la questione fondamentale è la seguente: un file è, da un punto di vista puramente fisico, una struttura monodimensionale, cioè, una semplice sequenza di byte, ci si può, quindi, domandare come disporre, all'interno di tale *array* ad una sola dimensione, una struttura che, invece, ha 3 dimensioni.

Un simile problema si può facilmente risolvere immaginando di muoversi all'interno del *cube* seguendo un percorso tale da avere, come risultato, sia una lettura completa di tutti i dati in esso contenuti, sia una disposizione, all'interno del file, che permetta la ricostruzione, in un secondo momento, della struttura di partenza.

La questione è affrontata, da un punto di vista pratico, considerando il *cube* come una matrice 3-D e decidendo quale indice, dei tre disponibili, far variare più velocemente, quale per secondo e quale più lentamente.

Esistono 3 differenti ordini di lettura-scrittura, definiti nel modo seguente:

- (SAMPLE, LINE, BAND), chiamato BSQ (Band Sequential);
- (SAMPLE, BAND, LINE), chiamato BIL (Band Interleaved by Line);

- (BAND, SAMPLE, LINE), chiamato BIP (Band Interleaved by Pixel).

La sequenza utilizzata è segnalata, all'interno della *label*, dalla *keyword* `AXIS_NAME`, la quale contiene una delle successioni precedenti.

Riguardo al formato dei dati il discorso va affrontato separando il caso del *core* da quello dei *suffix plane*: nel primo caso, infatti, tutti i pixel devono avere lo stesso formato all'interno di un file. Lo standard PDS prevede che tali pixel possano essere salvati utilizzando uno, due o quattro byte per ciascuno: nel primo caso si hanno pixel di tipo *unsigned integer*, nel secondo *integer* e nel terzo *real*.

Il discorso è, invece, leggermente differente quando si tratta dei *suffix plane*. In queste regioni, infatti, tutti i dati sono, generalmente, salvati utilizzando 4 byte per ogni pixel, anche se poi il tipo del dato può essere di natura differente in casi diversi. In generale, comunque, i tipi di dato che si possono incontrare sono gli stessi tre citati per il *core*.

È evidente come questa differenza sia importante nel momento in cui si progetta un software che abbia lo scopo di aprire e leggere questi file, in quanto il numero di byte letti ad ogni ciclo deve essere adeguato al tipo di regione nella quale il puntatore al file si trova in un determinato istante. Ciò implica anche che al momento della creazione del suddetto software si debba fare in modo di sapere quale regione si sta leggendo in ogni istante.

## 3.2) I cubi di VIMS

I file dati, o cubi, di VIMS sono tutti scritti rispettando le modalità consigliate dallo standard PDS: i file, infatti, sono composti da record di lunghezza fissa, pari a 512 byte; utilizzano un record fisico per ogni record logico e protocolli di scrittura standard sia per la parte testo che per quella binaria.

Riguardo al formato utilizzato per la scrittura, va evidenziato che tutti i cubi sono salvati utilizzando l'ordine di scrittura BIL, ovvero, SAMPLE, BAND, LINE. Il *core* è salvato utilizzando 2 byte per ogni pixel, cosicché il tipo di dato è un *integer*. Per le regioni che estendono le dimensioni del *core*, invece, va detto che sono presenti, generalmente, solo quelle poste oltre il termine delle coordinate spettrali e di quelle spaziali, vale a dire il *backplane* ed il *sideplane*. Entrambe tali regioni sono scritte su file utilizzando 4 byte per ogni pixel ma si differenziano per il numero di piani dai quali sono formati, come sarà mostrato più avanti, nell'esempio di *label* di VIMS riportato.

Va notato, inoltre, che il formato utilizzato per i dati di VIMS prevede l'utilizzo delle cosiddette *attached label*. I file, quindi, contengono, al loro interno, sia una parte in formato testo (ASCII), che descrive il dato contenuto nel file, sia una parte binaria che rappresenta il dato vero e proprio. Al fine di permettere ad un tool appositamente creato di rintracciare, all'interno del file, l'inizio della parte binaria, tra le *keyword* della *label* viene inserito un puntatore ai dati veri e propri.

### 3.2.1) La label presente nei cubi

Di seguito viene proposto un esempio di label tratta da un file acquisito durante la fase *cruise*, la fase, cioè, di volo che ha avuto luogo tra la partenza della sonda dalla Terra e l'arrivo in prossimità



di Saturno. Più in particolare tale file è stato acquisito durante il *fly-by* di Giove avvenuto nel 2000.

La struttura di tale *label* ricalca quella spiegata nel capitolo precedente, con la *label* stessa suddivisa in settori ognuno dei quali ha la funzione di spiegare un determinato aspetto del file o della missione. Si può, quindi, osservare il settore che descrive la struttura del cubo che contiene la parte binaria, la descrizione della fase della missione, lo stato di funzionamento dello strumento, ecc.

Accanto alla *label* vera e propria viene proposta una breve descrizione delle *keyword* più significative.

### 3.2.1.1) Parte iniziale della label

```
CCSD3ZF0000100000001NJPL3IF0PDS200000001 = CASSFDU_LABEL  
PDS_VERSION_ID = "PDS3"
```

Questa prima sezione contiene, nella prima riga, una *keyword* il cui scopo è riferire la label stessa alla missione Cassini. La seconda riga identifica la versione dello standard PDS utilizzata.

### 3.2.1.2) Struttura del file

```
/* File Structure */  
  
RECORD_TYPE      = FIXED_LENGTH  
RECORD_BYTES     = 512  
FILE_RECORDS     = 5983  
LABEL_RECORDS    = 19  
FILE_STATE       = CLEAN  
  
^HISTORY         = 20  
OBJECT           = HISTORY  
END_OBJECT      = HISTORY  
  
^QUBE = 45
```

La prima parte di questa sezione definisce le caratteristiche principali: RECORD\_TYPE, innanzitutto, fissa il tipo di record utilizzato e RECORD\_BYTES ne mostra la lunghezza in termini di byte. La *keyword* FILE\_RECORDS mostra la lunghezza del file in termini di numero di record occupati e la successiva, LABEL\_RECORDS, fa altrettanto per l'*header*, fornendo, così, un primo riferimento sulla suddivisione dello spazio occupato, all'interno del file, dalla parte ASCII e dalla parte binaria.

Segue un oggetto HISTORY, che è utilizzato per lasciare uno spazio vuoto tra *header* e dati binari al fine di avere una area per introdurre successivamente la descrizione di ulteriori elaborazioni che sono state eseguite sul dato presentato. L'oggetto è preceduto da un puntatore ^HISTORY che individua la posizione di partenza, in termini di record, di questa parte di *header*. In ultimo si trova il puntatore alla parte binaria, ^QUBE.

Il meccanismo per capire il funzionamento dei puntatori è molto semplice: per sapere il byte di partenza della parte, all'interno del file, indicata dal puntatore è sufficiente moltiplicare il numero di record presente come valore associato al puntatore (in questo caso 20 per HISTORY, 45 per QUBE) per il numero di byte dai quali è costituito un singolo record.

### **3.2.1.3) Struttura del cubo:**

```
/* Qube structure: Standard ISIS Cube of VIMS Data */
```

```
OBJECT = QUBE  
AXES = 3  
AXIS_NAME = (SAMPLE,BAND,LINE)
```

```
/* Core description. */
```

```
CORE_ITEMS = (64,352,64)  
CORE_ITEM_BYTES = 2  
CORE_ITEM_TYPE = SUN_INTEGER  
CORE_BASE = 0.0  
CORE_MULTIPLIER = 1.0
```

```
CORE_VALID_MINIMUM = -4095
CORE_NULL = -8192
CORE_LOW_REPR_SATURATION = -32767
CORE_LOW_INSTR_SATURATION = -32766
CORE_HIGH_REPR_SATURATION = -32764
CORE_HIGH_INSTR_SATURATION = -32765
CORE_MINIMUM_DN = -36
CORE_NAME = RAW_DATA_NUMBER
CORE_UNIT = DIMENSIONLESS
```

```
/* Suffix description. */
```

```
SUFFIX_ITEMS = (1,4,0)
SUFFIX_BYTES = 4
SAMPLE_SUFFIX_NAME = BACKGROUND
SAMPLE_SUFFIX_UNIT = DIMENSIONLESS
SAMPLE_SUFFIX_ITEM_BYTES = 4
SAMPLE_SUFFIX_ITEM_TYPE = SUN_INTEGER
SAMPLE_SUFFIX_BASE = 0.0
SAMPLE_SUFFIX_MULTIPLIER = 1.0
SAMPLE_SUFFIX_VALID_MINIMUM = 0
SAMPLE_SUFFIX_NULL = -8192
SAMPLE_SUFFIX_LOW_REPR_SAT = -32767
SAMPLE_SUFFIX_LOW_INSTR_SAT = -32766
SAMPLE_SUFFIX_HIGH_REPR_SAT = -32764
SAMPLE_SUFFIX_HIGH_INSTR_SAT = -32765
BAND_SUFFIX_NAME = (IR_DETECTOR_TEMP_HIGH_RES_1,
                    IR_GRATING_TEMP,
                    IR_PRIMARY_OPTICS_TEMP,
                    IR_SPECTROMETER_BODY_TEMP_1)
BAND_SUFFIX_UNIT = (DIMENSIONLESS, DIMENSIONLESS,
                    DIMENSIONLESS, DIMENSIONLESS)
BAND_SUFFIX_ITEM_TYPE = (SUN_INTEGER, SUN_INTEGER,
                          SUN_INTEGER, SUN_INTEGER)
```

All'inizio di questa sezione si trovano le *keyword* recanti il numero e il nome, nonché la disposizione, degli assi del *core*. Subito dopo, indicata dalla *label* “/\*core description \*/”, è posta la descrizione del *core* stesso: CORE\_ITEMS indica il numero di pixel

che si trovano lungo ognuno dei 3 assi del *core*. CORE\_ITEM\_BYTES E CORE\_ITEM\_TYPE descrivono il tipo ed il formato dei dati del *core* stesso.

Le due successive *keyword* meritano un discorso a parte. Quando si lavora con lo standard PDS bisogna tener presente che, per ragioni di compatibilità con lo standard ISIS, uno standard di archiviazione precedente al PDS, qualunque sia il tipo di dato usato per salvare i risultati delle acquisizioni, concettualmente, il *core* va sempre pensato come costituito da numeri reali. Allo scopo di permettere una conversione tra il tipo utilizzato per scrivere i valori e il tipo *real*, si utilizzano queste *keyword* nel modo seguente:

$$\text{valore\_real} = \text{CORE\_BASE} + (\text{CORE\_MULTIPLIER} * \text{valore})$$

Seguono una serie di *keyword* che indicano dei valori estremi per lo strumento, quali i valori di saturazione.

Le ultime due *keyword*, CORE\_NAME e CORE\_UNIT, specificano, rispettivamente, che tipo di valori sono contenuti nel *core* e la relativa unità di misura.

La seconda parte della sezione, identificata da “/\* suffix description \*/”, è analoga alla precedente ed è riferita ai piani ausiliari, *backplane* e *sideplane*. Questa parte è divisa in due blocchi, il primo relativo al *sideplane*, il secondo al *backplane*. Le *keyword* del tipo asse\_SUFFIX\_NAME specificano il tipo di dato contenuto: valori di corrente di buio nel primo, temperature di alcune componenti dello strumento nel secondo. Successivamente, per entrambi, sono indicate le unità di misura dei suddetti valori e il tipo di dato utilizzato per la loro rappresentazione.

#### **3.2.1.4) Dati di carattere generale**

```
/* Data description: general */
```

```
MISSION_NAME = "CASSINI"  
MISSION_PHASE_NAME = "EARTH-JUPITER CRUISE"  
INSTRUMENT_HOST_NAME = "CASSINI ORBITER"
```

```
INSTRUMENT_NAME = "VISUAL AND INFRARED MAPPING  
SPECTROMETER"  
INSTRUMENT_ID = "VIMS"  
DATA_SET_ID = "CO-J/JSA-VIMS-3-QUBE-V1.0"  
FLIGHT_SOFTWARE_VERSION_ID = "3.1"  
SOFTWARE_VERSION_ID = "VIMS V8.0 01-29-2001"  
TARGET_NAME = "JUPITER"  
PRODUCT_ID = "1_1353964221.095"  
SPACECRAFT_CLOCK_CNT_PARTITION = 1  
SPACECRAFT_CLOCK_START_COUNT = 1353964221.095  
SPACECRAFT_CLOCK_STOP_COUNT = 1353964312.220  
NATIVE_START_TIME = 1353964219.13978  
NATIVE_STOP_TIME = 1353964309.11082  
START_TIME = "2000-331T20:59:02.160Z"  
STOP_TIME = "2000-331T21:00:31.966Z"  
HOUSEKEEPING_CLOCK_COUNT = 1353964240.133  
PRODUCT_CREATION_TIME = "2001-058T13:18:31.000Z"  
OBSERVATION_ID = "VIMS_C23JU_18ATM2X2097_ISS"  
COMMAND_FILE_NAME = "JDISK13a.ioi"  
COMMAND_SEQUENCE_NUMBER = 14  
EARTH_RECEIVED_START_TIME = "2000-333T03:27:22.873Z"  
EARTH_RECEIVED_STOP_TIME = "2000-333T03:29:40.977Z"  
MISSING_PACKET_FLAG = "NO"  
NOTE = "N/A"  
PARAMETER_SET_ID = "JDISK13A1"  
PRODUCT_VERSION_TYPE = "FINAL"  
SEQUENCE_ID = "C23"  
TELEMETRY_FORMAT_ID = "S&ER3"  
DATA_REGION = (1,1,1,64,352,64)
```

In questa sezione sono fornite molte informazioni di carattere generale riguardanti la missione e l'acquisizione. Innanzitutto è specificato il nome per esteso e l'acronimo dello strumento stesso. Subito dopo viene indicato l'identificativo del *data set* di appartenenza. Seguono la versione del software di bordo utilizzato dallo strumento per l'acquisizione e quella relativa al software di generazione dei cubi a partire dai dati di telemetria inviati a terra dopo le osservazioni.

Continuando a scorrere la *label* si trova, per prima la *keyword* TARGET\_NAME, la quale fornisce l'informazione relativa al target osservato; successivamente PRODUCT\_ID identifica in maniera univoca il cubo. A questo punto sono poste alcune *keyword* che mostrano, in diversi formati, gli istanti di inizio e fine acquisizione, quelli di inizio e fine ricezione a terra dei dati ed il momento di creazione del presente file.

Alla fine della sezione si trovano SEQUENCE\_ID, che indica la sequenza di osservazione della quale la presente fa parte e DATA\_REGION, che indica la regione della matrice 3-D effettivamente utilizzata, partendo dall'eventuale offset e indicando poi l'occupazione lungo ciascun asse.

### **3.2.1.5) Stato di funzionamento dello strumento**

```
/* Instrument Status (IR, Visible) */
```

```
INSTRUMENT_MODE_ID = "IMAGE"  
INTERFRAME_DELAY_DURATION = 280.000000  
COMPRESSOR_ID = 1  
INST_CMPRS_NAME = "OMEGA"  
INST_CMPRS_RATIO = 5.035511  
DATA_BUFFER_STATE_FLAG = "ENABLED"  
INSTRUMENT_DATA_RATE = 94.208000  
MISSING_PIXELS = 0  
POWER_STATE_FLAG = ("ON","ON")  
GAIN_MODE_ID = ("LOW","LOW")  
EXPOSURE_DURATION = (20.000000,1280.000000)
```

Ha inizio, in questa sezione, la descrizione dello stato dello strumento durante l'acquisizione, suddiviso in più sotto-sezioni: viene specificato, innanzitutto, la modalità di acquisizione: IMAGE; vengono poi fornite informazioni sulla compressione dei dati: INSTR\_CMPRS\_NAME indica il tipo di compressione utilizzata e INSTR\_CMPRS\_RATIO il fattore di compressione.

Tra le informazioni più significative, poi, vanno citate quelle date da due *keyword* in particolare: `POWER_STATE_FLAG`, che indica quale dei due canali, rispettivamente infrarosso e visibile, siano accesi e `EXPOSURE_DURATION`, che fornisce i tempi di acquisizione di entrambi i canali.

```
/* IR high resolution, IR low resolution, Visible */  
DETECTOR_TEMPERATURE = (61.716393,60.942924,231.506805)  
  
/* IR primary, IR secondary, Visible */  
OPTICS_TEMPERATURE = (141.269485,132.399017,272.008240)
```

Successivamente sono mostrate le temperature della camera CCD e delle ottiche sia per il canale infrarosso che per quello visibile.

```
/* Sampling modes shown are IR, visible spatial, and visible spectral */  
SAMPLING_MODE_ID = ("UNDER","HI-RES","NORMAL")
```

A seguire si trova l'informazione relativa alla risoluzione spettrale e spaziale dell'acquisizione, ed è caratteristica di uno strumento come VIMS, in grado di fornire contemporaneamente dati di tipo spettrale e spaziale.

```
/* Instrument status: IR */  
BIAS_STATE_ID = "LOW"  
SCAN_MODE_ID = "BOTH"  
SHUTTER_STATE_FLAG = "ENABLED"  
INTEGRATION_DELAY_FLAG = "ENABLED"  
INTERLINE_DELAY_DURATION = 121.000000  
BACKGROUND_SAMPLING_MODE_ID = "SINGLE"  
BACKGROUND_SAMPLING_FREQUENCY = 0
```

Questo gruppo di *keyword* descrive, complessivamente, lo stato di funzionamento del canale infrarosso, fornendo informazioni complementari a quelle già viste nelle parti precedenti.

**Elaborazione ed archiviazione on-line di dati provenienti da  
missioni relative all'esplorazione del Sistema Solare**

---

```
/* Temperatures shown are for spectrometer, then grating. */
INSTRUMENT_TEMPERATURE = (139.692810,138.186264)
FAST_HK_ITEM_NAME = ("IR_DETECTOR_TEMP_HIGH_RES_1",
                    "IR_GRATING_TEMP",
                    "IR_PRIMARY_OPTICS_TEMP",
                    "IR_SPECTROMETER_BODY_TEMP_1")

FAST_HK_PICKUP_RATE = 2

/* Instrument status: visible */
ANTIBLOOMING_STATE_FLAG = "OFF"
```

In questa sotto-sezione, tra le altre cose, sono illustrate le temperature di alcune componenti dello strumento, specificando separatamente le temperature e le componenti dello strumento per le quali tali temperature sono state rilevate.

```
/* Spectral axis description */
GROUP = BAND_BIN
BAND_BIN_CENTER = (0.35, ... , 5.102)

BAND_BIN_UNIT = MICROMETERS
BAND_BIN_ORIGINAL_BAND = (0, ... , 351)
END_GROUP = BAND_BIN

END_OBJECT = QUBE
END
```

In questa sezione finale sono mostrati due array molto importanti, riportati solo parzialmente per motivi di spazio. Il primo, identificato da `BAND_BIN_CENTER`, fornisce la lunghezza d'onda  $\lambda$  relativa al centro della banda che viene raccolta dal pixel appartenente ad una certa colonna della camera CCD. I valori appartenenti all'array identificato dalla *keyword* successiva, `BAND_BIN_ORIGINAL_BAND`, indicano a quale colonna della CCD corrispondono le lunghezze d'onda presenti nell'array precedente. Infine si ha la chiusura dell'oggetto *qube* e della intera *label*.



## Parte II

### L'archivio on-line di VIMS-V

In questa seconda parte della tesi verrà mostrato il lavoro svolto durante il processo di progettazione e realizzazione dell'archivio on-line presso ASDC. Tale lavoro è stato suddiviso in più fasi, ognuna delle quali ha riguardato uno dei passi necessari alla creazione di tutta quella serie di programmi e script necessari per il funzionamento del sistema.

I passi appena citati sono i seguenti:

- ✚ Progettazione dell'archivio. Scelta dei parametri ingegneristici e osservativi da utilizzare nelle ricerche.
- ✚ Creazione degli script di lettura dei file di dati, degli elenchi di valori da inserire nella tabella e successivo inserimento.
- ✚ Creazione delle immagini di “preview” estratte dai file di dati.
- ✚ Scelta dei metodi di ricerca da mettere a disposizione degli utenti e dei metodi di visualizzazione dei risultati ottenuti in seguito all'esecuzione delle ricerche.
- ✚ Realizzazione della pagina di ricerca: gestione dell'accesso ai dati pubblici e a quelli riservati; schema di formazione della pagina stessa mediante utilizzo di differenti moduli.

- ✚ Esecuzione dei processi di ricerca e visualizzazione dei relativi risultati.
- ✚ Creazione della pagina dedicata al singolo file e funzionamento dell'applet per la "quick-look" dei dati.

Come già accennato nella parte I della presente tesi, ci si limiterà alla descrizione del lavoro svolto per uno strumento in particolare. Tale strumento è lo spettroscopio ad immagini VIMS-V. Nonostante l'archivio presente presso ASDC ospiti, infatti, dati provenienti da altri strumenti, si è ritenuto eccessivo illustrare il lavoro svolto per ciascuno di tali strumenti. Ciò in virtù del fatto che, in linea generale, i passi seguiti per i suddetti strumenti sono stati, tenute conto le dovute differenze causate dalle specificità di ognuno, gli stessi.

Si è, quindi, scelto di puntare ad una accurata descrizione di quanto messo in opera per VIMS-V, illustrando i motivi che hanno portato a fare certe scelte, presentando tutto il materiale software che si è sviluppato durante le diverse fasi del lavoro e limitandosi a mettere in evidenza, in questa sede, che tale lavoro non è stato il solo svolto ma che si è deciso, per i motivi appena riportati, che sia il solo presentato.

## Capitolo 4

# La creazione del database di VIMS-V

### 4.1) I parametri per le ricerche

#### 4.1.1) I criteri per la scelta

Nel capitolo 2 si è presentato il formato PDS, lo standard utilizzato per l'archiviazione dei dati relativi all'esplorazione del Sistema Solare, e, nel capitolo successivo, si è visto nel dettaglio come tale standard è utilizzato per i dati acquisiti da *VIMS-V*.

È stato mostrato come ogni file contenga, oltre alla parte binaria riguardante i dati veri e propri, un “*header*”, scritto in formato ASCII, il quale descrive minuziosamente il contenuto del file, le condizioni di funzionamento dello strumento e la posizione della sonda durante l'acquisizione. Tale descrizione avviene mediante l'utilizzo di un notevole numero di “*keyword*”, ad ognuna delle quali è attribuito un valore.

Il primo passo seguito nella creazione del database dei dati acquisiti da *VIMS-V* è stato decidere quali fossero, tra tutti quelli presenti nell'*header*, i parametri osservativi ed ingegneristici più utili agli utenti per eseguire le loro ricerche.

Un requisito fondamentale, nel momento in cui ci si avvia ad operare una scelta di questo genere, è, senz'ombra di dubbio, una buona conoscenza della missione e, più in particolare, dello strumento. Lo scopo del lavoro di creazione di un archivio on-line, qual è quello in questa tesi descritto, è quello di mettere a disposizione della comunità scientifica uno strumento in grado di rendere semplice la gestione di una grande mole di dati e di discriminare facilmente i file con i dati che interessano per un certo tipo di lavoro da tutti gli altri. Allo scopo di riuscire a fornire un servizio che sia il più efficiente possibile da questo punto di vista, è necessario, quindi, capire quali siano le necessità dei futuri utenti, al fine di comprendere in che modo cercheranno di utilizzare il database e quali siano i tipi di ricerca più utili per loro.

Un passo necessario, e perciò effettuato immediatamente all'avvio del lavoro, è stato quello di dialogare con alcune delle persone che sono coinvolte nel team che ha realizzato e gestisce lo strumento in modo da avere un'indicazione diretta del tipo di lavoro che viene svolto sui dati e di come questi siano letti ed interpretati. È facilmente comprensibile, infatti, che tipologie di strumenti differenti quali, ad esempio, un radar e uno spettrometro a immagini producono dati completamente differenti e richiedono un tipo di lavoro totalmente diverso per il loro utilizzo e la loro comprensione. Anche i parametri utili per le ricerche dei file, di conseguenza, saranno dissimili. Dallo scambio di informazioni con il team, quindi, nasce la lista dei parametri che saranno utilizzati per le ricerche.

Naturalmente, in una fase preliminare rispetto al suddetto dialogo, è necessario dedicare del tempo allo studio dello svolgimento della missione, per capire l'ambito nel quale si troverà ad operare lo strumento del quale ci si dovrà occupare. In un secondo momento, poi, una volta compreso ciò, è necessario farsi un'idea ben chiara delle caratteristiche fondamentali dello strumento stesso, delle modalità operative e di quali siano i parametri fondamentali che ne caratterizzano il comportamento durante la fase di acquisizione dei dati e di quali siano, al contrario, quelli secondari o dedicati a descrivere caratteristiche generali, tali da avere valori costanti per

tutta la missione o per fasi talmente estese da non costituire un buon fattore di discriminazione in sede di ricerca.

Durante la discussione relativa alla decisione sulle *keyword* da utilizzare si è tenuto conto di alcuni semplici criteri: uno dei primi ad essere individuati è stato senza dubbio la facilità di uso. Alcune *keyword*, infatti, hanno valori molto complessi derivanti dall'unione di differenti sigle alfanumeriche. Tali *keyword* sono, pertanto, di difficile utilizzo, in quanto appare improbabile che siano facilmente memorizzabili da un utente.

Tale criterio non può, comunque, per ragioni facilmente comprensibili, essere l'unico. Un fattore di primaria importanza, e del quale, quindi, si è sommamente tenuto conto nella scelta delle *keyword*, è rappresentato dallo stato di funzionamento dello strumento. Un simile criterio, infatti, può risultare molto utile, ad esempio, nel momento in cui si voglia eseguire un controllo del funzionamento di una certa modalità di acquisizione, o per discriminare, tra tutte le osservazioni eseguite relativamente ad un certo *target*, quelle effettuate utilizzando una certa modalità. La maggior parte dei parametri scelti in questa fase iniziale ha risposto proprio a tale esigenza di caratterizzazione del funzionamento dello strumento.

Alle *keyword* scelte in base ai due precedenti criteri si sono affiancate altre derivanti da considerazioni piuttosto ovvie, quali, ad esempio, specificare il *target* osservato, oppure la sessione, insieme di acquisizioni relative ad un certo periodo, la fase della missione o il nome del file contenente l'acquisizione.

Un insieme di *keyword* che avrebbero potuto essere molto utili per individuare le osservazioni di interesse è costituito, senza dubbio, dall'insieme dei parametri contenenti informazioni sulla geometria dell'osservazione. Tali parametri forniscono tutte le indicazioni necessarie a ricostruire quale fosse, al momento dell'acquisizione, la posizione della sonda che ospita lo strumento rispetto al *target* e rispetto al Sole. Tali informazioni, purtroppo, al momento della creazione del database di *VIMS-V* non erano disponibili e non sono, pertanto, state incluse nell'insieme di

parametri utilizzabili per le ricerche. L'implementazione della possibilità di ricerca mediante l'utilizzo di queste informazioni è previsto per il prossimo futuro, come attività di aggiornamento del database.

### **4.1.2) I parametri scelti**

Qui di seguito viene presentata la lista delle *keyword* selezionate per *VIMS-V* con la spiegazione del tipo di selezione dei dati che è possibile effettuare mediante il loro utilizzo:

**FILENAME:** parametro per la selezione del file o dei file mediante l'introduzione del nome completo o di una parte di esso.

**OBS\_SESSION:** nome della sessione di osservazione. Tale parametro rappresenta un insieme di osservazioni relative ad un determinato arco di tempo o ad una parte del missione. Dal momento dell'inserzione della sonda *Cassini* in orbita attorno al pianeta Saturno, ogni insieme di acquisizioni relative ad un orbita è inserita all'interno di una sessione denominata "Snn", essendo *nn* un numero progressivo indicante il numero dell'orbita percorsa.

**PATH:** ogni sessione può contenere una suddivisione in sotto-sessioni, relative ad intervalli temporali facenti parte dell'intervallo che è identificato dalla sessione. Il parametro "path" consente di specificare tali sotto-intervalli.

**DATATYPE:** specifica se si tratti di un file grezzo o calibrato.

**TARGET:** come dice il nome stesso, tale parametro consente di specificare il corpo osservato da *VIMS-V*.

**CORE\_ITEMS:** utilizzando questo parametro viene effettuata la ricerca sull'omonima *keyword* presente all'interno del file. Come già spiegato nel capitolo precedente con questo parametro viene specificato quante *bands*, *samples* e *lines* costituiscono l'acquisizione, ovvero, quali siano le dimensioni del "cubo" di dati contenuto nel file.

**DATA\_REGION:** questa *keyword* è collegata alla precedente. Ogni acquisizione effettuata da *VIMS-V*, infatti, altro non è che la lettura della radiazione arrivata su una certa regione della camera CCD. Diversi modi operativi, a causa di limiti nella capacità di memorizzazione dei dati, comportano l'utilizzo di regioni differenti di tale CCD. L'utilizzo del parametro `DATA_REGION` consente di ricercare file ottenuti utilizzando una certa regione della CCD.

**EXP\_DURATION:** consente di ricercare file specificando il tempo di esposizione per i due canali, infrarosso e visibile.

**INSTRUMENT\_MODE\_ID:** permette di discriminare, tra tutti i file di acquisizioni presenti nell'archivio, quelli relativi ad una certa modalità operativa. Valori possibili sono, ad esempio, "image", "point" o "line", al variare della tipologia di scansione effettuata.

**MISSION\_PHASE\_NAME:** consente di specificare in quale fase della missione si vogliono limitare le ricerche

## 4.2) MySQL

Il database utilizzato per la creazione dell'archivio on-line di *VIMS-V* è un noto RDBMS (Relational DataBase Management System) molto utilizzato sia in ambienti Unix che Windows, ovvero MySQL. La scelta di utilizzare proprio MySQL è dovuta ad alcune semplici considerazioni: innanzitutto si tratta di un database molto veloce. Tutti i test effettuati sulle prestazioni, disponibili in rete, dimostrano le alte prestazioni che è in grado di raggiungere. Tale caratteristica è, come sempre quando si ha a che fare con sistemi informatici, davvero critica. Ogni strumento produce grandi moli di dati, essendo testato terra prima del lancio, subito dopo il lancio e durante il volo, e dovendo operare per mesi o anni in orbita attorno al proprio *target*. Tutto ciò si traduce in un impressionante numero di file.

A titolo di esempio, allo scopo di rendere conto dell'entità di tale numero si consideri proprio il caso di *VIMS-V*. La sonda *Cassini*,

a bordo della quale è ospitato *VIMS*, è stata lanciata nel 1997 e, dopo un viaggio di 7 anni, è entrata in orbita attorno a Saturno nel 2004. Durante tale periodo sono state fatte acquisizioni per un totale di circa 16 mila file. Durante i primi due anni di stazionamento della *Cassini* in orbita sono stati acquisiti circa 26 mila file, con una media, quindi, di circa 13 mila file all'anno. Considerando che la missione nominale ha una durata prevista di 4 anni si avranno, alla fine di tale periodo, mantenendo una tale media di file all'anno, circa 68 mila file. È necessario, però, tener conto, che, se non si presenteranno inconvenienti tali da compromettere l'operatività della sonda e dello strumento, è già prevista una estensione della missione per un periodo complessivo pari ad altri 4 anni. Tenendo per buona la media citata in precedenza, si avrà, a fine missione, un totale di circa 120 mila file.

Questo valore giustifica da solo, senza il bisogno di commenti superflui, la necessità di utilizzare un prodotto dalle elevate prestazioni, che sia in grado di eseguire ricerche in un insieme così vasto di file in tempi brevi.

La seconda ragione per la quale è stato scelto MySQL è che si tratta di un prodotto estremamente affidabile. Al contrario di altri RDBMS presenti nel mercato è stato progettato e sviluppato proprio allo scopo di gestire grandi quantità di dati. Va tenuto conto del fatto che lo stesso RDBMS è utilizzato anche per tutti gli altri strumenti che sono presenti, o che verranno aggiunti in futuro, nell'archivio on-line. È, quindi, di vitale importanza poter contare su un sistema software che non soffra eccessivamente il crescente carico di lavoro che si è trovato e si troverà a dover affrontare col passare degli anni.

Un terzo e non trascurabile aspetto di MySQL è costituito dalla sua natura di prodotto *open source*. Con tale termine si indica un software rilasciato con un tipo di licenza per la quale il codice sorgente è lasciato alla disponibilità di eventuali sviluppatori, in modo che con la loro collaborazione, in genere libera e spontanea, il prodotto finale possa raggiungere una complessità maggiore di quanto potrebbe ottenere un singolo gruppo di programmazione. In linea generale i vantaggi di questo tipo di prodotto sono molteplici:



- per la sua natura aperta, è gratuito;
- essendo possibile modificare liberamente il software, è possibile personalizzarlo ed adattarlo alle proprie esigenze;
- il codice sorgente è sottoposto ad una revisione da parte di moltissime persone, pertanto è più difficile che contenga bachi e malfunzionamenti. In ogni caso, è sempre possibile per chiunque tenere un indice pubblico dei problemi, in modo che gli utenti li conoscano;
- se viene scoperto un baco o una falla di sicurezza, la sua correzione di solito è molto rapida;
- essendo il sorgente liberamente consultabile, non è possibile inserire intenzionalmente nel software *backdoor*, *trojans* o *spyware* senza che questi vengano prontamente scoperti ed eliminati, come invece è accaduto per alcuni software commerciali;
- non esistendo standard proprietari, le cui specifiche sono normalmente segrete, è molto più facile costruire software interoperabile;
- permettere a chiunque di modificare i sorgenti garantisce che ogni nuova funzionalità o copertura di un baco possa essere proposta da chiunque e immediatamente applicata dagli sviluppatori. Questo permette di avere già a disposizione un software che rispetta le esigenze di chi ha richiesto le modifiche;

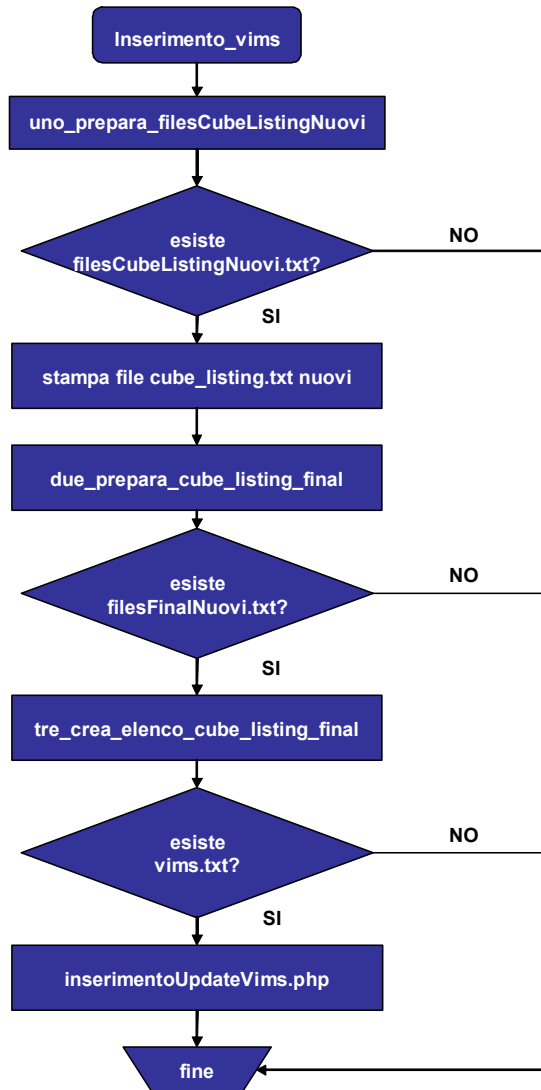
Non tutti i vantaggi appena elencati, comprensibilmente, sono stati considerati rilevanti per il lavoro svolto in ASDC ma è comprensibile come quelli riguardanti la sicurezza e l'usabilità su tutte le piattaforme sono stati valutati come di primaria importanza.

## 4.3) La procedura di inserimento dei dati nel database

La scrittura della procedura di inserimento dei valori relativi alle *keyword* selezionate per il database di *VIMS-V* ha richiesto la creazione di una serie di *script* che vengono chiamati in sequenza da uno *script* “contenitore” che si incarica di far eseguire agli *script* “specializzati” tutti i passi necessari, dal controllo di quali siano i file nuovi, fino alla scrittura delle informazioni nel database. Il tutto è svolto nella corretta sequenza.

Tutti gli *script*, ad eccezione di uno, sono stati scritti utilizzando il linguaggio di *scripting* “*C shell*” o “*perl*”. Il database di *VIMS-V*, come del resto tutti gli altri presenti in ASDC, è installato su una macchina sulla quale gira, come sistema operativo una delle più affidabili distribuzioni linux, per l'esattezza si tratta di “Fedora” core 2. La scelta di utilizzare come sistema operativo una delle distribuzioni di linux si basa sulla consuetudine, da parte della maggioranza degli enti pubblici o privati che posseggono delle macchine server, di affidarsi a quello che viene ritenuto il più affidabile dei sistemi operativi, linux appunto. È da sottolineare il fatto che linux fa parte della categoria dei software *open source* e, come tale, offre tutti quei vantaggi che sono già stati riportati a proposito di MySQL. Oltre ai suddetti vantaggi è di notevole rilevanza il fatto che linux, al contrario di Windows, non sia continuamente oggetto di attacco da parte di nuovi virus. Ciò contribuisce a renderlo ancora più apprezzato da parte di chi ha necessità di mettere in rete uno o più server. A tutto ciò si aggiunge una vecchia consuetudine, nell'ambiente scientifico ed universitario, ad utilizzare questo tipo di piattaforma, che proprio in tale ambiente è stato molto sviluppato.

Il *C shell* si caratterizza come un linguaggio piuttosto versatile per alcuni tipi di lavoro, in particolare quando si ha a che fare con semplici operazioni di lettura e scrittura di file.



**Figura 4. 1** schema di funzionamento dello script di inserimento dei dati di VIMS-V nel database.

Il *perl*, invece, si utilizza per operazioni più complesse di lettura ed elaborazione del contenuto testuale dei file. Tale linguaggio, infatti, nasce appositamente per gestire in maniera efficiente la gestione delle stringhe. Il terzo linguaggio di scripting utilizzato è stato il *php*, usato, in questo caso, per l'inserimento dei dati letti nei file di testo generati dagli script realizzati utilizzando gli altri due linguaggi in MySQL. Il *php*, infatti, presenta una serie di comandi e routine scritte proprio per essere utilizzate in combinazione con MySQL.

Al momento di stabilire la procedura di inserimento dei dati nel database e di organizzare le procedure che avrebbero svolto tale lavoro si è scelto, in maniera arbitraria, di affidarsi ad una serie di routine semplici, ognuna delle quali impiegata per eseguire un compito ben preciso, piuttosto che crearne una sola più complessa. Il motivo principale per il quale si è presa una simile decisione risiede nella possibilità offerta da questa impostazione di poter intervenire, qualora ce ne fosse stato bisogno, su una delle procedure per apportare le necessarie modifiche, essendo sicuri di non toccare tutto il resto. Quando si è cominciato lo sviluppo del sistema, infatti, si era ben consci del fatto che sarebbe stato possibile possibile, come in effetti è poi accaduto, che l'*header* dei cubi avrebbe subito delle modifiche o degli aggiornamenti. Una simile evenienza richiede, pertanto, un intervento di aggiornamento almeno su una parte delle procedure scritte per renderle compatibili con il nuovo *header*.

Qui di seguito, verrà ora illustrato tutto il processo eseguito per l'inserimento dei dati, descrivendo le operazioni svolte da ciascuno degli *script* illustrati in Fig. 4.1.

### **4.3.1) inserimento\_vims**

È lo *script* contenitore, che si occupa di eseguire nella giusta sequenza tutti gli altri, realizzato in *C shell*. Le operazioni svolte possono essere così descritte:

- ricerca delle sessioni e sottosessioni inserite;

- generazione dei file “cube\_listing\_final.txt”;
- creazione dell’elenco di tutti i file da leggere per l’inserimento;
- inserimento dei dati.

Come si può vedere dalla Fig. 4.1, in tre occasioni *inserimento\_vims* controlla che esista un determinato file di testo per poter proseguire con le operazioni successive. Se tali file non sono stati generati, evidentemente si è verificato un qualche problema durante l’esecuzione dello *script* precedente e quindi la variabile *proseguì* viene impostata a “q” (quit). In conseguenza di ciò lo *script* viene immediatamente terminato. In caso contrario, se cioè il file cercato esiste, la suddetta variabile rimane impostata su “ok” e il processo può proseguire. Viene, inoltre, lasciata libertà all’utente che si accorga di qualcosa che non funziona correttamente, di interrompere l’esecuzione prima dell’avvio di ognuno degli *script* “specializzati”. Infatti, al termine di ciascuno di essi e prima dell’avvio del successivo, il sistema chiede all’utente se continuare o interrompere, impostando, come nel caso precedente, la variabile *proseguì* a “q”.

### 4.3.2) uno\_prepara\_filesCubeListingNuovi

Questo *script*, scritto in *C shell*, segna l’avvio del processo di inserimento dei dati nel database. È molto semplice e si occupa di preparare un file di testo con l’elenco delle sotto-sessioni di dati copiati di recente sull’*hard disk* del server.

Tutti i dati relativi allo strumento *VIMS-V* sono custoditi all’interno della cartella */vims*, posta, come indicato dallo “*slash*” / iniziale direttamente sotto la cartella *root* di sistema. All’interno della suddetta cartella sono presenti tre cartelle: “RAW”, “CAL”, “PREVIEW”, le quali sono dedicate, rispettivamente, ai dati grezzi, calibrati ed alle immagini di *preview*, il cui utilizzo sarà spiegato in seguito. Le cartelle “RAW” e “CAL” sono utilizzate per contenere due differenti formati degli stessi dati: all’interno della cartella

“RAW” sono scritte tutte le sessioni di dati non calibrati o grezzi, da cui il nome; all'interno della cartella “CAL”, al contrario, sono conservati i dati nella forma calibrata.

Inizialmente, in una prima versione di questo script, si era deciso di far eseguire al sistema una ricerca all'interno di tutta la cartella “vims” allo scopo di individuare quali fossero le sotto-sessioni nuove. Ciò avveniva mediante un confronto tra tutte le sotto-sessioni rilevate e quelle registrate in un file appositamente creato e aggiornato automaticamente di volta in volta. All'aumentare del volume dei dati archiviati, però, si è notato che la ricerca in questione stava diventando troppo lunga, in termini di tempo speso a verificare se ogni sotto-sessione trovata fosse o no già stata archiviata. Per questa ragione si è deciso di far inserire all'utente, all'avvio dello *script*, in successione, il nome della sessione e il tipo di dato, grezzo o calibrato. Ciò, ovviamente, ha reso l'intero processo molto più snello e veloce.

Il passo successivo è la ricerca, all'interno di ogni sotto-sessione della sessione inserita, di un file denominato *cube\_listing.txt*. Per motivi di comodità si è deciso di utilizzare questo file, contenente l'elenco di tutti i file-cubi appartenenti ad una determinata sotto-sessione, quale “marcatore” della presenza di una sotto-sessione. L'utilizzo di questo file nasce dalla consuetudine da parte dei ricercatori, facenti parte del team di *VIMS-V*, che si occupano di generare i file in formato PDS, di porre il file stesso all'interno di ogni sotto-sessione, in modo da permettere in qualunque momento, ad ogni membro del team che esegua il download dei dati dall'apposito sito ftp, una verifica di quanto scaricato mediante un confronto tra i file presenti sulla propria macchina e quelli elencati nel suddetto file.

Una volta completato l'elenco di tutte le sotto-sessioni presenti nella sessione indicata inizialmente, viene eseguito un ciclo di verifica il quale ha lo scopo di controllare se sia presente, all'interno di ciascuna delle sotto-sessioni stesse, il file *cube\_listing\_final.txt*. Quest'ultimo file è quello generato alla fine del processo di lettura di tutti i file-cubo presenti in ogni sotto-

sessione e contiene, per ogni file-cubo, la lista dei valori relativi alle *keyword* utilizzate per il processo di ricerca. Questo controllo, pur se non strettamente necessario apparentemente, è stato inserito nell'eventualità che una stessa sessione sia aggiornata, dopo essere stata inserita nel database, con l'aggiunta di una o più sottosessioni. Nel caso la sotto-sessione controllata non sia stata già inserita, viene inserita nella lista riportata in un file di testo di servizio, chiamato *filesCubeListingNuovi.txt*.

Al termine del ciclo di controllo lo *script* termina e viene restituito il controllo a *inserimento\_vims*, il quale, se il file *filesCubeListingNuovi.txt* esiste, ne mostra il contenuto all'utente e si mette in attesa, altrimenti pone la variabile *proseguì* uguale a "q" e termina il processo.

### 4.3.3) Preparazione dei file "cube\_listing\_final"

#### 4.3.3.1) *due\_prepara\_cube\_listing\_final*

È realizzato in *C shell* ed è lo *script* centrale di tutto il processo insieme ad un altro quasi omonimo, eccezion fatta per il suffisso di quest'ultimo che è scritto in *perl*.

Innanzitutto cerca il file di servizio generato dallo *script* precedente e lo legge ricavandone un insieme di sotto-sessioni sulle quali andare ad operare. Per ciascuna di esse, viene eseguita una verifica dell'esistenza o meno del file *cube\_listing\_final.txt*: se il file esiste lo *script* non fa nulla e passa all'elemento successivo dell'insieme letto; se non esiste comincia il processo di generazione proprio del file *cube\_listing\_final.txt* che verrà descritto qui di seguito.

La prima attività che viene eseguita è la ricerca del tipo di file che si sta inserendo nel database, grezzo o calibrato. La ricerca avviene mediante un ciclo di controllo molto semplice il quale va a considerare, per ogni sotto-sessione, tutti i file in essa contenuti. Per

ciascun file viene estratta l'estensione e controllata con le due di riferimento predefinite: "QUB" per i dati grezzi, "cub" per quelli calibrati. Non appena il sistema individua una delle due estensioni di riferimento il ciclo si interrompe e lo *script* prosegue ricordando di quale tipo di file si tratti.

Come secondo passo, viene contato il numero di file-cubo trovati al fine di informare l'utente di ciò e, subito dopo, comincia il secondo ciclo, quello che si occupa di ricavare da ogni file i valori delle *keyword* sulle quali saranno eseguite le ricerche. Per poter eseguire lo *script* in *perl* che si occupa materialmente di leggere i valori, è necessario prima creare un elenco di tutti i file-cubo presenti. Per questo è eseguito un piccolo ciclo che verifica ogni file presente nella sotto-sessione inserendo nell'elenco solo i file-cubo ed esclude eventuali altri file di testo o di servizio presenti. Solo a questo punto è avviato il secondo *script* di questa fase.

#### **4.3.3.2) `due_prepara_cube_listing_final.pl`**

È chiamato passandogli, come argomenti, il tipo di file, grezzo o calibrato, il percorso completo che individua la sotto-sessione ed il file di servizio *fileElenco.txt*, generato alla fine dello *script* precedente.

Lo *script* si incarica di eseguire un ciclo il quale legge, uno alla volta, tutti i file-cubo presenti nel file *fileElenco.txt*, andando a cercare in ciascuno tutti valori necessari per l'inserimento nella tabella del database. Per ogni file viene aperto un ciclo che legge tutto l'*header* del file stesso interrompendosi nel momento in cui legge la riga contenente il valore di fine *header*. Ogni riga letta è passata all'interno di un altro ciclo il quale si occupa di verificarne il contenuto andando a cercare al suo interno le *keyword* stabilite per *VIMS-V* ed estraendone i relativi valori nel momento in cui esse sono presenti nella riga posta in esame. Per ogni *keyword* è presente una variabile ad essa associata la quale ne conterrà il valore. Il suddetto lavoro di estrazione dei valori delle *keyword*, presenta delle difficoltà, affrontate in fase di progettazione e scrittura dello *script*,



legate alla già citata variabilità della forma dell'*header* stesso, dovuta ad aggiornamenti effettuati nel corso della missione sulla sua composizione. Il risultato di questa attività di aggiornamento è stato l'utilizzo di una serie di routine, una delle quali generica ed utilizzata per più parametri, altre specializzate per una *keyword*. I parametri *target\_name*, *mission\_phase\_name*, *instrument\_mode\_id*, *scan\_mode\_id*, *event\_start\_time*, *event\_end\_time* e *spectral\_editing\_flag* utilizzano la procedura generica chiamata *subExtractParameter*.

Tale procedura è adatta all'estrazione di un valore scritto nel formato classico del linguaggio ODL utilizzato per lo standard PDS, quello nella forma:

$$keyword\_name = "keyword\_value"$$

andando a leggere il valore presente fra virgolette ed associandolo alla variabile legata alla *keyword* cercata.

Il parametro *spectral\_editing\_flag*, in realtà, può essere presente in una forma differente e quindi richiede una doppia gestione. Esso contiene, infatti, due valori indicanti ciascuno lo stato di accensione o meno di uno dei due canali, visibile o infrarosso. Nei file-cubo generati nelle prime fasi della missione tale informazione era suddivisa in due *keyword* indicanti ciascuna lo stato di uno solo dei due canali. Successivamente si sono unificate tali informazioni in un'unica *keyword*. Ciò ha, evidentemente, reso necessario gestire le due situazioni e perciò, oltre al caso già descritto, sono state inserite due *routine* dedicate ai due singoli canali. L'utilizzo, da parte dello script, della *routine* che ricerca i due valori insieme o separatamente dipende esclusivamente da come è costruito l'*header* del file. Nel caso sia presente la *keyword* "POWER\_STATE\_FLAG" viene chiamata la prima *routine*, nel caso siano presenti le due *keyword* "VIS\_CHANNEL\_STATE\_FLAG" e "IR\_CHANNEL\_STATE\_FLAG", invece, sono utilizzate le due *routine* separate.

Un ragionamento del tutto analogo vale sia per la *keyword* EXPOSURE\_DURATION, la quale può anche essere sostituita da due *keyword* analoghe riferite ciascuna ad un canale, sia per DATA\_REGION, la quale può essere sostituita addirittura da quattro

parametri, X\_OFFSET, Z\_OFFSET, SWATH\_WIDTH e SWATH\_LENGTH. In quest'ultimo caso è necessario che lo script tenga conto di quanti dei parametri che servono per ricostruire il valore globale siano già stati letti e, alla fine, dopo averli letti tutti, ricostruisca tale valore.

Alla fine del lavoro di lettura dei valori delle *keyword* di ciascun file, lo script aggiunge al file *cube\_listing\_final.txt* una riga con i suddetti valori. Dopo aver completato l'operazione per tutti i file presenti in una sotto-sessione, prima di passare alla sotto-sessione successiva, viene aggiunta una riga al file *filesFinalNuovi.txt*, creato all'avvio dello script corrente, contenente il *path* del file appena creato, file che sarà letto al passo successivo, descritto nel paragrafo qui di seguito.

#### **4.3.4) tre\_crea\_elenco\_cube\_listing\_final**

Questo terzo passo del processo di inserimento è gestito da un piccolo semplice *script*, scritto in *C shell*, che si occupa soltanto di eseguire un ulteriore controllo affinché non vengano letti ed inseriti in tabella, al passo successivo, file già inseriti in precedenza. La necessità di continui controlli, emersa durante la descrizione dei passi precedenti, nasce dalla consapevolezza di dover, nel corso degli anni, gestire una quantità notevole di file, come già dimostrato anche in precedenza, unita all'impossibilità, da parte di chi gestisce l'archivio, di memorizzare tutte le sessioni e sotto-sessioni che sono state inserite nonché gli aggiornamenti effettuati. Proprio una considerazione di questo tipo ha portato alla creazione di una serie di controlli sicuramente ridondanti ma utili per prevenire eventuali dimenticanze o errori durante le fasi di inserimento dei dati.

Lo *script* in questione, come prima attività, esegue una lettura del file *filesFinalNuovi.txt* per memorizzare in un *array* tutti i nuovi file segnalati come da leggere per eseguire l'inserimento dei dati. Successivamente legge un file chiamato *old\_vims.txt*. Tale file è un vero e proprio archivio, in formato ASCII, di tutte le sotto-

sessioni inserite, continuamente aggiornato durante ogni fase di inserimento dei dati, alla fine proprio dello script corrente.

Ogni file dell'array viene confrontato con tutti quelli presenti nel file *old\_vims.txt* e, se non presente in esso, viene scritto all'interno di un terzo file, *vims.txt*.

### 4.3.5) inserimentoUpdateVims.php

Questo *script*, come si deduce dal nome, è scritto in *php* ed è quello che si occupa dell'inserimento delle sessioni, con le relative sotto-sessioni, nel database e di un eventuale aggiornamento, qualora fossero resi disponibili dei file-cubo, appartenenti ad una certa sessione, in un momento successivo a quello nel quale si è effettuato l'inserimento della stessa nel database.

La prima attività svolta è la lettura del file *vims.lsm*, presente nella cartella ove risiedono gli *script* di inserimento dei dati, contenente i nomi dei parametri selezionati per le ricerche. La scelta di utilizzare un file nel quale tenere la lista dei suddetti parametri anziché scriverli direttamente all'interno dello *script* ha origine semplicemente dalla facilità con la quale si può modificare tale file senza andare a toccare lo *script* stesso, il quale, inoltre, fin dall'inizio è stato scritto per essere indipendente dal numero dei parametri oltre che dalla loro scelta. In tal modo si ottiene una procedura di inserimento che non risente di eventuali modifiche nella scelta delle *keyword* selezionate o del loro numero.

Immediatamente dopo aver letto le *keyword* lo viene aperto il file *vims.txt* per memorizzare quali siano i file *cube\_listing\_final.txt* nei quali trovare i nomi dei file-cubo ed i relativi valori delle *keyword* da inserire nel database.

Parte, quindi un ciclo nel quale ciascuno dei suddetti file di dati viene aperto. Innanzitutto viene smembrato il percorso del file *cube\_listing\_final.txt* in maniera da ricavarne il nome della sessione, che sarà inserita nel campo OBS\_SESSION, il nome della sotto-sessione, scritto nella colonna relativa al PATH, e il DATATYPE, per

l'omonimo campo. Successivamente il file stesso viene aperto ed è eseguito un ciclo durante il quale vengono lette tutte le righe, una ad ogni passo del ciclo stesso, e, durante tale passo, possono essere eseguite due procedure alternative, a seconda che si tratti di una procedura di aggiornamento dei dati riguardanti una sotto-sessione già inserita in precedenza o di inserimento dei dati di una nuova sotto-sessione.

Nel primo caso avviene un controllo nel database per verificare se il file-cubo sia già stato inserito in precedenza. Ciò avviene, alla luce di quanto detto in precedenza, per sopperire ad eventuali sviste o errori nella fase di definizione della procedura di aggiornamento. Se il file-cubo già esiste tra quelli nel DB è ignorato e si passa al successivo, altrimenti viene chiamata la procedura di inserimento, descritta brevemente più avanti.

Nel secondo caso, al contrario, si presuppone che tutto si stia svolgendo correttamente e si chiama direttamente la procedura di inserimento dei dati, la quale, sostanzialmente, si occupa di separare i valori delle *keyword* contenuti all'interno della riga stessa e prepara la *query* di inserimento dei suddetti valori nel database. Durante la preparazione di tale *query* viene convertita l'indicazione temporale del momento di inizio acquisizione dal formato presente nel file-cubo, nella forma *aaaa-ggg*, essendo *ggg* il numero progressivo del giorno dell'anno, alla forma "*aaaa-mm-gg*", dove "*aaaa*" è l'anno, "*mm*" il mese e "*gg*" il giorno.

Un riferimento a parte merita l'ultimo dei campi che vengono compilati prima di eseguire l'inserimento vero e proprio. Tale campo è quello che indica se il file-cubo al quale è associato è pubblico o privato. I file, appena inseriti, e per un periodo di tempo di un anno, sono privati; per poterli visualizzare tra i risultati delle ricerche e scaricare gli utenti devono conoscere la *username* e la *password* necessari per poter accedere all'area riservata. Tutto ciò nasce da una precisa scelta delle agenzie che garantiscono ai membri dei team che gestiscono gli strumenti, i quali hanno investito notevoli risorse economiche nonché alcuni anni di lavoro per progettare e realizzare gli strumenti stessi, un diritto di precedenza nell'accesso e

nell'analisi dei dati acquisiti durante lo svolgimento della missione. Alla fine di tale periodo, chiamato appunto *proprietary period*, i dati divengono pubblici e sono accessibili da tutta la comunità scientifica.

Alla fine di ciascuno dei cicli riguardanti uno dei file *cube\_listing\_final.txt* viene segnalato all'utente il numero dei file-cubo inseriti, ad indicare che tutto si è svolto correttamente.



## Capitolo 5

### VIMS\_Preview\_Creation

Nei capitoli precedenti si è accennato alla possibilità di visualizzare il risultato di una ricerca mediante immagini di *preview* che rendano conto di come il *target* selezionato sia stato inquadrato. Questa possibilità è data dalla particolare natura di *VIMS* il quale è in grado di produrre spettri ed immagini spaziali durante la medesima acquisizione.

Allo scopo di produrre tali immagini è stato sviluppato questo software, il quale, a partire da ogni file-cubo, genera tre immagini: due immagini spaziali, una nel visibile ed una nell'infrarosso, ed una contenente una serie di profili di riga, relativi ad un certo numero di *lines*, tutti generati prendendo in considerazione il *sample* col maggior segnale. In altre parole, considerata la matrice di punti che forma uno spettro acquisito, si valuta, per un certo numero di spettri, la riga che contiene il valore maggiore per il segnale e, da ciascuno, si ricava il profilo.

Questo programma è stato realizzato utilizzando il linguaggio IDL, sviluppato dalla “ITT Visual Information Solutions”, nella sua versione per Windows, versione disponibile in ASDC. Ciò ha comportato una piccola complicazione dovuta al fatto che le immagini dovevano essere generate su una macchina dotata di Windows per poi essere trasferite sul server che, come detto, utilizza come sistema operativo una delle distribuzioni Linux.

In questo capitolo verrà descritto il suddetto programma, mostrandone le caratteristiche, derivanti dalle consuetudini nello

scrivere codice tipiche di chi utilizza questo linguaggio, e spiegando, passo per passo, il processo di funzionamento dello stesso.

## 5.1) I criteri per la creazione delle immagini

Nel momento in cui si è deciso di generare delle immagini RGB a partire dalle immagini monocromatiche presenti nei cubi, la prima decisione che si è dovuta prendere è stata quella riguardante le bande, tra quelle disponibili, da utilizzare.

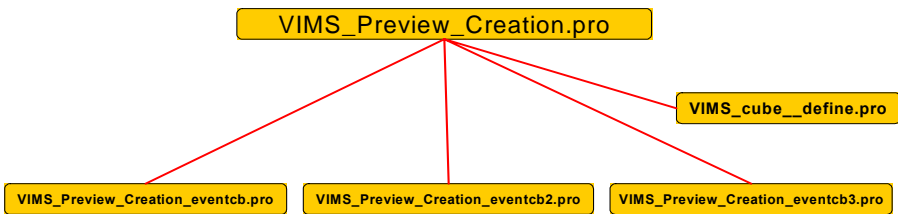
Riguardo al canale visibile si è utilizzato il sistema fotometrico standard, definito da Johnson e Morgan nella prima metà degli anni 50 del secolo scorso (Johnson e Morgan, 1953). Tale sistema prevede l'utilizzo di tre lunghezze d'onda ben definite, una nel blu, una nel verde, una nel rosso. Proprio l'utilizzo di queste tre  $\lambda$  è il motivo per il quale le immagini generate seguendo questo criterio sono chiamate RGB. Le suddette lunghezze d'onda sono le seguenti: 440 nm per il blu, 550 nm per il verde e 700 nm per il rosso.

Per il canale infrarosso non esistono standard di riferimento per la creazione di immagini RGB. Per tale motivo si è utilizzata l'esperienza accumulata osservando alcuni cubi, nonché la consulenza scientifica del team di VIMS-V per definire tre lunghezze d'onda che fossero adatte a permettere la generazione di buone immagini in particolar modo Saturno. Tali lunghezze d'onda sono state scelte evitando le zone di assorbimento di sostanze presenti nella atmosfera di Saturno stesso, in particolar modo il metano. Le bande selezionate sono, pertanto, state le seguenti: per il blu la banda 137 ( $\lambda = 1519$  nm), per il verde la banda 197 ( $\lambda = 2512$  nm) e per il rosso la banda 317 ( $\lambda = 4511$  nm).



## 5.2) La struttura del programma

La struttura del codice con il quale è realizzato il programma è quella tipica utilizzata per la realizzazione di software in IDL. Il programma è composto di cinque moduli, uno dei quali è il principale e richiama tutti gli altri. Il suddetto modulo principale si chiama `VIMS_Preview_Creation.pro`, ed è il file che viene aperto quando si vuole eseguire il programma. Come sempre accade, quando si usa IDL, per eseguire un programma suddiviso in più



**Figura 5. 1 Schema dei moduli del programma di creazione delle immagini di preview.**

moduli è necessario, come primo atto, aprire il file principale e, successivamente, compilarlo. All'interno del suddetto file viene inserita una istruzione la quale fa in modo che, all'atto della compilazione di tale file siano compilati anche gli altri componenti.

Come si può vedere dalla figura, quasi tutti i moduli che formano il resto del programma hanno il nome del file principale al quale è aggiunto il suffisso “\_eventcbn” (d’ora in poi ci si riferirà a tali file utilizzando solo il suffisso, visto che la prima parte del nome è la stessa per tutti), essendo *n* un numero progressivo. Ognuno di questi file contiene un set di routine e istruzioni che, tutte insieme, definiscono un insieme particolare di funzioni omogenee. L’ultimo file, “`VIMS_cube__define.pro`” è un file di natura leggermente differente dagli altri. Esso, infatti, rappresenta l’oggetto cubo e

definisce il metodo di lettura dei file di dati e contiene i metodi per restituire alle funzioni i parametri del cubo stesso.

Nei prossimi paragrafi saranno descritti, uno per uno, tutti i moduli che formano il programma e, per ciascuno di essi, verranno descritte le funzioni contenute. Poiché ogni modulo contiene un insieme di funzioni omogenee, utilizzate in un certo momento del processo di creazione delle immagini, la descrizione sarà fatta seguendo il funzionamento del processo di creazione stesso.

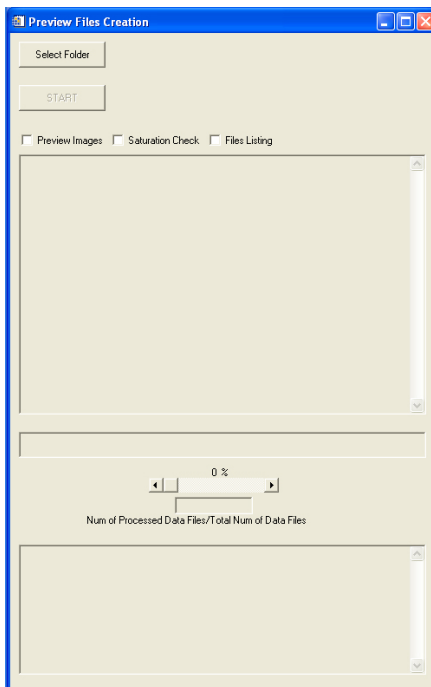
### **5.2.1) VIMS\_Preview\_Creation.pro**

Il modulo che da il nome a tutto il programma è utilizzato, fondamentalmente, per tre scopi ben distinti:

- i) creazione dell'interfaccia grafica del programma;
- ii) definizione delle “strutture” contenenti le variabili ed i riferimenti agli oggetti utilizzati durante tutto l'arco del processo di creazione delle immagini di *preview*;
- iii) smistamento degli eventi generati dalla pressione dei tasti presenti sull'interfaccia grafica.

La funzione descritta al punto 3 è la più semplice: essa si limita, infatti, attraverso il tipico stile-IDL ad intercettare tutti gli eventi generati dalla pressione di un qualsiasi pulsante ed affidarne la gestione ad una apposita procedura, presente nel file “\_eventcb1”.

Il grosso delle istruzioni presenti nel modulo principale riguarda, però, la creazione dell'interfaccia grafica, mostrata in Fig.5.2. Tale interfaccia è molto semplice poiché la funzione che ha è sostanzialmente quella di consentire all'utente di selezionare una cartella contenente dei file di dati ed eseguire la procedura di creazione delle immagini, mostrando all'utente, in ogni momento, quale sia il file elaborato e a che punto si trovi il processo nel suo insieme.



**Figura 5. 2** Interfaccia grafica del programma.

Il pulsante più in alto, “Select Folder”, apre una finestra di dialogo che consente all’utente di scorrere il contenuto del disco rigido alla ricerca della cartella contenente i file di dati. Una volta selezionata la cartella di lavoro, che può anche contenere delle sottocartelle, il programma, di *default*, seleziona tutte e tre le opzioni possibili: “*Preview Images*”, “*Saturation Check*”, “*File Listing*”.

Le suddette opzioni rappresentano le tre operazioni che il programma è in grado di eseguire: la prima è quella fondamentale per la quale è stato sviluppato il programma e non necessita di spiegazione. La seconda funzione, non prevista originariamente, è stata aggiunta in un secondo momento per fornire agli utenti alcune informazioni supplementari nel momento in cui, al termine di una ricerca, osservando una delle tabelle possibili contenenti i risultati della ricerca stessa, questi selezionano un file e aprono la pagina ad

esso dedicata. Per essere più precisi tale funzione ha lo scopo di verificare, durante la generazione delle immagini contenenti i profili di riga relativi alle differenti *lines*, la presenza di saturazione del segnale all'interno dei cubi. L'intervallo di valori del segnale, in *digital number*, utili per le misurazioni va, all'incirca, da 0 a 4095. Tale soglia superiore indica che la CCD di VIMS-V è andata in saturazione durante la misura. Il verificarsi di questa condizione in una determinata zona di un profilo è un elemento negativo in quanto rende indistinguibili le caratteristiche dello spettro in tale intervallo di lunghezze d'onda. Si è quindi ritenuto utile offrire agli utenti dell'archivio on-line informazioni relative alla presenza o meno di pixel saturati all'interno di ciascuna delle *line* che formano un cubo. L'eventuale presenza di pixel saturi viene registrata in un apposito file di testo che viene trasferito sul server insieme alle immagini. Questo file sarà poi letto al momento della generazione della pagina relativa ad un file selezionato tra quelli che soddisfano il criterio di ricerca impostato. Tutto ciò sarà spiegato nel prossimo capitolo.

La terza funzione ha lo scopo di generare un file di testo contenente l'elenco di tutti i file dei quali si sono generate le immagini di *preview*. La presenza di questa funzione è giustificata dalla possibilità, accennata in precedenza, di dover aggiornare il contenuto di una sessione inserendo dei file originariamente non presenti. Questa possibilità si è verificata più volte, nel corso dello svolgimento del lavoro. Per evitare di generare più volte le immagini a partire da un certo file-cubo, quindi, si è pensato di generare la suddetta lista. Al momento di aprire un file per generare le immagini, il programma controlla che tale file non sia già stato sottoposto al processo. Se risulta presente nella lista il suddetto file non viene letto e, di conseguenza, non vengono generate le immagini. Ciò velocizza moltissimo l'esecuzione del programma nel momento in cui si esegue la generazione delle immagini in seguito ad un aggiornamento dei file-cubo di una determinata sessione.

Al di sotto delle tre *checkbox* relative alle opzioni descritte in precedenza, è posta una grande finestra di testo nella quale è visualizzato l'elenco delle cartelle sulle quali il programma ha eseguito il processo di creazione delle immagini. Più in basso si trova

una *progressbar* che indica la percentuale di lavoro svolto all'interno della cartella corrente, con in più l'indicazione del numero di file processati e del loro numero totale. Infine, ancora più in basso un'altra finestra di testo che riporta i nomi di eventuali file per i quali si è verificato un errore o non è possibile procedere alla creazione delle immagini di *preview*. Il tasto "START", infine, serve per dare l'avvio al processo.

Un'ulteriore funzione di questo modulo, come accennato in precedenza, è quella di definire le "strutture" che sono utilizzate durante tutto l'arco del processamento dei file. Tali strutture sono descrivibili come un insieme di variabili di natura anche diversa fra loro, tutte raccolte in un unico elemento che permette di gestirle in maniera comoda. All'interno di una struttura possono essere presenti variabili numeriche, stringhe ma anche puntatori ad oggetti o ad altre entità che fanno parte del linguaggio. L'utilizzo, all'interno di un programma, di questo tipo di strumento semplifica enormemente la gestione delle variabili rispetto all'altro metodo possibile, cioè l'utilizzo di blocchi di variabili globali che vanno riportati all'inizio di ogni procedura nella quale anche solo una di esse è utilizzata. L'utilizzo delle strutture, inoltre, consente di gestire in maniera molto semplice anche le variabili associate ai "widget", ovvero, gli oggetti grafici utilizzati per costruire le interfacce grafiche. La gestione effettuata da IDL di tali entità è tale da dover assegnare ad ogni struttura definita un nome univoco. Ciò perché esse sono contenute in un'unica area di memoria e, in caso di due programmi che lavorino contemporaneamente, si potrebbero creare degli errori con valori destinati ad una variabile appartenente ad una certa struttura assegnati ad un'altra. La struttura definita per il programma qui descritto è stata nominata "structPC", e così verrà identificata nel prosieguo del capitolo.

### 5.2.2) VIMS\_Preview\_Creation\_eventcb1.pro

Questo modulo è il più semplice. Il suo unico scopo è quello di far eseguire al programma i compiti associati alla pressione dei

vari tasti o *checkbox* presenti sull'interfaccia grafica. Poiché i tasti presenti non sono molti e poiché questo modulo, come già accennato, non si occupa di eseguire direttamente i compiti ma solo di lanciare i giusti processi, è abbastanza corto.

La gestione degli eventi utilizzata è quella tipica di IDL ed è la seguente: al momento della generazione di un evento qualsiasi, sia esso la pressione di un tasto o il passaggio del mouse su un'immagine, un gestore si occupa di capire quale sia la tipologia di evento, distinguendo, per esempio, tra uno dei due casi appena citati. Tale evento viene poi passato al gestore di tutti gli eventi della categoria in questione, per esempio il gestore della pressione di un tasto. Quest'ultimo riconosce quale dei tasti presenti sull'interfaccia grafica è stato premuto e avvia l'esecuzione della *routine* ad esso associata. Nel modulo in esame i casi da gestire sono i seguenti:

- i) tasto "Select Folder": in questo caso il modulo lancia la *routine* "selectRoutine", passando alla *routine* stessa, come argomento, la struttura che contiene la variabile alla quale sarà associato il nome del file da leggere;
- ii) tasto "START": il gestore controlla che almeno una delle tre opzioni descritte in precedenza sia selezionata e, successivamente, avvia la *routine* "startRoutine", la quale avvia il processo di generazione delle immagini di *preview*. In caso contrario da un messaggio di errore all'utente.
- iii) Gestori delle *checkbox* di selezione/deselezione delle possibili opzioni: per ognuna di queste il programma imposta una apposita variabile utilizzata durante il processo principale, al momento in cui deve decidere quali delle possibili attività vanno svolte.
- iv) Tasto "QUIT": il programma, prima di chiudersi, libera la memoria della macchina eliminando tutti gli oggetti caricati in memoria e dereferenziando,

ovvero indicando come non più utilizzati, tutti i puntatori.

### 5.2.3) VIMS\_Preview\_Creation\_eventcb2.pro

È il vero cuore di tutto il programma: in questo modulo, infatti, sono contenute tutte le procedure che vengono richiamate durante la creazione delle immagini di *preview*. Tali procedure verranno ora descritte nell'ordine nel quale sono eseguite.

Prima di illustrare le suddette procedure, però, è necessario illustrare che all'interno di questo modulo sono contenute le semplici procedure che impostano le variabili che indicano al programma quali delle opzioni, già menzionate in precedenza, sono state selezionate. Tali procedure si limitano, all'avvio del processo, a porre a 0 le variabili relative alle opzioni non selezionate ed a 1 quelle relative alle opzioni selezionate. Il controllo dello stato di selezione di una opzione è effettuato verificando una particolare proprietà dell'oggetto *checkbox* ad essa associato.

Oltre a ciò è posta qui anche la *routine* "selectRoutine" di selezione della cartella che contiene i dati a partire dai quali si vogliono generare le immagini. La pressione del tasto "Select Folder" è associata alla istruzione "DIALOG\_PICKFILE" la quale causa la visualizzazione della finestra di Windows di selezione di una cartella. Tale cartella viene, una volta scelta, memorizzata in una variabile e utilizzata in una fase successiva del processo.

#### 5.2.3.1) L'avvio della procedura

La procedura che dà il via al processo principale è chiamata "startRoutine". La prima azione svolta è l'acquisizione dei valori delle tre variabili relative alle tre opzioni più volte menzionate. Lo svolgimento del processo di creazione delle immagini, infatti, dipende in maniera critica da quali di queste siano selezionate.

Una volta completata tale operazione viene recuperata la variabile che contiene la cartella selezionata in precedenza e, al suo interno vengono cercati tutti i file “cube\_listing.txt”, utilizzati quali marcatori della presenza di una sotto-sessione di dati. Tutti i file trovati, con il relativo percorso, vengono memorizzati in un *array*.

Dopo questa operazione, se è stata selezionata l'opzione di creazione delle immagini di preview, viene creata una finestra di 750 pixel di larghezza e 400 di altezza, la quale sarà utilizzata per rappresentare sullo schermo i profili di riga generati a partire da ciascun file, profili che poi saranno salvati in un file immagine.

Subito dopo, parte un ciclo su tutti gli elementi dell'*array* ricavato durante la ricerca dei file “cube\_listing.txt”. Durante lo svolgimento di tale ciclo avverrà la creazione delle immagini, come sarà descritto nel prosieguo.

All'interno del ciclo si possono riconoscere due fasi ben distinte: nella prima, se è stata selezionata l'opzione di creazione delle immagini di *preview* o quella di controllo della saturazione dei pixel, viene chiamata la procedura “folderStructureBuildUp” la quale riceve, ad ogni passo del ciclo, come argomento, il percorso di uno dei file “cube\_listing.txt”; nella seconda fase, se è attiva la terza delle opzioni descritte, quella che ha lo scopo di generare un file di testo contenente l'elenco di tutti i file dei quali si sono generate le immagini di *preview*, e contemporaneamente le altre due sono disattive, viene chiamata la procedura “setQubFilePathArr”. Quest'ultima viene anche utilizzata nella prima fase. La differenza fra i due casi è che nel secondo caso il programma si limiterà a verificare quali file immagine siano stati già generati, mentre nel primo si occupa anche di crearli.

### **5.2.3.2) Costruzione delle cartelle per le immagini**

L'avvio della procedura “folderStructureBuildUp” segna l'inizio del processo di creazione delle immagini. Innanzitutto viene recuperato il percorso di uno dei file “cube\_listing.txt”, contenuto in



una stringa memorizzata nella struttura “structPC”, e, subito dopo, viene chiamata la procedura “setQubFilePathArr”, la quale ha un triplice scopo: il primo è quello di ricavare il percorso di lettura dei file dati, a partire dalla suddetta stringa; il secondo è ricavare il numero di file dati presenti nella cartella; il terzo è stabilire quale tipo di dati sia contenuto nella cartella in esame, isolando l’estensione dei file-cubo. Come detto in precedenza, infatti, i file grezzi hanno estensione “QUB”, mentre i file calibrati hanno estensione “cub”.

Terminata l’esecuzione di questa *routine* il controllo ritorna alla precedente che, in assenza di errori, chiama la procedura “folderStructureBuildUpRoutine”. Essa ha lo scopo di suddividere il percorso del file che le viene passato come argomento in maniera tale da separare i vari componenti: la parte comune, la sessione, la sotto-sessione e l’indicatore del *despiking*, qualora si tratti di file calibrati. Tale operazione è effettuata mediante una scomposizione della stringa ed una ricerca di particolari caratteri o stringhe, contenute nel percorso, le quali si dimostrano utili per distinguere i vari componenti. Nel momento della scrittura del codice che si occupa di questa attività, ovviamente, si è tenuto conto della conoscenza generale del percorso, il quale ha la seguente struttura:

percorso\_base + “\” sessione + ”\” + “sotto-  
sessione + “\” + ind-despiking + “\” + nome\_file

Immediatamente dopo aver riconosciuto tutte le componenti del percorso di lettura la stessa routine si occupa di costruire la stringa contenente il percorso di scrittura per i file immagine. Tale percorso ha una struttura analoga a quella mostrata prima, con la differenza che nel campo sessione alla sotto-stringa “PDS”, presente nel campo di lettura, è sostituita la parola “PREVIEW”. Ciò fa sì che, nel momento in cui si vanno a scrivere i file immagine, si crei una struttura di cartelle simile e parallela a quella contenente i file cubo. Tutta la struttura di cartelle creata, a partire dalla sessione in giù, viene poi trasferita sul server, sotto la cartella “PREVIEW”

Il motivo della scelta di una simile soluzione sta nella facilità con la quale, una volta che l'utente ha selezionato un particolare file e vuole visualizzare la pagina ad esso dedicato, si possono ricercare, a partire dal percorso del file cubo, i percorsi dei file immagine ad esso associati. Bastano, infatti, poche semplici sostituzioni di stringhe, a partire dal suddetto percorso, per ricavare i *path* delle immagini.

Terminato questo compito il controllo ritorna alla procedura "folderStructureBuildUp", dalla quale la *routine* corrente era stata chiamata, la quale prova a creare il percorso di cartelle definito nel *path* di scrittura definito in precedenza. Poiché tale *path* è definito a partire da quello di lettura è possibile che non sia possibile crearlo. Nel caso, infatti, si stiano leggendo dati da DVD l'impossibilità di scrivere è evidente. Viene, in questo caso, gestito l'errore mediante la richiesta di definire una nuova locazione dalla quale partire per costruire il percorso di scrittura. Terminata questa operazione anche questa procedura restituisce il controllo alla *routine* dalla quale era stata invocata, cioè, "startRoutine".

### **5.2.3.3) Creazione dei file di controllo**

A questo punto, eseguita la prima parte del lavoro, ovvero la preparazione delle cartelle dentro le quali sistemare le immagini, ha inizio la seconda parte, cioè, la lettura di tutti i file cubo contenuti all'interno del *path* di lettura. A tale scopo viene chiamata la *routine* "qubFileSearch".

La prima azione eseguita da questa procedura è quella di verificare l'esistenza del file "saturationInfo.txt" nel *path* di scrittura. Tale possibilità può verificarsi, chiaramente, solo qualora si sia eseguito un aggiornamento del contenuto di una sessione, mediante l'aggiunta di nuovi file ad essa appartenenti ma rilasciati in una fase successiva a quella del rilascio del resto della sessione stessa. Se tale file esiste viene aperto con l'opzione che la scrittura delle informazioni abbia inizio al termine del file stesso, nella modalità *append*; se, al contrario, il file non esiste esso viene creato.

In quest'ultimo caso viene posto all'inizio del file un *header* che spiega il significato della informazioni scritte nel prosieguo. Esistono, infatti, per ciascuno dei due canali dello strumento, visibile ed infrarosso, due casi possibili: saturazione ed errore. Per il visibile si ha saturazione se è presente, in ogni line, almeno un pixel con segnale pari a 4095 digital numbers (DN), mentre si ha errore se il segnale riportato supera tale valore, eventualità non possibile data la codifica a 12 bit delle letture del segnale presente sulla CCD al momento della acquisizione. Riguardo l'infrarosso, invece, la situazione è un pochino meno definita, infatti, si ha saturazione quando il segnale letto fluttua all'interno di un intervallo compreso tra i 3500 e i 3700 DN.

La seconda operazione eseguita consiste nella lettura, se esiste, del file "filesList.txt", contenuto anch'esso all'interno del *path* di scrittura. Per questo file vale, ovviamente, lo stesso discorso fatto per il precedente riguardo la sua esistenza al momento dell'esecuzione del processo qui descritto. Il suddetto file viene innanzitutto letto per memorizzare gli elementi in esso presenti. Ciò è fatto allo scopo di verificare ognuno dei file letti nei passi successivi del processo ed impedire che vengano generate nuovamente immagini già esistenti. Anche questo file viene, poi, aperto in scrittura in modalità *append* o creato a seconda che esista già o meno.

Terminate queste due operazioni preliminari si passa al ciclo sui file della cartella in esame. Tale ciclo ha inizio con la verifica che il file in esame non sia già stato processato in precedenza. Se tale evenienza non si è verificata, viene eseguita la procedura di lettura del cubo mediante il modulo `VIMS_cube__define.pro`, illustrato più avanti.

Il passo successivo, qualora sia abilitato il controllo relativo alla presenza di saturazione nel cubo, è l'esecuzione della apposita procedura, "saturationFinder". Sul file viene eseguito un ciclo di controllo di ciascuna *line*, recuperata grazie ad un opportuno metodo creato per l'oggetto cubo definito dal modulo "cube\_\_define".

Ogni *line* è una matrice bidimensionale che ha un numero di colonne pari al numero di *bands* e un numero di righe pari al numero di *samples*. Ciascuna riga della matrice viene analizzata dalla procedura “saturationCheck” la quale suddivide la riga stessa in due intervalli: gli elementi da 0 a 95 sono quelli appartenenti al canale visibile, gli altri appartengono al canale infrarosso.

Il controllo della saturazione è differente nei due casi e, quindi, la procedura di verifica opera su ciascuna parte in maniera appropriata. È necessario specificare, però, che non necessariamente i due canali operano contemporaneamente. È, infatti, possibile che sia acceso solo il canale infrarosso. Prima di procedere alla divisione della riga, quindi, è effettuato un controllo sullo stato di accensione dei canali: questa operazione è eseguita in maniera semplice recuperando dall'oggetto cubo il valore della *keyword* preposta a fornire una tale informazione. I due array così ottenuti, o l'unico array nel caso di accensione di un solo canale, sono quindi sottoposti alla duplice verifica riguardante la presenza di elementi in saturazione o di valori errati, nel senso spiegato in precedenza, del segnale.

In caso di valori di saturazione o errati viene invocata la procedura “saturationSamplesArrayUpdate” alla quale sono passati, come argomento, gli indici dei pixel che presentano problemi insieme ad un codice che indica se sia stata riscontrata saturazione o si siano rilevati pixel con valori errati del segnale.

Gli indici ricevuti vengono aggiunti ad un array, insieme all'indicazione del problema riscontrato, che sarà poi trasferito sul file “saturationInfo.txt” al termine del controllo del file cubo in esame. Alla fine del controllo di tutti i file di una determinata cartella il suddetto file ha un contenuto del tipo di quello riportato in figura 5.3.

### 5.2.3.4) La scelta della riga per i profili

Per creare i profili di riga da visualizzare è necessario, innanzitutto, selezionare la riga stessa. A tale compito è preposta la procedura “rowFinder” la quale esegue un ciclo su tutte le immagini monocromatiche presenti nel cubo. Tale operazione è possibile, come descritto in precedenza per le *line*, grazie ad un metodo

```
SATURATION
a) VIS channel: DN on at least one pixel = 4095
b) IR channel: more than 1 pixel on a single sample with
   DN value in the range [3600, 3750]
-----
ERROR
a) VIS channel: DN on at least one pixel > 4095
b) IR channel: more than 1 pixel on a single sample with
   DN value over 3750
-----
Filename: V1523690351_1.QUB
Line: 25
Samples: 23 - IR: Error
-----
Line: 26
Samples: 22 - IR: Error
-----
...

*****
Filename: V1523691239_1.QUB
Line: 19
Samples: 41 - IR: Error
         42 - IR: Error
-----
Line: 20
Samples: 38 - IR: Error
         39 - IR: Error
         40 - IR: Error
...

```

**Figura 5. 3 Esempio di contenuto parziale del file saturationInfo.txt.**

appositamente creato, il quale restituisce l'immagine specificata nella banda spettrale che gli viene passata come argomento.

Su ciascuna immagine la *routine* trova il pixel che contiene il segnale maggiore, ne individua la posizione esatta e lo registra in un array appositamente creato. Una volta eseguita questa operazione su tutte le immagini viene eseguita una elaborazione statistica per individuare quale sia, fra tutti i pixel registrati nell'array, quello che contiene il segnale più alto. Una volta selezionato tale pixel viene registrata, nella struttura "structPC" la riga di appartenenza. Tale riga sarà utilizzata in seguito per la generazione dei profili. Terminato il proprio compito, il controllo viene restituito alla procedura chiamante, la quale immediatamente avvia, una dopo l'altra, prima la procedura di creazione dei profili di riga e, successivamente, quella delle due immagini RGB, una nel visibile e una nell'infrarosso.

### **5.2.3.5) La creazione dei profili di riga**

La procedura "spectraCreation" esegue, come primo passo, un ciclo su tutte le *line* contenute nel cubo. Per ciascuna di esse estrae il profilo relativo alla riga ricavata in precedenza dalla *routine* "rowFinder", e memorizza il massimo valore di ciascun profilo in un array. Terminata questa fase si passa alla creazione di un congruo numero di profili, in maniera da fornire all'utente un'idea precisa del contenuto del file in esame. Il numero di profili che vengono generati è pari ad un decimo del numero di *line* presenti nel cubo. Osservando un certo numero di cubi, infatti, si è dedotto che, considerando un numero di *line* per cubo compreso tra le 30 e le 60, ciò portava alla creazione di un 3-6 profili: un numero sufficiente per fornire un'indicazione del contenuto del file e non troppo grande da generare un'immagine confusa, con troppi profili che si accavallano rendendo impossibile l'interpretazione. L'esperienza accumulata grazie alla suddetta osservazione dei file-cubo ha portato, inoltre, all'individuazione di due casi: il caso in cui il numero di *line* presenti nel file è minore di 10 e quello in cui è maggiore di tale valore.

Nel primo caso il programma si limita a creare un solo profilo di riga, dalla *line* 0 se il cubo contiene un solo spettro, dalla 1 in caso contrario. Nel secondo caso viene generato un primo profilo dalla *line* che ha presentato il maggiore dei valori raccolti nell'array di cui sopra. Tale operazione viene eseguita dalla procedura "firstSpectrumCreation", la quale imposta i valori limite degli assi, oltre a disegnare il primo profilo. In seguito la procedura, partendo da questa posizione, avanza ogni volta di 10 *line* e crea un altro profilo. Nel momento in cui è raggiunta la fine del cubo riparte dalla prima *line* fino a tornare alla *line* di partenza, completando il ciclo. Questo lavoro è svolto dalla procedura "oplotsProcedure".

L'immagine con i profili così generati è salvata in un file che ha lo stesso nome del file cubo con la differenza che, alla fine del *filename* stesso, è aggiunto il suffisso "\_SPE.bmp". la stessa immagine viene anche mostrata in video all'utente, utilizzando la finestra creata all'avvio del programma.

### 5.2.3.6) La creazione delle immagini RGB

Terminata la creazione dei profili viene immediatamente chiamata la procedura di generazione delle immagini RGB. La *routine* "imagesCreation" effettua un controllo sulla possibilità di eseguire tale operazione. Nel caso, infatti, il cubo contenga una sola *line* o un solo *sample* o una sola *band* non è possibile portare a termine questa fase. Qualora si presenti una tale evenienza vengono eseguite le *routine* "noPreviewRoutineVIS" e "noPreviewRoutineIR", le quali creano una immagine quadrata, di 32 pixel di lato, nella quale compare la scritta "N/A", *Not Available*, ad indicare proprio l'impossibilità di generare l'immagine. Se, come avviene normalmente, non ci sono problemi, invece, vengono eseguite le *routine* "previewRoutineVisRgb" e "previewRoutineIRRrgb" che operano, sostanzialmente, nello stesso modo. Innanzitutto vengono definite le dimensioni dell'immagine, dimensioni che saranno pari al numero di *lines* per la larghezza e al numero di *samples* per l'altezza, e il nome del file, uguale al

filename del cubo al quale è aggiunto il suffisso “\_V.jpg” per il visibile e “\_IR.jpg” per l’infrarosso.

La produzione dell’immagine RGB avviene in due fasi. La prima fase consiste nel recuperare le immagini monocromatiche alle giuste lunghezze d’onda. Tale operazione viene eseguita utilizzando le funzioni “getChannelColorImage”, dove “Channel” può assumere i valori “Vis” o “Ir”, a seconda del canale sul quale si opera, e “Color” può assumere i valori “Red”, “Green” o “Blue”. In totale si hanno, quindi, sei funzioni che restituiscono le sei immagini di base necessarie per generare le due immagini RGB desiderate. I due tripletti di immagini vengono poi inseriti ciascuno in un array di byte, array utilizzati, poi, dalla funzione “WRITE\_JPEG”, per generare, come si capisce dal nome della funzione stessa, i file con le immagini. Oltre ad essere scritte in un file le suddette immagini RGB vengono mostrate all’utente, utilizzando la funzione di IDL “COLOR\_QUAN”, la quale riceve in ingresso le tre immagini monocromatiche e restituisce proprio un’immagine RGB.

### **5.2.3.7) La fase finale**

In seguito alla generazione delle immagini, il programma esegue due operazioni che hanno il solo scopo di aggiornare l’utente sull’andamento delle attività e sulle caratteristiche del file in esame. In questa fase, infatti, vengono eseguite due *routine*: la prima è “widTextUpdate”, la seconda è “progressBarUpdate”. La prima *routine* ha lo scopo di mostrare le caratteristiche salienti del file-cubo in esame. Essa, infatti, mostra nella apposita casella di testo, descritta all’inizio, le dimensioni del cubo di dati. La seconda, invece, aggiorna la progress-bar ed il numero di file processati. L’ultima operazione è la cancellazione dalla memoria dell’oggetto cubo relativo al file appena aperto, al fine di evitare la saturazione della memoria della macchina in uso.



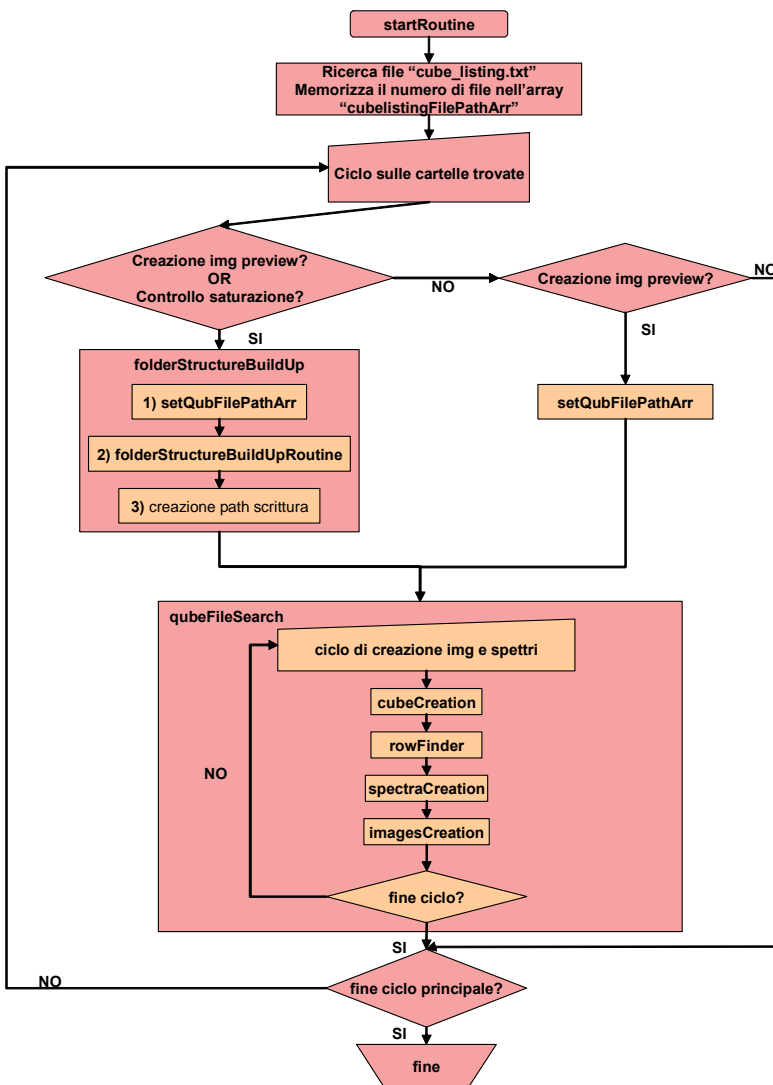


Figura 5. 4 Diagramma di flusso relativo al programma di generazione delle immagini RGB e degli spettri, VIMS\_Preview\_Creation.



## Capitolo 6

### Il sistema di ricerca dei dati

In questo capitolo verrà presentato il funzionamento del cuore del sistema di ricerche che permette agli utenti di selezionare i file secondo diversi criteri di ricerca e, al termine di esso, di visualizzare i risultati in forme differenti. La presentazione verrà effettuata seguendo lo schema di funzionamento del processo di ricerca stesso, in tutti i suoi passi dall'inizio alla fine, ovvero dalla selezione dello strumento del quale si vogliono ricercare i dati sino alla visualizzazione dei risultati ottenuti dalla consultazione del database.

Il diagramma che illustra le diverse fasi di una ricerca è mostrato in Figura 6.1. Inizialmente si ha una pagina contenente i *form* relativi a quattro differenti criteri predefiniti di selezione dei file. Una volta riempito il modulo scelto con i dati necessari, ha inizio il processo di ricerca, processo che termina con la presentazione all'utente di una tabella contenente i file che soddisfano il criterio impostato.

Nel prosieguo del capitolo verrà mostrata la struttura della pagina iniziale di ricerca (*home page*) e ne saranno descritti il funzionamento e le modalità d'uso. Verranno, inoltre, illustrate le modalità di svolgimento delle ricerche, il processo di creazione e le caratteristiche delle tabelle con i risultati. Sarà, infine, presentata la pagina dedicata ai file, la quale mostra le caratteristiche salienti dei dati contenuti all'interno dei file stessi e dalla quale è possibile avviare l'applet di *quick-look* dei dati che sarà descritta accuratamente nel prossimo capitolo.

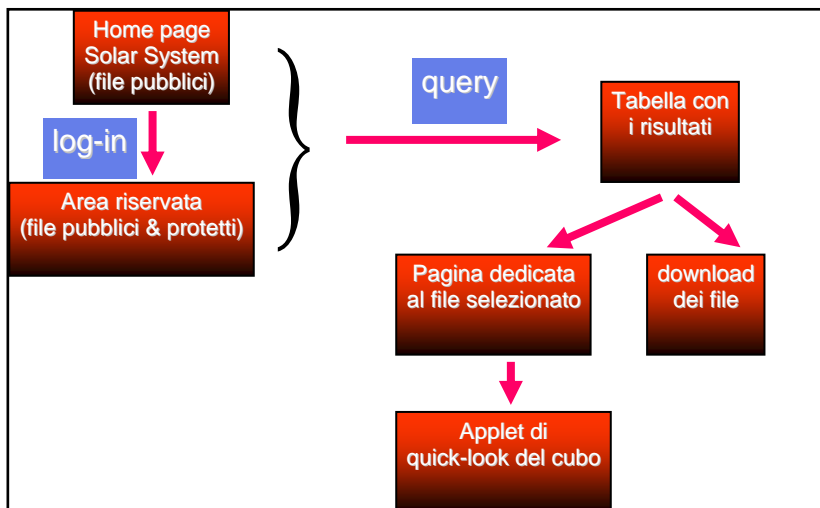


Figura 6. 1 Schema di funzionamento del processo di ricerca dei dati.

## 6.1) Le possibilità di utilizzo

Il sistema di ricerca dei dati è stato sviluppato tenendo conto delle caratteristiche dei file che vengono prodotti al termine del processo di elaborazione di quanto acquisito dallo strumento. Se da una parte, infatti, il servizio offerto non differisce, nelle sue linee generali, da un qualsiasi altro servizio di ricerca di dati basato su database, dall'altra, chiaramente, esso contiene alcune caratteristiche peculiari, come sarà spiegato più avanti, dovute allo scopo che ci si prefiggeva nel momento dell'ideazione e al contenuto dei file oggetto di ricerca.

Le possibilità di utilizzo del sistema sono schematizzate in Figura 6.2: vengono forniti quattro differenti criteri di ricerca e tre diverse tabelle risultato. Due delle tre tabelle consentono l'accesso alla già citata pagina dedicata al singolo file e, da quest'ultima, è possibile l'accesso al software di *quick-look*. L'ultima, la *Download*

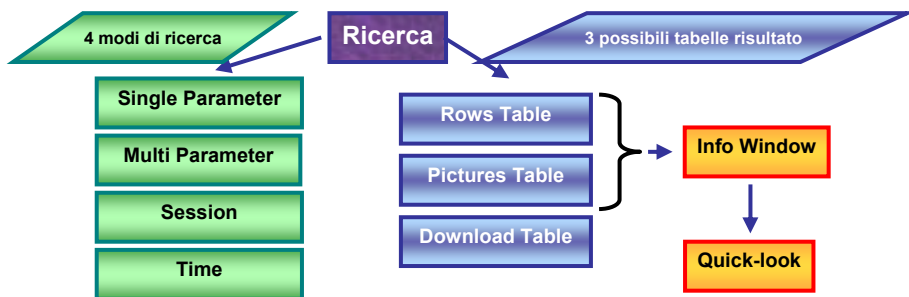


Figura 6. 2 Schema delle possibili ricerche e tabelle risultato.

*Table*, invece, serve solamente scaricare dal sito grosse quantità di dati in una volta sola.

## 6.2) I linguaggi utilizzati

Prima di proseguire con la descrizione di quanto realizzato, è necessario spendere alcune parole sui linguaggi utilizzati nella creazione delle pagine web. I due più importanti, in questa fase del lavoro, sono stati *javascript* e *php*.

Si tratta di due tipi di linguaggio molto differenti fra loro, non solo per la sintassi, ma per la loro stessa natura. Il *javascript* è un linguaggio che lavora "lato *client*": ciò significa che quando una pagina *html* viene caricata da un browser, qualunque esso sia, il codice *javascript* che ne fa parte, sia esso inserito direttamente all'interno della pagina o contenuto in un file a parte, viene caricato anch'esso ed eseguito sulla macchina dell'utente, ovvero sulla macchina *client*.

Il *php*, invece, è un linguaggio che lavora "lato *server*": esso, cioè, viene eseguito sul *server*, dove svolge tutte le operazioni che gli vengono assegnate, e produce, come risultato finale della propria attività, del codice *html* che va a far parte della pagina caricata. Il codice *php*, quindi è eseguito prima che la pagina *html* sia spedita dal server al *browser* dell'utente. Il codice *html* generato da chi si occupa

di sviluppare il sito web e quello generato dagli script *php* al momento dell'arrivo di una connessione da parte di un *client*, quindi, vanno a formare una pagina web unica che viene spedita in blocco al *client* stesso.

### 6.3) La home page

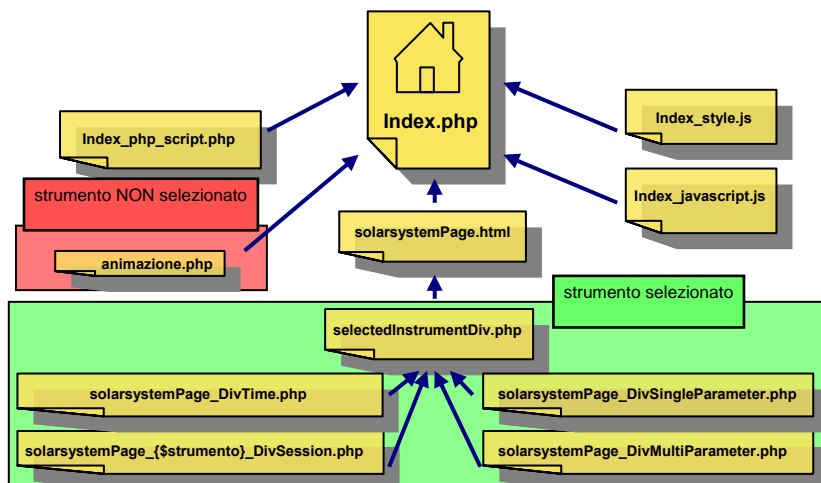


Figura 6. 3 Schema della creazione modulare della home page.

La *home page* del servizio di ricerca è stata creata in maniera modulare. Ciò significa che non è costituita da un solo file ma viene creata dal lavoro combinato di diversi script *php*, *javascript* e da un file *html* che costituisce lo scheletro di ciò che viene presentato all'utente. La struttura complessa di questa pagina è rappresentata in Figura 6.3, mentre l'aspetto della pagina, così come si presenta all'utente è, invece, rappresentato nelle due figure successive.

Quando l'utente si connette alla pagina di ricerca va a richiamare il file *index.php*, il quale è uno script contenitore, nel senso già visto nel capitolo 4. Esso, infatti, contiene al suo interno

una serie di istruzioni tra le quali sono presenti diversi richiami ad altri script *php* o *javascript* le cui funzioni sono molteplici, dovendo determinare che tipo di pagina, tra i due possibili, visualizzare. La *home page*, infatti, può presentarsi in due versioni differenti. Al momento in cui è caricata per la prima volta, infatti, l'utente si trova davanti ad una pagina sostanzialmente vuota, a parte una semplice animazione che mostra in sequenza le missioni e gli strumenti supportati, nella quale compare un menu che consente di selezionare una missione e lo strumento, relativo missione stessa, del quale si vogliono ricercare i dati, tale pagina è quella presentata qui di seguito, in Fig. 6.4.



Figura 6. 4 Home page al momento del primo caricamento, quando nessuno strumento è ancora stato selezionato.

### 6.3.1) Strumento non selezionato

Nel momento in cui la pagina viene caricata per la prima volta, questa si presenta come in Fig. 6.4. In questo caso lo script *index.php* constata che non è stato definito alcuno strumento e imposta la variabile che definisce il tipo di accesso ai dati, pubblico o privato, a “pubblico”. Tale controllo si basa sul meccanismo che

sfrutta la possibilità di caricare una pagina inviandole delle variabili con i rispettivi valori. Il *php*, al momento del caricamento delle pagine, infatti, consente di controllare, attraverso l'uso di particolari funzioni, se ad un certo numero di variabili sia stato assegnato o meno un valore. A titolo di esempio viene mostrato, nel frammento di codice 6.1 riportato qui di seguito, il controllo effettuato per determinare se sia stato selezionato uno strumento. Come si può vedere, viene effettuato il controllo sulla variabile "strumento": se è stato definito per essa un nuovo valore al momento di ricaricare la pagina tale valore viene assegnato alla apposita variabile, altrimenti viene assegnato ad essa un valore pre-determinato che indica la non avvenuta selezione.

```
if (isset($_POST[strumento])) {
    $strumento = $_POST['strumento'];
} else {
    $strumento = 'none_instrument';
}
```

**Codice 6. 1 Controllo per la verifica dell'avvenuta selezione di uno strumento da parte dell'utente.**

In questa fase, non essendo selezionato alcuno strumento, vengono saltati altri controlli, che saranno spiegati più avanti, riguardanti altre variabili. Lo script passa quindi alla fase di visualizzazione della pagina: innanzitutto scrive l'intestazione della pagina stessa e, in seguito, include, ovvero ingloba al proprio interno, lo script *index\_php\_script.php*. L'operazione definita dall'istruzione `include(nomeFile)` comporta che il codice scritto all'interno del file incluso entri a far parte del codice del file nel quale è incluso. Tale operazione è comoda, per esempio, quando uno stesso codice è usato all'interno di più script, o, come in questo caso, quando deve essere eseguito solo in determinate condizioni. All'interno di questo script è contenuto un blocco di codice, racchiuso in una funzione chiamata `menuOrSelectedInstrumentImage()`, che adatta una parte della *home page* alla situazione corrente. Nel caso in esame, non essendo selezionato uno strumento, la funzione scrive nella parte sinistra



della pagina il menu che consente la selezione della missione e del relativo strumento. Subito dopo viene incluso il file *index\_javascript.js*, che sarà illustrato più avanti, e, infine, viene incluso il file *solarsystemPage.html*, il quale contiene l'ossatura della pagina *html* che sarà visualizzata.

## 6.3.2) Strumento selezionato

### 6.3.2.1) Area pubblica

Quando, dopo aver caricato la *home page* per la prima volta l'utente seleziona uno strumento tra quelli disponibili, la pagina viene ricaricata. Durante tale operazione viene passato come argomento alla pagina stessa, mediante la tecnica descritta in precedenza, un valore ben preciso: il nome dello strumento selezionato, associato alla variabile "strumento". Ciò causa una serie di comportamenti differenti rispetto al caso illustrato in precedenza.

Durante l'esecuzione dello script *index.php* viene eseguito, innanzitutto, un controllo sul tipo di accesso. Poiché si è appena selezionato uno strumento l'accesso viene impostato a "pubblico" e non viene richiesta la verifica dell'identità dell'utente mediante *username* e *password*. Al contrario del caso precedente, poiché lo strumento è selezionato, viene eseguito un controllo su tutta una serie di variabili utilizzate per consentire all'utente di effettuare le scelte che determinano il tipo di ricerca che egli vuole condurre: tali variabili riguardano, infatti, il tipo di *form* da utilizzare, la tabella risultato da mostrare al termine della ricerca, il tipo di dato, grezzo o calibrato, e, nel caso delle ricerca per sessione, il valore del parametro *obs\_session*. Terminata questa operazione vengono caricato i moduli *index\_php\_script.php*, e, successivamente, *index\_javascript.js* e *solarsystemPage.html*, il quale, a sua volta, include il file *selectedInstrumentDiv.php*.

A differenza di quanto accadeva nel caso precedente, al momento del caricamento di *index\_php\_script.php* vengono eseguite

alcune importanti operazioni. Prima di spiegare quali siano tali operazioni è necessario effettuare una premessa: l'architettura di tutto il sistema è stata definita in maniera tale da adattarsi, in linea generale, automaticamente a tutti gli strumenti ospitati, senza dover ricorrere a moduli differenti a seconda dello strumento selezionato. Questa versatilità è stata realizzata sfruttando la facilità, descritta in precedenza, di far dialogare *php* e MySQL. Nella pratica ciò si esprime nell'utilizzo da parte del sistema di una coppia di tabelle ausiliarie dove sono contenute le informazioni necessarie a creare una pagina adatta alle ricerche per lo strumento selezionato. Tali tabelle sono chiamate, rispettivamente, *{Sstrumento}Specifiche* e *{Sstrumento}SearchParameters*, ove *{Sstrumento}* è da intendersi come una variabile qui utilizzata per indicare, in maniera generica, uno dei possibili strumenti. La seconda tabella contiene i parametri di ricerca, ovvero le *keyword*, selezionate per ognuno degli strumenti; la prima, invece, un lungo elenco di informazioni utilizzate per determinare per ciascuno strumento, ad esempio, il *path* comune a tutti i file per il loro *download*; il tipo di dati ospitati; i *form* e le tabelle risultato utilizzabili; le estensioni per i vari tipi di file, nonché lo *username* e la *password* definiti per accedere all'area riservata, contenente i dati che possono essere consultati solo dai membri del team che gestisce lo strumento.

Al momento del caricamento della *home page*, dopo la selezione dello strumento, viene effettuato dallo script un accesso al database contenente le suddette tabelle e, quindi, è eseguita la *query* che permette il recupero di tutte le informazioni necessarie. I valori letti sono poi custoditi in variabili o array utilizzati al momento opportuno durante la creazione della *home page* stessa.

Al termine della fase di lettura e memorizzazione dei valori letti nel database, viene eseguita l'operazione di inclusione dello script *index\_php\_script.php*, del quale è eseguita la funzione, già citata, *menuOrSelectedInstrumentImage()*, che, al contrario del caso precedente, quando non era stato selezionato alcuno strumento, visualizza, nella parte sinistra della *home page*, una immagine contenente il nome della missione e dello strumento selezionati e,

sopra di essa, l'immagine-link per l'accesso all'area riservata. Il risultato di tutte queste operazioni è visibile nella Fig. 6.5.

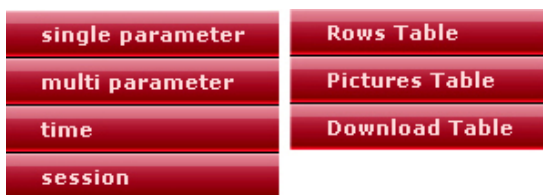


Figura 6. 5 Home page dopo la selezione dello strumento VIMS-V.

Nella parte centrale della *home page* è visualizzato uno dei *form* di ricerca, ovvero quello per le ricerche mediante singolo parametro, il quale viene visualizzato di *default*. Al di sopra sono disponibili i due menu per la selezione del modulo di ricerca e della tabella risultato da utilizzare.

Ciascuno dei *form* di ricerca utilizzabili all'interno della *home page* è contenuto in un *layer*. L'utilizzo dei *layer* è una tecnica standard nella creazione delle pagine web. In sostanza, per descrivere il funzionamento di tale tecnica, si può pensare di immaginare la pagina web, o una parte di essa, composta da più strati, uno dei quali visibile all'utente mentre gli altri sono trasparenti. In tal modo è stato possibile preparare più *form* di ricerca utilizzando sempre la stessa parte di pagina e visualizzandone solo uno alla volta. Il menu di selezione è mostrato in Fig. 6.6. Esso appare quando il mouse è portato sopra il tasto individuato dalla stringa "Search by...", visibile anche nella Fig. 6.5.

L'operazione di caricamento e visualizzazione sia dei *form* per le ricerche che dei due menu posti al di sopra merita qualche



**Figure 6.6** Menu per la selezione, rispettivamente, del form di ricerca e della tabella risultato.

chiarificazione in più. Essa, infatti, avviene sfruttando alcune delle informazioni che sono state recuperate nella fase precedente dalla tabella *{Strumento}Specifiche*, la quale, come già accennato, contiene i nomi dei *form* utilizzabili per lo strumento selezionato. Dopo aver letto tali informazioni averle registrate in un array, i valori così memorizzati sono utilizzati per la creazione del menu nella pagina web.

Come illustrato dalla Fig. 6.3, il file *solarsystemPage.html* include, al proprio interno, lo script *selectedInstrumentDiv.php*, il quale governa l'inserimento degli appropriati *form* di ricerca al momento della creazione della pagina web. Tale operazione avviene nel seguente modo: prima di tutto è creato il tasto per la comparsa del menu stesso. Ciò, ovviamente, accade sempre. Immediatamente dopo, lo script esegue un ciclo durante il quale verifica, grazie alle informazioni raccolte dalle tabelle ausiliarie, quali siano i *form* da inserire e, conseguentemente, include i file *php* con i *layer* che li contengono. Immediatamente dopo vengono aggiunti alla pagina i tasti relativi ai *form* inclusi, i quali vanno a formare il menu. Una parte di questa fase è rappresentata nel frammento di codice 6.2. Nella Fig. 6.3 è anche mostrato come il già citato script *selectedInstrumentDiv.php*, a sua volta, includa i *layer* che costituiscono i *form* veri e propri. Lo svolgimento di questa operazione è riportato nel frammento di codice 6.3.

```
for ($sf=0; $sf<count($searchFormsArray); $sf++) {
    print "<!------->\n";
    $myName = "layer".$searchFormsArray[$sf].ID";
    if ($sf == 0) {
        $myVisibility = "show";
    } else {
        $myVisibility = "hidden";
    }
    print "<DIV ID=$myName style=\"position:absolute; width:150px;
        height:30px; z-index:1; top:335px; left:455px; visibility:$myVisibility\">\n";
    print "<table width=150 height=30 border=0 cellpadding=0>\n";
    print "<tr height=30>\n";
    print "<td width=324 height=30>&nbsp;</td>\n";
    print "<td width=300 height=30><img src=\"/images/solarsystem/{$myName}.gif\"
        width=150 height=30 border=0></td>\n";
    print "<td width=250 height=30>&nbsp;</td>\n";
    print "</tr>\n";
    print "</table>\n";
    print "</DIV>\n";
}
```

**Codice 6. 2** Ciclo di inserimento dei layer che formano il menu per la selezione del form di ricerca.

```
for ($pp=0; $pp<(count($searchFormsArray)); $pp++) {
    $myString = "include(\"solarsystemPage_Div{$searchFormsArray[$pp]}.php\");";
    eval($myString);
    $myString = "writeForm".$searchFormsArray[$pp]."(";
    for ($rr=0; $rr<count($formParamArray[$pp]); $rr++) {
        $myString .= $formParamArray[$pp][$rr];
        if ($rr < (count($formParamArray[$pp])-1)) {
            $myString .= ", ";
        }
    }
    $myString .= ")";
}
```

**Codice 6. 3** Ciclo di inserimento dei layer che formano il menu di selezione del form di ricerca.

### 6.3.2.2) Area riservata

Cliccando sull'immagine-*link* per l'accesso all'area riservata viene richiesto al server di ricaricare la *home page* ma, a differenza di quanto visto in precedenza, viene eseguita una parte del codice

dello script *index.php* che fino ad ora era rimasta inerte: viene, infatti, eseguito la verifica di *username* e *password* per consentire all'utente di avere l'accesso ai dati non pubblici. Prima di ricaricare la pagina, quindi, appare all'utente una finestra nella quale deve inserire i corretti valori per i due campi. Una volta che l'utente ha dato conferma di tali valori lo script esegue un confronto tra quanto inserito dall'utente quanto è conservato nella già citata tabella *{ \$strumento } Specifiche*. Se le due terne di valori strumento, *username* e *password* coincidono la pagina è ricaricata e visualizzata correttamente, differenziandosi dalla pagina per i dati pubblici solo per l'assenza del *link* per l'ingresso all'area riservata. Nel caso le due terne non coincidano, invece, viene caricata una pagina di errore, descritta nello script *accessDenied.php*. La procedura di controllo appena descritta è mostrata nel frammento di codice 6.4

```
function controlloAccesso($instrument, $user, $pwd, $my_connection) {
    #---esecuzione della query per il recupero-----
    #---dei dati di accesso nel DB-----
    $query = "SELECT * FROM protectedAccess WHERE strumento = \"\" . $instrument . \"\"";
    $result = mysql_query($query) or die ("Username and password checkout failed!");
    #---recupera i dati dalla query-----
    while ($esitoQuery = mysql_fetch_row($result)) {
        $strumento = $esitoQuery[1];
        $utente = $esitoQuery[2];
        $password = $esitoQuery[3];
    }
    #---controllo della correttezza dei dati inseriti---
    if ( ($instrument==$strumento)&&($user==$utente)&&($pwd==$password) ) {
        $esito = TRUE;
    } else {
        $esito = FALSE;
    }
    return $esito;
}
```

**Codice 6. 4 Procedura di verifica dei dati per l'accesso all'area riservata.**

## 6.4) `index_javascript.js`

Il file `index_javascript.js` costituisce il vero gestore della *home page*. Esso infatti presiede alla gestione dei *rollover* delle immagini (ovvero l'effetto di animazione che si ha portando il mouse su una immagine); alla selezione dei *form* e delle tabelle risultato; al controllo dell'inserimento di tutti i valori necessari per l'esecuzione di una ricerca e, in generale, al corretto svolgimento di tutte le operazioni che è necessario eseguire per il corretto funzionamento della pagina stessa, la maggior parte delle quali trasparenti per l'utente.

### 6.4.1) Verifica del browser usato dall'utente

Uno dei compiti principali dello script in esame è quello di verificare quale browser sia utilizzato dall'utente. Uno degli inconvenienti dell'"universo internet", infatti, è legato alle diversità che caratterizzano i differenti browser. Ciò è dovuto in particolar modo al comportamento non omogeneo da essi tenuto di fronte ad un set di istruzioni *javascript*.

È frequente, infatti, che per ottenere uno stesso comportamento da parte di due browser differenti, in una determinata situazione, occorranò due istruzioni leggermente differenti. Nella stesura del codice *javascript* relativo ad un sito web, quindi, si deve tener conto di tali eventualità badando a gestire tutti i casi possibili mediante l'utilizzo di una funzione che identifichi il browser utilizzato e, in ogni funzione, un differente set di istruzioni per ogni possibilità.

Determinare quale programma si stia utilizzando per la consultazione dell'archivio *on-line* è, di conseguenza, il primo compito da svolgere. Questo compito è svolto dalla funzione `Is()`, una parte della quale è mostrata nel frammento di codice 6.5, nel quale viene mostrata la porzione di codice che si occupa di accertare se il programma utilizzato sia "Firefox". Al termine della esecuzione di questa funzione viene restituita al sistema una variabile contenente

l'informazione sul browser rilevato. Tale variabile viene poi utilizzata dalle altre funzioni per determinare quali blocchi di istruzioni utilizzare per svolgere i compiti loro assegnati.

```
function Is() {
    var ua, s, i;
    this.iedom = false;
    this.nsdm = false;
    this.opdom = false;
    this.ffdom = false;
    this.version = null;

    ua = navigator.userAgent.toLowerCase();

    s="firefox";
    if((i=ua.indexOf(s)) >= 0) {
        this.ffdom = true;
        this.version = 6.1;
        return;
    }
    ...
}
```

**Codice 6. 5** Porzione di codice relativo alla funzione per la identificazione del browser utilizzato dall'utente.

```
function getObject(name) {
    if (is.iedom || is.opdom) {
        browserObject = document.all[name];
    } else if (is.nsdm || is.ffdom) {
        browserObject = window;
    }
    return browserObject;
}
```

**Codice 6. 6** Codice della funzione getObject().

```
function getStyle(name) {
    if (is.iedom || is.opdom) {
        browserStyle = document.all[name].style;
    } else if (is.nsdm || is.ffdom) {
        browserStyle = document.getElementById(name).style;
    }
    return browserStyle;
}
```

**Codice 6. 7** Codice della funzione getStyle().

---



Ad utilizzare la funzionalità di riconoscimento appena descritta, sono due ulteriori funzioni, `getObject()` e `getStyle()`, le quali recuperano la variabile citata in precedenza, contenente il tipo di browser individuato, e se ne servono per la creazione di un oggetto, adatto al browser in uso, utilizzato per gestire gli eventi. Le due funzioni sono mostrate in Codice 6.6 e 6.7. Nel momento in cui una qualunque funzione deve eseguire una operazione, come prima azione richiama una delle due suddette funzioni dalle quali recupera l'oggetto da esse creato e, successivamente, se ne serve per eseguire il proprio compito.

### 6.4.2) Gestione dei *layer* e dei rispettivi menu

La gestione della selezione di un *form* di ricerca o di una tabella risultato avviene nella medesima maniera. Sarà pertanto illustrato, qui di seguito, ciò che accade nel momento in cui un utente seleziona il tipo di ricerca che desidera effettuare.

Innanzitutto il passaggio del mouse sul tasto “Search by...” genera la visualizzazione del menu di selezione. Ciò avviene in quanto al tasto è associato un codice da eseguire sia nel momento in cui il mouse è posizionato sul tasto stesso, sia in quello in cui ne esce. In entrambi i casi viene chiamata la funzione `rolloverImage(label, onoff)`, la quale, nel primo caso, genera la comparsa del menu e gestisce l'evidenziazione del tasto sul quale si trova posizionato il cursore; nel secondo caso, invece, fissa un intervallo di tempo, della durata di 300 ms, che ha inizio dopo l'uscita del mouse dall'area individuata dai tasti del menu. Al termine di tale intervallo il menu è distrutto.

In seguito alla selezione di uno dei *form* definiti per lo strumento precedentemente scelto, viene chiamato in causa l'evento `onClick()`, associato alla pressione del tasto sinistro del mouse, il quale a sua volta, chiama la funzione `selDeselLayer(label)`. Questa funzione si occupa di visualizzare il *layer* selezionato, definito dall'argomento `label`, e di rendere invisibile quello visibile in precedenza. Durante tali operazioni vengono utilizzate le funzioni

getObject() e getStyle() descritte in precedenza, come è visibile dal frammento di codice 6.8, nel quale è mostrata una porzione della funzione selDeselayer(label).

```
...
if (selectedLayer != label) {
  //----trova quale elemento dell'array 'image' contiene l'immagine--
  //----relativa al layer selezionato-----
  for (i = 0; i < image.length; i++) {
    if (image[i][0] == label) {
      posiz = findImagePosition(label);
      image2 = findImage(i);
      //----imposta come selezionata l'immagine relativa-
      //----al layer scelto-----
      document.images[posiz].src = image2;
    }
  }
  for (i = 0; i < image.length; i++) {
    if (image[i][0] == selectedLayer) {
      posiz = findImagePosition(selectedLayer);
      //----imposta come deselezionata l'immagine-----
      //----relativa al layer non scelto-----
      document.images[posiz].src = image[i][1];
    }
  }
  browserStyle = getStyle('ParameterSelection');
  browserStyle.visibility='hidden';
  selectedLayer = label;
  layerSelection(label);
...

```

**Codice 6. 8 Frammento di codice dalla funzione di selezione del layer contenente il form di ricerca scelto dall'utente.**

### **6.4.3) Gestione delle date**

Uno dei *form* di ricerca consente di definire un intervallo temporale all'interno del quale ricercare tutti i file acquisiti. Tale *form* presenta due campi contenenti due date le quali definiscono un intervallo temporale della durata di tre mesi. Tali date predefinite possono essere agevolmente impostate a valori differenti: sono, infatti, presenti, ai bordi dei due campi con le suddette date, due

pulsanti che generano un incremento ed un decremento di tre mesi della data presente nel campo al quale si riferiscono. Oltre a ciò l'utente può anche impostare manualmente i due campi. Una tale possibilità presenta, pur nella sua semplicità, due problemi: il primo riguarda le possibilità di definire un limite inferiore ed uno superiore, per le due date, che gli utenti non possano oltrepassare; il secondo consiste nel definire l'intervallo mostrato all'utente al momento del caricamento della pagina.

Per risolvere entrambi i problemi qui esposti sono state scritte alcune *routine* ad hoc. Al momento del caricamento della *home page*, tra le varie informazioni che vengono raccolte dalla tabella *{Strumento}Specifiche* ci sono, infatti, anche la data di inizio e quella di fine missione. Tali valori vengono scritti in un array che fa parte del codice javascript iniziale della pagina. Al caricamento della pagina, poi, viene eseguita una funzione, denominata *setMissionLimitDates()*, la quale ha un duplice compito: crea due oggetti data contenenti i suddetti limiti e imposta sia la data di inizio missione come limite inferiore dell'intervallo di ricerca pre-definito sia la data-limite superiore per l'intervallo stesso.

Altre due *routine* create sono quelle usate nella definizione mediante tasti dell'intervallo di ricerca, chiamate *addDate()* e *decDate()*. Esse servono, come già accennato, per aumentare o diminuire una delle due date-limite mediante gli appositi tasti. Ad ogni pressione di uno di questi tasti, prima di eseguire l'operazione di aggiornamento della data impostata, le *routine* controllano che la data che sta per essere impostata non oltrepassi le date di inizio o fine missione. Nel caso ciò avvenga viene mostrato all'utente un messaggio di avviso.

#### 6.4.4) Altre funzioni

Oltre alle *routine* citate nei paragrafi precedenti, sono presenti all'interno dello script *index\_javascript.js*, altre funzioni preposte a compiti di vario genere. Ad esempio la funzione *submitCheck()* si occupa di verificare che tutti i campi del *form*

selezionato siano compilati. Se così non è presenta un messaggio di errore differenziato a seconda del *form* utilizzato.

Un'altra *routine* utile è `resetForm()`, la quale azzerà il *form* svuotando tutti i campi testo riempiti dall'utente e riportando i parametri che hanno un valore pre-definito a tale valore.

## **6.5) Gli script di ricerca**

In questa sezione del capitolo verranno descritti gli *script* che si occupano di eseguire le ricerche impostate dall'utente nonché di chiamare, al termine di esse, lo *script* che ha il compito di generare la tabella risultato scelta inizialmente. Ciascuno è chiamato nel momento in cui l'utente clicca sul tasto *submit*, presente in fondo ad ognuno dei moduli di ricerca. A seconda del *form* selezionato viene richiamato un apposito *script php* il quale riceve come argomenti i valori di tutti i campi, oltre al nome dello strumento scelto all'avvio, presenti nel *form* stesso e li utilizza per creare la *query* i cui risultati saranno, in seguito, mostrati all'utente.

Il funzionamento generale degli *script* di ricerca è lo stesso per tutti. Ciò che li differenzia è, sostanzialmente, il numero di parametri che compongono la *query* che essi devono andare a costruire partendo da essi e, quindi, anche il sistema di creazione della *query* stessa.

Durante la fase di lettura dei valori dei parametri provenienti dal *form* vengono anche cercati i valori relativi ad alcuni campi presenti nelle tabelle risultato mostrate a seguito delle ricerche. In due delle tre possibili tabelle, infatti, è prevista la possibilità di definire il numero di risultati da mostrare in ciascuna pagina e di selezionare quale pagina, tra quelle a disposizione, si desidera vedere. Nella *Rows Table*, inoltre, è possibile selezionare quali *keyword* devono essere visualizzate nella tabella e quali no e, infine, quale *keyword* usare per ordinare la tabella. Queste possibilità offerte all'utente implicano che nel momento in cui l'utente stesso seleziona

una pagina o decide di variare il numero di risultati all'interno della stessa pagina o di ordinare la tabella secondo una determinata *keyword*, ciò che avviene è che la *query* che ha generato la tabella viene eseguita nuovamente e, al momento della creazione, vengono seguite le impostazioni definite in precedenza. Per farlo è necessario che vengano passati allo *script* alcuni parametri e array che riportano le impostazioni dell'utente riguardo la composizione della tabella, oltre a tutti quelli provenienti originariamente dal *form* di ricerca.

### 6.5.1) Ricerca per singolo parametro

Questo tipo di ricerca è eseguita dallo *script searchSingleParameter\_ss.php*. Il suo primo compito è quello di leggere i valori di tutti i parametri passati quali argomento e scriverli in variabili ed array che sono utilizzati in un secondo momento per la creazione della *query*. La lettura dei parametri avviene secondo le modalità già descritte in precedenza, utilizzando delle funzioni che il *php* mette a disposizione degli sviluppatori.

Terminata tale fase di lettura, viene chiamata la funzione *creaQuery()*, la quale si occupa, come si deduce dal nome, di costruire la *query*. La prima parte della *query* dipende dal tipo di tabella risultato selezionata: se, infatti, è stata selezionata la *Download table*, nell'istruzione *select* vengono specificati i nomi dei campi sui quali si vuole eseguire la *query* stessa. Tali campi sono definiti nella tabella *{\$strumento}Specifiche*. Per le altre tabelle risultato, invece, l'operazione di *select* avviene su tutti i campi. Secondariamente viene inserita la condizione che definisce il criterio di ricerca. Nell'inserimento di tale criterio lo *script* tiene conto di alcuni casi particolari che deve gestire, quali *obs\_session* e *filename*. Per questi due campi, infatti, sono previsti controlli particolari per limitare al massimo il numero di volte nelle quali il sistema, a causa di piccoli errori involontari, non trovi alcuna corrispondenza nel database con quanto inserito dall'utente. Alla *query*, poi, vengono aggiunte le condizioni riguardanti il *datatype*, il *despiking* del file e, cosa più importante, quella riguardante l'autorizzazione relativa

all'accesso ai dati non pubblici. Se la *query* viene eseguita dall'area riservata, infatti, vengono mostrati all'utente tutti i file che soddisfano il criterio impostato. Se, al contrario, si è eseguita la *query* dall'area pubblica viene inserita, in questa fase, una limitazione al numero di file mostrati nella tabella risultato. Alla *query*, in questo caso, infatti, viene aggiunta la condizione per la quale nel campo "ACCESS" relativo ai file sia presente il valore "PUBLIC".

Per completare la *query*, infine, sono aggiunte due altre parti: una riguarda il numero di risultati da visualizzare nella pagina e quale pagina, tra quelle disponibili, deve essere mostrata all'utente. Conoscendo queste due informazioni, infatti, è facile calcolare, tra tutte le *entry*, ovvero le righe all'interno del database che soddisfano il criterio di ricerca, quali presentare. La seconda parte aggiunta è quella relativa all'ordinamento della tabella, che può essere fatto in sede di interrogazione al database grazie ad una apposita istruzione SQL.

### **6.5.2) Ricerca con più parametri**

La ricerca mediante l'utilizzo di una combinazione di più parametri è eseguita dallo script *searchMultiParameter\_ss.php*. Come nel caso precedente la prima operazione eseguita consiste nella lettura di tutti i parametri passati allo script dal *form* dal quale esso è richiamato. Durante tale fase vengono riempiti gli array relativi ai differenti componenti della *query*, ovvero, i parametri selezionati; i relativi valori, il parametro di uguaglianza per ciascuno di essi, che può essere il simbolo "=" o la *keyword* "like"; gli operatori di congiunzione tra due condizioni, "and" o "or" e, infine, le parentesi. In aggiunta vengono, come nel caso precedente, rilevati eventuali valori dei parametri che definiscono la struttura della tabella da mostrare al termine della ricerca.

La fase successiva riguarda la creazione della *query*. Durante tale fase, analogamente al caso precedente, vengono gestiti alcuni casi particolari per i parametri *filename* e *obs\_session* e, al termine,

di questo controllo viene effettuata la creazione vera e propria della *query*. Tale operazione viene portata a termine mediante l'esecuzione di un ciclo eseguito alla fine della creazione della prima parte della *query* stessa. Durante tale ciclo vengono passate in rassegna tutte le *keyword*: per ciascuna si controlla se sia stato associato ad essa un valore e, nel caso ciò sia vero, si crea la condizione da aggiungere alla *query* utilizzando il tipo di uguaglianza impostato dall'utente. Prima e dopo la costruzione di ogni condizione si verifica se sia stata inserita una parentesi di apertura o di chiusura di un blocco di condizioni e, infine, tra l'una e l'altra viene riportato l'operatore logico selezionato dall'utente.

A completare la *query*, come nel caso precedente, seguono sia la condizione relativa all'accesso ai dati riservati, sia quelle riguardanti il formato della tabella.

### 6.5.3) Ricerca per intervallo temporale

È eseguita dallo script *searchTime\_ss.php*. La lettura dei parametri passati dal *form* è piuttosto breve in quanto l'utente può solamente inserire due date: una di inizio ed una di fine intervallo temporale.

Successivamente alla creazione della prima parte della *query*, viene inserita la condizione per delimitare la ricerca dei file all'interno dell'intervallo temporale definito dall'utente e, infine, vengono inserite le condizioni relative all'accesso ai dati riservati ed alla composizione della tabella.

### 6.5.4) Ricerca per sessione

Quest'ultimo criterio di ricerca è leggermente differente dagli altri. Esso, infatti, consente esclusivamente di ricercare dati contenuti all'interno di file acquisiti durante una determinata sessione di osservazione. La prima parte dello script intercetta i valori passati allo script stesso dal *form*, insieme, come per i casi

precedenti, ai valori per la formattazione della tabella con i risultati. La creazione della *query* è piuttosto semplice, poiché lo script si deve limitare a mettere insieme le due semplici condizioni determinate dall'appartenenza dei file ricercati alla sessione ed alla sotto-sessione determinate dall'utente. A queste condizioni, come anche per gli altri casi visti in precedenza, si aggiungono le condizioni relative alla possibilità o meno di accedere ai dati riservati e quelle relative alla suddetta formattazione della tabella.

## **6.6) Le tabelle con i risultati**

Le tre possibili tabelle con i risultati costituiscono l'ultimo passo di ogni processo di ricerca. La selezione del tipo di tabella che l'utente desidera visualizzare deve essere compiuta durante la fase di compilazione del *form* di ricerca utilizzato. Il parametro che indica la scelta effettuata viene poi passato allo script che esegue la ricerca il quale, al termine della stessa, in base al valore di tale parametro, carica la tabella corretta. La suddetta operazione di caricamento avviene nel seguente modo: lo script, durante le fasi preliminari, include il file contenente la funzione di creazione della tabella scelta dall'utente e, alla fine del processo di esecuzione della *query* chiama la funzione in esso contenuta, funzione che si occupa di scrivere la pagina *html* contenente la tabella stessa.

Come già accennato, due delle tre tabelle possono essere parzialmente personalizzate modificando alcune impostazioni di visualizzazione: la tabella *Rows Table* consente di modificare il numero di risultati visualizzati in una pagina; di selezionare la pagina, tra quelle possibili, che si vuole visualizzare; di selezionare uno o più file per il loro *download* e, infine, di scegliere quali parametri devono essere presenti all'interno della riga relativa ad un file e quali no. Essa permette, inoltre, di accedere alla pagina dedicata ad un singolo file. All'interno di tale pagina è mostrato un riassunto di tutte le informazioni riguardanti il file; sono visualizzati, all'interno di una immagine, alcuni spettri che permettono di rendersi



conto di che tipo di dati siano contenuti nel file stesso; può essere visualizzato l'*header* del file in una apposita finestra e, soprattutto, può essere caricata l'applet per la quick-look dei dati che sarà descritta nel prossimo capitolo.

La *Pictures Table* fornisce quasi tutte le possibilità offerte dalla precedente. In essa, infatti, non sono mostrati i valori delle *keyword* relative ai file e, di conseguenza, tale pagina non può essere personalizzata in tal senso. La *Download Table* non offre nessuna delle possibilità fornite dalle precedenti tabelle, in quanto nata solo per permettere di eseguire il *download* di grosse quantità di dati.

Nel seguito verrà descritto il processo di creazione delle tre suddette tabelle ed il loro utilizzo.

### 6.6.1) Rows Table

Lo script che si occupa di visualizzare all'utente la *Rows Table* è chiamato *creaTabella.php*. Esso contiene la funzione *creaTabella()*, che si occupa di eseguire tale compito, ed altre funzioni richiamate al suo interno.

La prima operazione eseguita dallo script è quella di accedere al database per consultare la tabella *{Sstrumento}Specifiche*, dalla quale ricava l'informazione riguardante quali *keyword*, tra quelle possibili, visualizzare di *default*. La lista di tali campi sarà consultata più avanti, nel corso della scrittura delle righe riguardanti i singoli file.

La seconda operazione che viene eseguita consiste nella scrittura della prima parte della pagina *html* che ospita la tabella: in particolare viene scritto tutto il codice *javascript* che governa il funzionamento della pagina. Tale codice consiste in alcune funzioni utilizzate per consentire all'utente di utilizzare le possibilità offerte dalla tabella. Sono presenti, infatti, diverse funzioni: *setFileDownload(n)* aggiunge il file selezionato alla lista dei file dei quali deve essere eseguito il *download*; *checkDownload()* verifica, prima della esecuzione del *download* stesso, se almeno un

file è stato selezionato, altrimenti restituisce un messaggio di errore. La coppia di funzioni `setFileSelectedView(n)` e `checkView()` ha uno scopo analogo ma riferito alla visualizzazione della pagina dedicata ad un file. È poi presente un'altra funzione relativa al *download* dei file, chiamata `selectDeselectAll()`, la quale seleziona, o de-seleziona, tutti i file per il successivo *download*. Due delle funzioni sono dedicate alla formattazione della tabella: `setFileOrder()` imposta la *keyword* rispetto al quale si vuole eseguire l'ordinamento della tabella, mentre `setVisualTitle()` imposta le *keyword* che l'utente vuole visualizzare e quelle che devono rimanere invisibili all'interno di una riga della tabella. Entrambe, al termine, determinano un nuovo caricamento della tabella, che viene riscritta rispettando le impostazioni stabilite dall'utente.

Al termine di queste due operazioni ha inizio la scrittura delle diverse componenti che formano la pagina vera e propria, visualizzata all'utente. Nella parte più in alto della pagina viene posta la sezione che consente di determinare il numero di righe che devono essere mostrate in ogni pagina di risultati. Qui è presente un campo nel quale inserire il valore con, accanto, un tasto che imposta la visualizzazione di tutte le righe in un'unica pagina. Sulla destra è presente il tasto per caricare la pagina nuovamente seguendo la impostazioni. Al di sotto di questa sezione è presente la serie di tasti che consente di scegliere quale pagina di risultati visualizzare tra quelle trovate dal sistema. Più in basso è posto un riquadro contenente una serie di tasti-indicatori di colore verde o rosso. Tali tasti consentono di personalizzare la tabella decidendo quali *keyword* mostrare al suo interno e quali no. I tasti verdi indicano le *keyword* attualmente mostrate, quelli rossi le *keyword* invisibili all'utente. Premendo su un tasto questo passa da un colore all'altro, indicando il cambio dello stato di visualizzazione. Una volta completata la fase di personalizzazione, premendo il tasto "Re-load page", la tabella viene ricaricata seguendo le impostazioni stabilite. La tabella vera e propria è situata più in basso. Nella parte sinistra è presente, in ogni riga, un tasto che serve per selezionare il file, eventualmente insieme ad altri, per un successivo *download*. Per scaricare i file (o i file), è

ARDC MULTIMISSION ARCHIVE SOLAR SYSTEM EXPLORATION  
Query Results

Results per page: 20 Show all results Re-load page

Select Page: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Select parameters to be shown:  Shown param  Not shown param Re-load page

disable automatic tooltip enabled

Select/Deselect All For Download Download Selected Files

	DATATYPE	FILENAME	OBS_SESSION	PATH	SPIKES	TARGET	COREITEM (S,B,L)	POWERSTATEFLAG	OBSDATE
<input type="checkbox"/>	RAW	V1356766601_2.QUB	C23_PREVIEW/	CA/	-	GANYMEDE	12,352,12	ON,ON	2000-12-29
<input type="checkbox"/>	RAW	V1356766620_2.QUB	C23_PREVIEW/	GA/	-	GANYMEDE	12,352,12	ON,ON	2000-12-29

**Figura 6. 7 Esempio di Rows Table.** Nella parte in alto è visibile la sezione per la selezione del numero di righe per pagina. Al di sotto sono presenti i tasti per la selezione della pagina da mostrare. Più in basso sono posti i tasti per la selezione dei campi da rendere visibili. In fondo è visibile la tabella vera e propria. In ogni riga, sulla sinistra, è presente un tasto per la selezione del file per il successivo download. Al di sopra della tabella è posto il tasto “Select/deselect all for download” consente di selezionare o de-selezionare tutti i file per il download. In fondo alla tabella è posto, non visibile qui, il tasto per l’avvio del *download*.

necessario premere un tasto posto in fondo alla tabella. Allo scopo di facilitare la selezione dei file è presente, inoltre, in cima ed in fondo alla tabella, un tasto per la selezione (o la de-selezione) di tutti i file della pagina. Cliccando sul nome del file, invece, si apre un piccolo menu contestuale che consente di accedere alla pagina dedicata al file o di avviare l’*applet* per la *quick-look* dei dati in esso contenuti. Durante il processo di scrittura della tabella, insieme al codice *html* che crea quanto mostrato all’utente, vengono inserite altre righe di codice utili per le operazioni di visualizzazione delle immagini di *preview*, funzionalità non ancora illustrata e spiegata qui di seguito. Al passaggio del mouse sul nome del file corrisponde l’esecuzione della funzione `showFile()`, la quale fa sì che venga mostrata una finestra semi-trasparente, presentata in Figura 6.8, contenente le

immagini di *preview* generate durante la fase di inserimento del file nel database. Tale finestra contiene, in alto, le due immagini spaziali RGB, relative ai canali visibile ed infra-rosso, e, in basso, l'immagine con gli spettri campione. Ciò è possibile grazie alle istruzioni riportate in blocchi *javascript* associati alla riga della tabella e ad un set di istruzioni posti in tre file *javascript*, disponibili in rete gratuitamente ed utilizzati dal sistema, che hanno questo tipo di visualizzazione come unico scopo. La finestra, di *default*, appare accanto al mouse ma può anche essere spostata, trascinandola in un'altra parte dello schermo. Inoltre, è possibile bloccarne la posizione mediante il piccolo tasto a forma di spillo presente in alto a sinistra della finestra stessa. Tale funzionalità può essere disattivata (ed eventualmente ri-attivata in seguito) grazie ad un tasto posto al di sotto del riquadro all'interno del quale sono presenti gli indicatori di visualizzazione delle *keyword*.

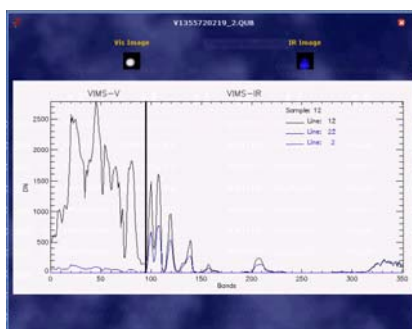


Figura 6.8 Finestra semi-trasparente, contenente le immagini di preview, visualizzata portando il mouse sopra il nome del file.

## 6.6.2) Pictures Table

Il processo di creazione di questa tabella risultato è analogo a quello della tabella descritta nel paragrafo precedente. Ciò che differenzia le due tabelle è che in questo tipo, al posto dei valori delle *keyword* relativi ai singoli file, vengono mostrate all'utente le immagini spaziali relative ai due canali, visibile ed infrarosso, generate dal programma descritto nel capitolo precedente. Ciò è

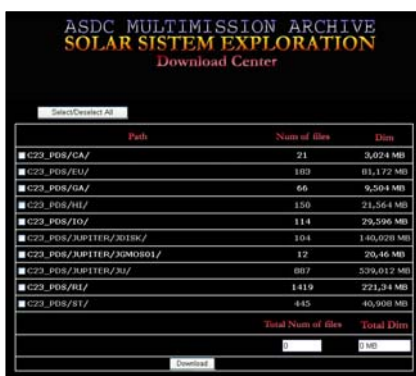
possibile grazie a qualche sostituzione di stringa nel percorso che individua un file dati, in modo da ottenere i percorsi dei corrispondenti file immagine. L'utilizzo di questo tipo di visualizzazione del risultato di una ricerca è stato pensato principalmente per fornire ai ricercatori una indicazione diretta di come il *target* prescelto fosse stato inquadrato dallo strumento durante la fase di acquisizione. Il funzionamento della *Pictures Table* è analogo a quello della *Rows Table*. Anche in questo caso, come nel precedente, portando il mouse sopra il nome del file si ha la comparsa di una finestra semi-trasparente la quale mostra l'immagine con gli spettri-campione generati in fase di creazione delle immagini di *preview*.



Figura 6. 9 Pictures Table. Nella parte più in alto sono visibili, le sezioni per l'impostazione del numero di risultati da mostrare nella singola pagina e per la selezione della pagina da mostrare. In basso è visualizzata una parte della tabella di immagini. In ogni cella, sulla sinistra, è posta l'immagine ricavata dal canale visibile e, sulla destra, l'analogica ricavata dal canale infrarosso. Sotto le immagini è presente il tasto per l'aggiunta del file alla lista di quelli da scaricare. Più in basso sono presenti le informazioni relative alla sessione e sotto-sessione di appartenenza del file stesso.

### 6.6.3) Download Table

La *Download Table*, come accennato in precedenza, è nata per consentire di effettuare il *download* di grandi quantità di dati nella maniera più agevole possibile. Il processo di creazione ed il funzionamento di questa tabella sono molto differenti da quelli relativi alle due viste in precedenza. La prima operazione eseguita dalla funzione `creaTabellaDownload()` è la scrittura di tutto il codice *javascript* necessario al funzionamento della pagina stessa.



Path	Num of files	Dim
■ C20_PDS/CA/	21	3,024 MB
■ C20_PDS/EU/	103	81,172 MB
■ C20_PDS/GA/	66	9,504 MB
■ C20_PDS/HI/	150	21,564 MB
■ C20_PDS/IO/	114	29,596 MB
■ C20_PDS/JUPITER/301BK/	104	140,020 MB
■ C20_PDS/JUPITER/30M0801/	12	20,46 MB
■ C20_PDS/JUPITER/31/	807	539,012 MB
■ C20_PDS/RI/	1419	221,24 MB
■ C20_PDS/S1/	445	40,908 MB
Total Num of files		Total Dim
0		0 MB

Figura 6. 10 Download Table. Sulla sinistra sono visibili i nomi delle sotto-sessioni contenenti file che soddisfano il criterio di ricerca impostato. Sulla destra, in ciascuna riga, sono mostrati due valori relativi alla sotto-sessione riportata nella riga stessa: il primo valore rappresenta il numero di file che appartengono a quella sotto-sessione e che soddisfano il criterio di ricerca impostato. Il secondo valore rappresenta la dimensione, in MegaByte, dell'insieme costituito dai suddetti file. In basso due campi indicano il numero totale di file selezionati e la relativa dimensione.

Tale codice, descritto più avanti, differisce completamente da quello relativo alle tabelle viste in precedenza, dal momento che deve esclusivamente gestire la selezione ed il *download* delle sotto-sessioni di file da parte dell'utente.

Quando si seleziona questa tabella il sistema ordina i risultati della ricerca in maniera da suddividere i file che soddisfano il criterio impostato in gruppi appartenenti alla stessa sotto-sessione. Ciò è

mostrato in Figura 6.10. Di ogni gruppo così individuato, vengono riportati, in ciascuna riga, il numero dei file dai quali il gruppo stesso è costituito e la relativa dimensione, espressa in MegaByte.

Al momento della scrittura della pagina *html*, per ogni file contenuto nell'array dei file che soddisfano il criterio di ricerca impostato, viene eseguita una ricerca nel database in maniera da ricostruirne il percorso completo e, una volta eseguita una tale operazione, verificarne, mediante l'utilizzo di istruzioni *C shell* eseguite dal *php*, la dimensione. Le informazioni così trovate sono poi conservate all'interno della pagina in blocchi invisibili all'utente ma utilizzabili dal *javascript* della pagina stessa. Nel momento in cui si seleziona una sotto-sessione, essa è aggiunta all'elenco delle sotto-sessioni delle quali eseguire il *download* dalla funzione `setCheckPathVal(n)`. Nello stesso istante è chiamata anche la funzione `calculateTotal(n)`, la quale restituisce sia il numero totale dei file facenti parte delle sotto-sessioni selezionate, che la relativa dimensione totale. Alle funzioni appena citate si aggiungono, inoltre, sia quella per selezionare o de-selezionare tutte le sotto-sessioni, associata al tasto "Select/Deselect All", sia, infine, la funzione `downloadCenterCheck()`, la quale controlla che almeno una sotto-sessione sia stata selezionata prima dell'avvio del processo di *download*.

Una volta avvenuta la selezione delle sotto-sessioni, l'utente può scaricare sul proprio computer tutti i file che ne fanno parte semplicemente premendo il tasto "Download". Allo scopo di abbreviare il tempo necessario all'esecuzione di tale operazione, si è deciso di far precedere l'operazione di *download* vero e proprio da una fase di creazione di un unico file-archivio compresso. Tale processo è eseguito dallo script `downloadFromDownloadCenter.php`. All'interno di tale file-archivio viene mantenuta l'informazione sul percorso di ognuno dei file. In questo modo, nel momento in cui l'archivio viene aperto, all'utente vengono restituite una serie di cartelle, corrispondenti ognuna ad una sessione, contenenti i file-dati suddivisi per sotto-sessioni, esattamente come sono conservati nella struttura originaria presente sul server.

## 6.7) La pagina di quick-look

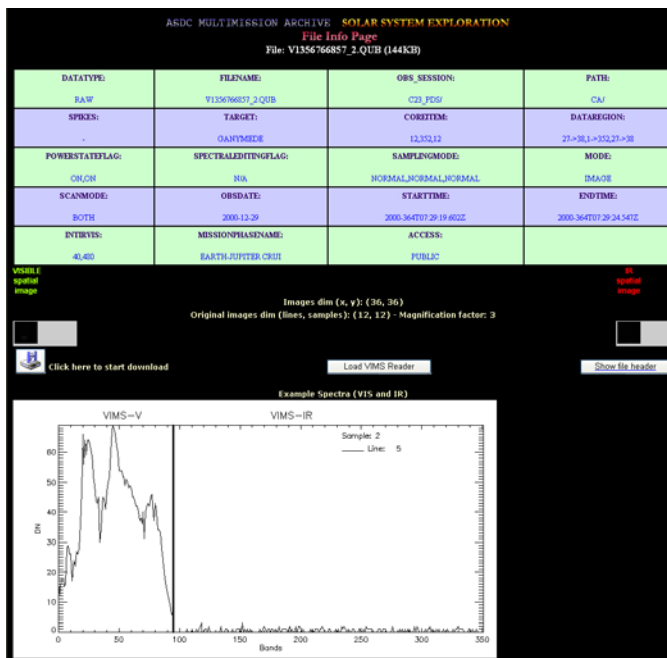


Figura 6. 51 Pagina di quick-look.

La pagina di quick-look è stata concepita per permettere ad un ricercatore interessato in particolar modo ad un file di vederne le caratteristiche salienti tutte raccolte, allo scopo di farsi un'idea precisa del tipo di dato conservato nel file stesso. Tale pagina contiene, infatti, tutti gli elementi che il sistema è in grado di fornire riguardo un determinato file: come si può vedere dalla Figura 6.11, infatti, sono mostrate sia le immagini spaziali nei due canali visibile ed infrarosso, sia gli spettri campione.

Nella parte destra della pagina, inoltre, sono ricapitolati tutti i valori delle *keyword*, relative al file, conservati nel database. A ciò si aggiunge la possibilità di visualizzare l'*header* completo del file stesso. È presente, infatti, un tasto chiamato "Show header", il quale



richiama la funzione *javascript* `urlPopUp(url)`, la quale mostra, in una finestra di *pop-up*, il file di testo individuato dalla variabile *url*. Tale file di testo è creato al momento della visualizzazione della pagina: viene, infatti, eseguito uno script *perl* chiamato *txtFile.pl* il quale riceve due argomenti: il primo è il percorso completo del file al quale la pagina è dedicata; il secondo è il nome del file di testo nel quale sarà riversata una copia dell'*header*. Tale script legge tutto l'*header* del file e lo copia, separandolo quindi dalla parte binaria, nel già citato file di testo. La funzione `urlPopUp(url)`, quando chiamata, quindi, non fa altro che inserire nella finestra da essa stessa creata il contenuto del suddetto file.

L'ultima funzione della pagina, nonché la più importante, è quella di richiamare, mediante il tasto "Load VIMS Reader", posto tra le due immagini spaziali del *target*, l'applet java che consente di visualizzare, in tempo reale, gli spettri e le immagini monocromatiche contenute all'interno del file. Tale funzionalità sarà descritta in maniera esauriente nel prossimo capitolo.



# Capitolo 7

## VIMS Reader

### 7.1) Considerazioni preliminari

Fin dal momento nel quale si è avuta l'idea di realizzare un archivio on-line di dati derivanti da strumenti a bordo di sonde volte all'esplorazione del Sistema Solare, si è stabilito che una delle caratteristiche fondamentali avrebbe dovuto essere la possibilità, offerta agli utenti, di poter avere velocemente accesso al contenuto dei file-cubo. Si è, infatti, pensato che avrebbe potuto essere molto utile a chi avrebbe eseguito ricerche, poter verificare velocemente se il contenuto di un determinato file fosse o meno adatto al tipo di lavoro portato avanti.

Chiunque abbia dimestichezza con lo standard PDS e sia abituato a lavorare con i file di dati trascritti seguendo tale standard sa che aprire uno di questi file e, a partire dal flusso binario in essi trascritto, visualizzarne il contenuto, sotto forma di spettri o immagini monocromatiche, come avviene per *VIMS*, non è affatto banale.

#### 7.1.1) La scelta di utilizzare Java

Proprio per tale motivo, si è stabilito di realizzare un'applicazione che permettesse di visualizzare spettri ed immagini derivate dal cubo selezionato. Una delle caratteristiche principali di

una tale applicazione avrebbe dovuto essere quella di lavorare sulla macchina del singolo utente, in maniera da non sovraccaricare il server di troppo lavoro, soprattutto in caso utilizzo dell'archivio on-line da parte di più utenti contemporaneamente. Oltre a ciò l'applicazione avrebbe dovuto essere adatta all'ambiente del web, in maniera da integrarsi perfettamente con l'archivio, realizzato proprio per essere utilizzato via web. Una volta messe a fuoco queste due caratteristiche la scelta del linguaggio e del tipo di applicazione è stata pressoché immediata: è ben noto, infatti, che queste sono proprio le caratteristiche delle cosiddette *applet* realizzate in Java.

Il linguaggio Java, distribuito da Sun, è un linguaggio molto potente, che permette agli sviluppatori di realizzare praticamente qualsiasi tipo di applicazione. È, inoltre, stato ideato e sviluppato proprio per avere, tra le sue finalità quelle di essere utilizzato in ambiente web. Proprio a tale scopo, infatti, gli sviluppatori di Sun mettono a disposizione di chi utilizza Java una ricca serie di librerie le quali consentono di sviluppare applicazioni in grado di utilizzare, senza problemi di sorta, la rete per gli scopi più disparati.

Un'altra caratteristica fondamentale che ha portato alla scelta di Java quale linguaggio per lo sviluppo dell'applicazione per la lettura dei dati di *VIMS* è stata la cosiddetta "portabilità" di Java. Uno dei problemi che affliggono alcuni linguaggi, infatti, è la difficoltà nel trasportare un'applicazione sviluppata all'interno di un certo ambiente (Windows o Linux in una delle sue molteplici distribuzioni o altro) in un altro ambiente.

Ciò che avviene, tipicamente, è che il codice scritto dallo sviluppatore del software, qualunque sia il linguaggio utilizzato, è trasformato da un compilatore nel "linguaggio macchina", una forma diversa facilmente comprensibile ed interpretabile dal computer, o meglio, dal sistema operativo presente sul computer stesso. Se si trasporta il codice compilato da un sistema operativo all'altro, generalmente, al momento dell'esecuzione del programma si hanno degli errori dovuti al fatto che il sistema operativo della macchina sulla quale è stato trasportato il software, non è in grado di comprendere il codice macchina scritto per essere interpretato dal

sistema operativo sotto il quale è avvenuta la compilazione. In pratica, ciò che avviene, è che le istruzioni riportate nel codice macchina sono semplicemente incomprensibili perché scritte per un sistema operativo differente.

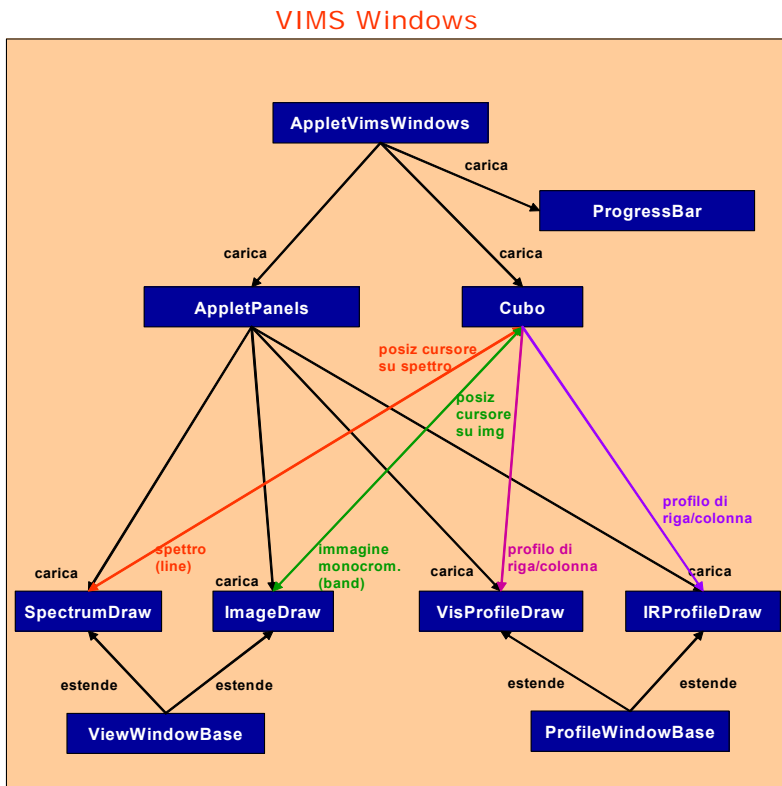
Per ovviare a tale grave limitazione, Sun ha introdotto la “Java Virtual Machine” (JVM), la quale costituisce uno strato software che si pone tra il programma ed il sistema operativo presente su un determinato computer. Sun stessa sviluppa e distribuisce versioni di JVM adatte ai differenti sistemi operativi presenti sul mercato. Ciò garantisce che le applicazioni Java siano utilizzabili in qualunque ambiente. Ciò, nel caso dell’archivio online, si traduce nella possibilità di eseguire l’*applet* da parte di qualunque utente, a prescindere dal proprio sistema operativo. Il funzionamento della JVM è spiegabile facilmente: al momento della compilazione del programma il compilatore Java produce una serie di file scritti in un linguaggio intermedio tra il codice ASCII, facilmente leggibile dal programmatore, e il codice macchina, interpretabile solo da un computer. Tale codice intermedio, detto “bytecode” è poi, al momento della esecuzione del programma stesso, interpretato dalla JVM che fornisce al sistema operativo le istruzioni nel corretto formato.

Come accennato sopra il tipo di applicazione che si è deciso di realizzare è noto col nome di *applet*. Questo tipo di applicazione è stato ideato proprio per essere caricato a partire da una pagina web: l’*applet* può presentarsi all’utente come una qualsiasi applicazione contenuta in una finestra o può essere incorporata all’interno di una pagina web. Una volta caricati i dati nella memoria del computer dell’utente, o *client*, l’applicazione è completamente indipendente dal contesto e dal *server* e sfrutta solo le risorse del *client*. Ciò garantisce che il server non sia sovraccaricato di lavoro e che il servizio rimanga, quindi, sempre efficiente.

## 7.2) La struttura del programma

Il programma **VIMS Reader** è strutturato, nel tipico stile “a oggetti” di Java, in una serie di file, ognuno dei quali con una specifica funzione. Il file *main*, quello cioè, che governa il caricamento dell'applicazione e dei dati necessari al suo funzionamento, è chiamato *AppletVimsWindows.java*. Questo file si occupa di caricare in memoria una *progress-bar* che visualizza lo stato del caricamento dell'applicazione, il cubo dati e, infine, l'interfaccia grafica dell'*applet*. La *progress-bar* è descritta dal file *ProgressBar.java*, ed è chiamata solo nella fase di caricamento del programma, ad indicare all'utente, il quale ancora non può ancora vedere l'interfaccia grafica, che l'applicazione è stata avviata. Il cubo è descritto dal file *Cubo.java*. Esso contiene le istruzioni per la lettura del cubo-dati presente nel file selezionato dall'utente nonché tutti i metodi necessari per restituire agli oggetti che ne fanno richiesta, sia i profili sia gli spettri e le immagini monocromatiche.

L'interfaccia grafica è contenuta nel file *AppletPanels.java* ed in altri file da esso richiamati. Il suddetto file, infatti, contiene la descrizione del contenitore esterno e dei pulsanti, mentre i pannelli sui quali sono visualizzati profili, spettri ed immagini monocromatiche sono descritti da classi derivanti da altri file. Le immagini monocromatiche sono caricate sul pannello descritto dal file *ImageDraw.java*; gli spettri sul pannello descritto da *SpectrumDraw.java*. I suddetti pannelli sono derivati entrambi da una medesima classe “padre” *ViewWindowsBase.java*. Il termine classe “padre” indica una classe a partire dalla quale è possibile definirne un'altra, detta classe “figlio”, dalla quale quest'ultima deriva proprietà e metodi, avendo al contempo la possibilità di aggiungerne di nuovi. Il profilo di riga nel visibile, nonché i profili di colonna, sono visualizzati sul pannello descritto dal file *VisProfileDraw.java*, mentre il profilo di riga nell'infrarosso è visualizzato sul pannello descritto dal file *IRProfileDraw.java*. Entrambi i pannelli dei profili derivano dalla classe “padre”



**Figura 7. 1** Schema delle relazioni esistenti tra i file che costituiscono il programma VIMS Windows.

*ProfileWindowBase.java*. Uno schema della struttura del programma, con tutti i file dai quali è costituito e con una indicazione schematica dei rapporti tra gli oggetti, è visualizzato in Fig.7.1.

## 7.3) Il caricamento dell'applet

Il processo di caricamento dell'*applet* è costituito essenzialmente da tre fasi: nella prima si ha la visualizzazione di una *progress-bar* la quale da un'indicazione dello stato di avanzamento del processo stesso; la seconda consiste nel caricamento in memoria del cubo di dati e la terza nella visualizzazione dell'interfaccia grafica.

### 7.3.1) La richiesta al server

Il tutto ha inizio quando l'utente seleziona il *link* sulla pagina web dedicata al singolo file o lancia l'*applet* dal menu contestuale che si apre cliccando sul nome del file sia nella *Rows Table* che nella *Pictures Table*. In quel momento si ha l'avvio dell'*applet*. Ciò consiste nell'invio di una richiesta al server il quale risponde inviando al *browser* dell'utente, senza che questi si accorga di nulla in quanto tale processo è trasparente per l'utente stesso, tutti i file che compongono il programma. Oltre a ciò il *client*, ovvero il computer dell'utente, invia al server anche il nome del file-cubo che si vuole leggere, con il relativo *path* per ritrovarlo. Non appena tali file sono stati ricevuti vengono letti dalla JVM e sono caricate in memoria tutte le classi che dai file stessi derivano. Subito dopo la JVM si occupa di individuare la classe *main* e di eseguire, tra i metodi in essa contenuti, quello denominato *init()*, il quale contiene le prime istruzioni da eseguire.

### 7.3.2) Il metodo *init()*

La prima operazione che il metodo *init()* compie è quella di separare il nome del file-cubo dal resto del suo percorso, al fine di visualizzarlo, in seguito, sulla barra principale dell'applicazione.



Eseguita questa semplice operazione viene caricata la *progress-bar*, istanziando in memoria la classe definita dal file *ProgressBar.java*. Col termine “istanziare” si intende quel processo per il quale, a partire da una classe, si ha la creazione, nella memoria dell’elaboratore, dell’oggetto che da essa deriva. In altre parole, istanziare una classe, creando a partire da essa un oggetto, è simile al creare, per esempio, un’automobile a partire dal suo progetto. Questa azione comporta la visualizzazione di una piccola finestra dotata, appunto, della suddetta *progress-bar* che scorre durante il processo di caricamento dei dati, e sulla quale sono visualizzati, via via che tale processo va avanti, messaggi esplicativi di ciò che sta avvenendo: il primo messaggio che viene visualizzato è “*Loading data file...*”, subito prima che abbia inizio il download dal server del file cubo; il secondo è “*Loading graphic interface...*”, durante il caricamento dell’interfaccia grafica.

Una volta visualizzata la *progress-bar* il programma da subito l’avvio al download del file-cubo richiesto dall’utente. Tale operazione è realizzata sfruttando le capacità di lavorare sulla rete di Java. Per essere più precisi viene sfruttato l’oggetto URL definito in una delle librerie Java. Tale oggetto consente di stabilire una connessione con un file remoto, una volta che gli siano state fornite le indicazioni relative alla macchina con la quale si vuole stabilire una connessione e sulla posizione del file che su tale macchina è situato. L’*applet*, in questa fase, stabilisce la connessione con due differenti file, infatti, oltre al file-cubo, va a leggere un file di testo posto sul server, e chiamato *VIMS\_tabella\_numcol\_lambda.txt*, il quale contiene tutte le lunghezze d’onda  $\lambda$  associate alle colonne della CCD di *VIMS-V*. Ciò avviene per fornire all’utente che andrà a visualizzare dei profili di riga, la possibilità di avere sull’asse delle ascisse, non solo il numero progressivo che caratterizza le colonne delle CCD ma anche le  $\lambda$  ad esse associate, informazione che nell’analisi dei dati è di primaria importanza.

Le connessioni così aperte vengono poi utilizzate nell’operazione di caricamento in memoria dell’oggetto cubo, al quale tali connessioni sono passate quali argomenti del costruttore

del cubo stesso. Tutto ciò che riguarda l'oggetto cubo, la lettura dei file ed i metodi in esso definiti per le visualizzazioni dei dati saranno spiegate in maniera esauriente nel prossimo paragrafo.

L'ultima operazione che viene eseguita, terminata la fase precedente, riguarda la costruzione dell'interfaccia grafica, effettuata subito dopo che il programma ha distrutto la *progress-bar* ormai inutile. Anche questa operazione viene effettuata a partire dalla classe *AppletPanels*, alla quale sono passati come argomenti del costruttore sia il nome del file che l'oggetto cubo.

## 7.4) L'oggetto Cubo

Le prime righe di istruzioni presenti all'interno della classe *Cubo*, come avviene per tutte le classi, sono quelle che definiscono le librerie Java utilizzate all'interno della classe e, successivamente, la definizione di tutte le variabili condivise da tutti i metodi della classe. In questa sezione, in particolare, sono date le definizioni di tutti gli *array* di dati che sono utilizzati per memorizzare le informazioni da restituire agli oggetti che ne facciano richiesta allo scopo di visualizzare una qualche parte del cubo: *l'array cubo*, per esempio, contiene il cubo nella sua interezza; *l'array frame* è in grado di contenere un *frame*, ovvero una *line*; *l'array image* è in grado di contenere una immagine monocromatica, ovvero una *band*. Oltre a questi sono definiti anche i due *array* per contenere il profilo di riga spezzato nelle due componenti, visibile e infrarosso, e quello per contenere il profilo di colonna.

Ciascuno dei suddetti *array* è definito due volte, con due nomi leggermente differenti. Ciò è necessario poiché nell'archivio on-line sono contenuti due tipi di dati: i dati grezzi e quelli calibrati. I dati grezzi sono caratterizzati dall'avere ogni singolo pixel appartenente al cubo descritto da un valore intero, quindi a due byte. I dati calibrati, invece, hanno ciascun pixel descritto da un valore *float*, per il quale sono utilizzati quattro byte. Questa scelta è legata alla natura di Java. Una stessa variabile, qualunque sia la sua natura,

non può, infatti, essere ridefinita. È stato, quindi, necessario definire due insiemi di variabili, uno per i cubi grezzi e l'altro per quelli calibrati, da utilizzare di volta in volta a seconda dei casi. Nel prosieguo del capitolo verrà discusso tutto ciò che concerne il caso dei dati grezzi. Il discorso relativo ai dati calibrati è da considerarsi assolutamente uguale, fatte salve alcune differenze legate al tipo differente di dato utilizzato nei due casi. Per tale motivo, al fine di evitare inutili ripetizioni, verrà presentato solo il primo dei due casi.

### 7.4.1) Il costruttore

Al momento del caricamento dell'*applet*, vengono stabilite due connessioni, una col file di dati e l'altra con un file di testo contenente le lunghezze d'onda  $\lambda$  associate alle colonne della CCD che vengono passate quali argomenti al costruttore del cubo.

Il costruttore è un metodo speciale, presente all'interno di ogni classe. Esso contiene le informazioni per la creazione dell'oggetto definito da una certa classe. Nel momento della creazione è possibile passare alcuni parametri a tale costruttore in modo da inizializzare l'oggetto da esso costruito, ovvero assegnare a tale oggetto uno stato iniziale ben determinato. L'istruzione di creazione di un oggetto, in generale, ha la forma seguente:

```
Oggetto nome_oggetto = new Oggetto(par1, ..., parN);
```

nella quale "Oggetto" rappresenta la classe che viene istanziata, "nome\_oggetto" è il nome del particolare oggetto creato, "new" è la parola che indica la creazione di un nuovo oggetto e `Oggetto(par1, ..., parN)` è il modo di passare i parametri al costruttore dell'oggetto stesso. Nel caso del cubo-dati di *VIMS* la forma dell'istruzione è la seguente:

```
Cubo qube = new Cubo(urlConn, urlConn2, dataType);
```

nella quale *qube* è il nome del particolare oggetto Cubo creato, `urlConn`, `urlConn2` e `dataType` sono i parametri passati. In particolare i primi due parametri rappresentano le connessioni citate in

precedenza, "dataType" è la stringa utilizzata per indicare se si tratta di un file grezzo o calibrato. Non appena ha ricevuto i parametri il costruttore esegue tre attività: legge i parametri necessari dall'*header* del cubo, legge i dati conservati nella parte binaria del cubo stesso e, infine, legge l'*array* con le  $\lambda$ . Le suddette azioni sono descritte nei prossimi tre paragrafi.

### 7.4.2) Lettura dei valori delle keyword

Questa attività è svolta dal metodo `setQubeParameters` (`URLConnection urlConn`), la quale riceve, come argomento, il parametro di connessione al file cubo remoto. Il metodo crea, utilizzando una classe appositamente sviluppata da Sun, un flusso di dati, a partire dal file remoto, per poterlo leggere. Tale flusso viene utilizzato da un'altra classe per poter eseguire una lettura di tipo testuale. Una volta aperto il file viene eseguito su di esso un ciclo in lettura alla ricerca di una serie di *keyword* utilizzate per ricavare le informazioni necessarie per leggere il cubo di dati custodito nel file stesso. Tale ciclo legge, una riga alla volta, tutto l'*header* alla ricerca, come spiegato, delle *keyword* necessarie, finché non incontra la riga di fine *header*, la quale causa la fine del ciclo.

Le *keyword* che vengono ricercate sono le seguenti:

*^QUBE*: un puntatore che indica il numero di record dal quale è composto l'*header*;

*CORE\_ITEMS*: che contiene i valori indicanti le dimensioni del cubo di dati scritto all'interno del file;

*CORE\_ITEM\_BYTES*: che indica il numero di byte utilizzati per descrivere il singolo pixel;

*SUFFIX\_ITEMS*: che contiene il numero di piani ausiliari che affiancano il cubo di dati;

*SUFFIX\_BYTES*: che mostra il numero di byte utilizzati per il singolo pixel nei piani ausiliari.

Quando la riga letta contiene la *keyword* *BAND-BIN\_CENTER*, essendo quest'ultima posta sempre alla fine dell'*header*, come già accennato, il ciclo ha termine.

Ogni volta che una delle suddette *keyword* viene trovata nella riga letta dal file, viene immediatamente estratto il valore della stessa e associato ad una variabile che sarà utilizzata successivamente per la lettura del cubo o semplicemente per mostrare all'utente, in un'apposita finestra di testo, alcune informazioni utili relative al cubo. Nel caso di valori multipli il programma provvede a separarli ed associare ciascuno dei componenti ad una specifica variabile.

### 7.4.3) Lettura del cubo

Questa attività è la più importante fra tutte, in quanto, grazie ad essa, tutti i dati custoditi nel cubo dati, vengono poi messi a disposizione per essere visualizzati all'utente. La lettura della parte binaria del file è eseguita dal metodo `setQubeArray` (`URLConnection urlConn`), il quale riceve, analogamente al metodo per la lettura dell'*header*, come argomento, la connessione al file remoto. Anche in questo caso viene utilizzata una classe, la stessa del caso descritto nel paragrafo precedente, per creare un flusso di dati il quale, poi, viene utilizzato da un'altra classe che, a differenza del caso precedente, esegue una lettura binaria del flusso stesso. Preliminarmente alla lettura vera e propria, il metodo esegue un calcolo del numero di byte che deve saltare, partendo dall'inizio del file, e, quindi, la posizione all'interno dello stesso dalla quale cominciare la lettura. Tale spazio è, come si può facilmente comprendere, quello occupato dall'*header*. Il suddetto calcolo avviene utilizzando uno dei valori delle *keyword* lette in precedenza, cioè, il valore di *^QUBE*. Il valore di questa *keyword*, unito al numero di byte che formano un record, fornisce al metodo il valore ricercato, secondo la semplice formula:

$$skip = 512 \cdot (qube\_param - 1)$$

essendo “skip” il numero di byte cercato, “qube\_param” il valore associato a  $\wedge QUBE$ , 512 il numero di byte in un record. Quest'ultimo valore, pur essendo formalmente libero è, nella sostanza, fisso. In questa fase vengono anche definiti, nell'ordine, il numero di byte occupati dal *sideplane* e il numero di byte occupati dal *backplane*. Analogamente a quanto visto prima le formule utilizzate per il calcolo dello spazio occupato dai piani sono le seguenti:

$$skip\_backplane = (samples + 1) \bullet 4$$

$$skip\_sideplane = suffix\_bytes$$

essendo “suffix\_bytes” il numero di byte utilizzati per ogni pixel dei piani ausiliari. I valori di queste due variabili saranno giustificati più avanti, alla fine della spiegazione di come avviene la lettura del cubo. Terminata questa fase ha inizio la lettura del cubo. Il programma, dopo aver saltato il numero di byte definiti dalla variabile *skip*, infatti, dà il via al ciclo mostrato qui di seguito:

```
for (li=0; li<lines; li++) {
    for (ba=0; ba<bands; ba++) {
        for (int sa=0; sa<samples; sa++) {
            qube[sa][ba][li] = dis.readShort();
        }
        //----salta la lettura del sideplane-----
        dis.skipBytes(skip_sideplane);
    }
    //----salta la lettura del backplane-----
    for (int i=0; i<bands_suffix; i++) {
        dis.skipBytes(skip_backplane);
    }
}
```

**Codice 7. 1 frammento di codice dal metodo di lettura del cubo.**

È stato affermato in precedenza che esistono ordini di scrittura differenti utilizzati per registrare i cubi all'interno dei file. Si è anche visto che l'ordine utilizzato per i dati di *VIMS* è quello

chiamato BIL (Band Interleaved by Line). Ciò fa sì che i cubi vadano letti una *line* alla volta e che ogni *line* debba essere letta una *band* alla volta. In pratica ad ogni ciclo viene letto uno spettro, corrispondente a quanto letto durante una acquisizione dalla camera CCD. Ciascuno spettro è letto una colonna alla volta. Questo è esattamente ciò che fa il ciclo mostrato sopra: il ciclo principale è eseguito su tutte le *line*. Ad ogni passo di tale ciclo vengono eseguiti, in sequenza, due cicli secondari: il primo è eseguito su tutte le *bands*, ovvero su tutte le colonne dell'*array* 2-D costituito da una *line*; il secondo su tutte le colonne del *backplane*.

Il primo dei cicli appena citati contiene, a sua volta un ciclo su tutti gli elementi di un *sample*, ovvero su tutti i pixel di una colonna. Durante tale ciclo è effettuata la lettura dei valori contenuti nei suddetti pixel, lettura eseguita utilizzando il metodo `readShort()`, il quale legge una coppia di byte alla volta. Come già spiegato, infatti, nel caso dei dati non calibrati, i dati del cubo sono registrati utilizzando una coppia di byte per ogni pixel. Non appena terminata la lettura dei pixel di una colonna, è eseguita un'istruzione la quale fa sì che un certo numero di byte seguenti siano saltati. Al termine di ogni colonna, infatti, si trovano registrati alcuni byte facenti parte del *backplane*. Tali byte, in numero pari al valore *suffix\_bytes*, contengono informazioni non utili agli scopi che si prefigge l'*applet* e, per tale motivo, non sono letti.

Il secondo dei cicli presenti all'interno del ciclo principale serve ad evitare che vengano letti i byte che fanno parte del *sideplane*. Dopo aver letto tutte le colonne di uno spettro, infatti, sono registrate alcune colonne le quali contengono informazioni ausiliarie che non sono utili ai fini dell'*applet*. Come per il *backplane*, anche in questo caso, quindi, i byte relativi a tali colonne sono saltati.

#### 7.4.4) Lettura del file con le $\lambda$

La lettura del file contenente le lunghezze d'onda  $\lambda$  associate alle colonne della CCD è eseguita dal metodo `setLambdaArray`

(`URLConnection urlConn2`), il quale, come i precedenti, riceve, quale argomento, la connessione al file `VIMS_tabella_numcol_lambda.txt`. Tale file ha una struttura molto semplice: si tratta, infatti, di un file di testo contenente una sola colonna lungo la quale sono riportati i valori delle lunghezze d'onda. Essendo noto il numero di elementi che devono essere letti, pari al numero di colonne della matrice, ovvero al valore della variabile `band`, viene eseguito un piccolo ciclo durante il quale, ad ogni passo è letto un valore.

```
public void setLambdaArray(URLConnection urlConn3) {
    try {
        lambdaIsr = new InputStreamReader(urlConn3.getInputStream());
        //System.out.println("Available:" + urlConn3.getInputStream().available());
        lambdaBr = new BufferedReader(lambdaIsr);
        String line = "";
        int pos;
        for (int i=0; i<352; i++) {
            line = lambdaBr.readLine();
            pos = line.indexOf(".",0);
            lambdaArray[i] = Float.parseFloat(line.substring(pos-1).trim());
            //System.out.println(lambdaArray[i]);
        }
    } catch (Exception e) {
        System.out.println("Errore nella lettura dell'array di lambda:" + e);
    }
}
```

**Codice 7. 2 Metodo di lettura delle lunghezze d'onda associate alle colonne della CCD.**

### **7.4.5) I metodi per il recupero dei dati**

Al termine delle operazioni descritte nei tre paragrafi precedenti, e una volta completate le altre operazioni, eseguite automaticamente dalla JVM, previste per la creazione di un oggetto, l'oggetto cubo è pronto e a disposizione di una qualunque altra classe che voglia utilizzare i suoi metodi per recuperare i dati in esso contenuti. L'oggetto cubo, infatti, dispone di una serie di metodi



sviluppati per permettere di avere accesso agli spettri, alle immagini monocromatiche e ai profili di diverso tipo che è possibile ottenere a partire da esso. Qui di seguito vengono presentati tali metodi, il cui utilizzo sarà descritto più avanti, quando verranno illustrate le altre classi e come queste siano utilizzate per la visualizzazione dei suddetti dati.

### 7.4.5.1) profilo di riga dallo spettro

Il profilo di riga, ovvero il profilo lungo la direzione delle *bands*, viene mostrato in due finestre differenti, al fine di separare la parte visibile da quella infrarossa. I due profili sono generati da due metodi che funzionano contemporaneamente ma separatamente.

```
public int[] getVisRowProfile(int sample, int line) {
    int min=100000, max=-100000;
    for (int ba=0; ba<96; ba++) {
        visProfile[ba] = (int) qube[ba][sample][line];
    }
    for (int i=0; i<96; i++) {
        if (visProfile[i]<min) {min=visProfile[i];}
        if (visProfile[i]>max) {max=visProfile[i];}
    }
    //----gli elem da 0 a 95 contengono il profilo,-----
    //----il 96 contiene il min e il 97 il max-----
    visProfile[96]=min;
    visProfile[97]=max;
    return visProfile;
}
```

#### Codice 7.3 Metodo di recupero della porzione appartenente al canale visibile di un profilo di riga.

Il profilo di riga nel visibile è restituito dal metodo `getVisRowProfile(int sample, int line)`, il quale, come si vede, riceve quali argomenti la *line* e il *sample* all'interno di tale *line* dalla quale va estratto il profilo di riga.

Per la parte infrarossa è stato definito il metodo `getIRRowProfile(int sample, int line)`, assolutamente

analogo. La differenza tra i due metodi risiede solo nella porzione di spettro che essi prendono in considerazione. Il primo dei due metodi, infatti, estrae dalla riga solo i primi 96 elementi, quelli relativi proprio alla parte visibile. Il secondo metodo, invece, estrae i rimanenti 256 elementi, appartenenti al canale infrarosso. In entrambi i casi i valori della porzione di profilo vengono registrati, mediante un ciclo sui pixel appartenenti alla porzione di profilo stessa, nell'apposito vettore definito all'inizio della classe.

Entrambi i metodi, una volta terminato il riempimento del vettore di propria competenza, eseguono un controllo su tutti gli elementi, mediante un ciclo, per determinare l'elemento col massimo valore e quello col minimo valore. Questi due valori vengono inseriti, rispettivamente, all'ultimo e al penultimo posto del vettore stesso.

### **7.4.5.2) profilo di colonna**

Per profilo di colonna si intende il profilo lungo una *sample*, ovvero lungo una colonna della matrice che forma lo spettro. Tale profilo è restituito da getColumnProfile (int band, int line).

```
public int[] getColumnProfile(int band, int line) {
    columnProfile = new int[samples+2];
    int min=100000, max=-100000;
    for (int sa=0; sa<samples; sa++) {
        columnProfile[sa] = (int) qube[band][sa][line];
    }
    for (int i=0; i<samples; i++) {
        if (columnProfile[i]<min) {min=columnProfile[i];}
        if (columnProfile[i]>max) {max=columnProfile[i];}
    }
    //----gli elem da 0 a (samples-1) contengono il profilo,----
    //----il samples contiene il min e il samples+1 il max-----
    columnProfile[samples]=min;
    columnProfile[samples+1]=max;
    return columnProfile;
}
```

**Codice 7. 4 Metodo per il recupero di un profilo di colonna.**

Questo metodo funziona in maniera del tutto analoga ai metodi per la generazione dei profili di riga. Esso riceve come argomenti la *line* e la *band* interna alla *line* dalla quale estrarre il profilo, il quale viene registrato in un *array* al cui interno, alla fine di un ciclo di controllo analogo a quello precedentemente descritto, vengono inseriti anche i valori massimo e minimo tra quelli presenti nell'*array* stesso.

### 7.4.5.3) immagine monocromatica

Uno degli scopi dell'*applet* è quello di fornire una visualizzazione rapida delle immagini monocromatiche del *target* inquadrato durante l'acquisizione del cubo. Tale compito è affidato al metodo `getImageMono` (`int band`), il quale ha proprio il compito di restituire alla classe che gliela richiede l'immagine monocromatica ad una determinata lunghezza d'onda  $\lambda$ , passata quale argomento al metodo stesso.

```
public short[][] getImageMono(int band) {
    imageMono = new short[lines][samples];
    for (int li=0; li<lines; li++) {
        for (int sa=0; sa<samples; sa++) {
            short value = qube[band][sa][li];
            imageMono[li][sa] = value;
        }
    }
    return imageMono;
}
```

#### Codice 7. 5 metodo di recupero di una immagine monocromatica.

Il procedimento usato per ricavare l'immagine monocromatica utilizza l'*array* 3-D *qube*, che contiene tutto il cubo, ed esegue un ciclo su di esso per estrarre tutti i pixel che fanno parte dell'*array* 2-D che si ottiene da *qube* una volta fissata una delle tre dimensioni, quella delle *bands*.

#### 7.4.5.4) spettro

Analogamente al metodo precedente anche in questo caso si tratta di ricavare un *array* 2-D dall'*array* 3-D *qube*. A questa operazione provvede il metodo `getSpectrum2D(int line)` il quale riceve come argomento una determinata *line* ed estrae da *qube*, fissando anche in questo caso una delle tre dimensioni. Il risultato, però, in questo caso, è una matrice contenente esattamente ciò che viene registrato dalla camera CCD durante un'acquisizione, ovvero uno spettro o *frame*.

```
public short[][] getSpectrum2D(int line) {
    spectrum2D = new short[bands][samples];
    short value;
    for (int ba=0; ba<bands; ba++){
        for (int sa=0; sa<samples; sa++) {
            value = qube[ba][sa][line];
            //System.out.println(value + ", ");
            spectrum2D[ba][sa] = value;
        }
    }
    return spectrum2D;
}
```

**Codice 7. 6 Metodo di recupero di uno spettro.**

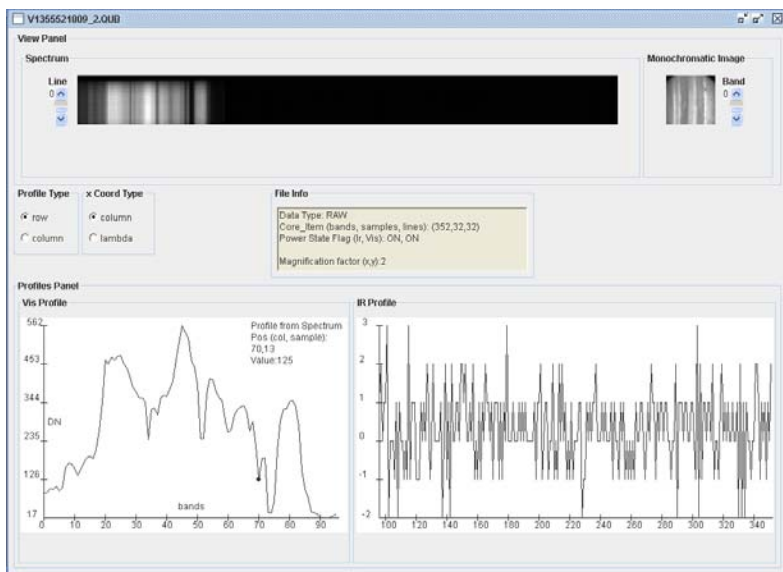
#### 7.4.5.5) altri

Oltre ai suddetti metodi sono presenti anche altri metodi, definiti in maniera molto semplice, i quali hanno come scopo quello di restituire i valori relativi ad alcune *keyword*. Esempio di ciò sono i metodi `getBands()`, `getLines()` e `getSamples()` i quali, come si intuisce dal nome, hanno il compito di restituire, rispettivamente, il numero di *bands*, *lines* e *samples*. È anche presente il metodo `getValue(int band, int sample, int line)` il quale restituisce il valore di un pixel determinato dalle coordinate passate dai tre valori che il metodo in questione riceve quali argomenti.

## 7.5) L'interfaccia grafica

L'interfaccia grafica è caricata e mostrata all'utente alla fine del processo di lettura del file cubo remoto. L'*applet*, così come si mostra all'utente, è presentata nella Figura 7.2, qui di seguito.

Da una prima occhiata si individuano immediatamente tre



**Figura 7. 2** Interfaccia grafica dell'applet VIMS Windows.

fascie distinte, disposte verticalmente l'una sopra l'altra. La prima fascia, individuata dal contenitore denominato *View Panel*, comprende, sulla sinistra, all'interno del pannello denominato *Spectrum* lo spettro relativo ad una certa *line* ed il selettore che consente di selezionare la *line* che l'utente vuole visualizzare. Sulla destra, invece, all'interno del pannello *Monochromatic Image*, è presente l'immagine monocromatica col relativo selettore di banda da visualizzare. Di *default*, all'avvio, vengono visualizzate la *line* 0 e la *band* 0.

La seconda fascia, quella centrale, contiene tre pannelli indipendenti: il primo da sinistra, *Profile Type*, contiene i selettori per stabilire il tipo di profilo che si intende visualizzare. Si può infatti visualizzare sia il profilo di riga che quello di colonna. Il secondo pannello, *x Coord Type*, contiene i selettori per il tipo di scala che si vuole visualizzare lungo l'asse delle x. È possibile, infatti, utilizzare sia i numeri progressivi delle colonne della CCD che le lunghezze d'onda ad esse associate. Il terzo pannello, *File Info*, infine, contiene un riquadro nel quale sono riportate alcune informazioni relative al cubo. Vengono qui riportate, infatti, le dimensioni del cubo di dati contenuto nel file, il *datatype*, l'indicazione di quali dei due canali erano accessi durante l'acquisizione del file e, infine, il fattore di ingrandimento utilizzato per visualizzare le immagini monocromatiche e gli spettri. Tipicamente le acquisizioni di *VIMS-V* sono eseguite in maniera tale da generare immagini e spettri piuttosto piccoli in termini di pixel occupati visualizzandoli su un monitor. Ciò rende difficile apprezzare i particolari. Per questo motivo si è deciso, in fase di visualizzazione, di applicare un ingrandimento ad entrambi. L'indicazione riportata nel pannello indica proprio quale sia tale fattore di ingrandimento. Il metodo col quale tale operazione è eseguita verrà spiegato più avanti, quando si parlerà della visualizzazione di spettri e immagini e degli oggetti ad essa preposti.

La terza ed ultima fascia, individuata dal pannello *Profile Panel*, è quella posta più in basso, fascia che comprende i due riquadri di visualizzazione dei profili. Il riquadro sulla sinistra, chiamato *Vis Profile*, è quello nel quale vengono visualizzati sia la porzione nel visibile dei profili di riga sia i profili di colonna. Il riquadro sulla destra, chiamato *IR Profile*, è utilizzato per visualizzare la porzione infrarossa dei profili di riga.

Le quattro finestre utilizzate per visualizzare lo spettro, sono gestite da quattro oggetti, o meglio da due coppie di oggetti uguali. I due oggetti che gestiscono la visualizzazione degli spettri e delle immagini monocromatiche sono infatti definiti da due classi che hanno una classe padre comune e si differenziano solo per alcuni metodi. Analogo discorso per i due oggetti che visualizzano i profili.

Le classi dalle quali derivano i suddetti oggetti saranno descritte di seguito. La descrizione verrà effettuata seguendo la divisione in due categorie, come messo in evidenza dalla appartenenza delle finestre di visualizzazione a due fasce differenti, individuate dai due pannelli *View Panel* e *Profile Panel*.

## 7.6) View Panel

Le due classi dalle quali derivano i due oggetti responsabili della visualizzazione degli spettri e delle immagini monocromatiche sono, rispettivamente, *SpectrumDraw* e *ImageDraw*. entrambe queste classi hanno in comune, o come si dice in gergo “ereditano”, la classe padre *ViewWindowBase*. Ciò significa che i metodi e le variabili definite all’interno di quest’ultima classe appartengono anche alle classi che la ereditano. Il concetto di ereditarietà nella programmazione ad oggetti nasce dalla volontà di organizzare razionalmente il codice scritto e di limitarlo il più possibile al tempo stesso. Se più classi, infatti, hanno delle variabili in comune o utilizzano uno stesso metodo per eseguire una certa operazione si può pensare di definire una classe che abbia tali variabili e metodi definiti e poi di “includerla” in altre classi che abbiano, oltre alle caratteristiche comuni definite dalla classe padre, delle proprie specificità. Ciò evita di dover riscrivere più volte lo stesso codice e, in caso di correzione ad un certo metodo, in comune, di dover effettuare la stessa operazione più volte. Ovviamente tutto questo discorso si inquadra in una “filosofia” di scrittura del codice caratteristica della programmazione ad oggetti. Tuttavia non è scopo di questa tesi una discussione approfondita di tale aspetto.

### 7.6.1) ViewWindowBase.java

È la classe padre, come già accennato, delle due che visualizzano gli spettri e le immagini monocromatiche. Tale classe estende la classe *Canvas*, la quale è la classe preposta da java alla

visualizzazione di figure ed immagini di qualunque tipo, siano esse derivate da file o create colorando i pixel che la costituiscono. Essa è definita in maniera che l'oggetto che da essa deriva riceva, al momento della creazione, il cubo *cube* quale parametro.

I metodi contenuti in questa classe sono solo tre: due di essi servono, in momenti diversi, uno per impostare la *line* che si vuole visualizzare e l'altro per ricavare quale sia la *line* selezionata. Il terzo metodo è quello che si occupa di produrre l'ingrandimento sia dello spettro che dell'immagine monocromatica. Esso, infatti, riceve come argomento un *array* bidimensionale di certe dimensioni e restituisce un *array* di dimensioni doppie. Tale lavoro è eseguito mediante un doppio ciclo, uno sulle righe ed uno sulle colonne della matrice ricevuta, durante i quali vengono presi, uno ad uno, tutti i pixel della matrice stessa e trasformati ciascuno in una matrice quadrata di due pixel di lato. Ovviamente tali matrici 2 x 2 vengono opportunamente collocate all'interno della matrice restituita alla fine del processo, in maniera da avere un corretto ingrandimento dell'immagine di partenza.

## **7.6.2) SpectrumDraw.java**

Questa classe, oltre ad ereditare la classe padre descritta in precedenza, implementa un'altra classe importante, chiamata *MouseMotionListener*, la quale permette di gestire il movimento del mouse all'interno della finestra grafica, associando a tale movimento degli eventi definiti al momento della scrittura del codice. La parola "implementazione" merita una piccola chiarificazione: in Java esistono classi dette "interfacce" le quali posseggono dei metodi "vuoti", ovvero per i quali non sono definite istruzioni. Tali classi possono essere implementate, ovvero "inglobate" in una classe che ne utilizza i metodi i quali, però, necessitano di essere ridefiniti.

La classe *MouseMotionListener*, per esempio, possiede due metodi, *MouseMoved*(*MouseEvent e*) e *MouseDragged*(*MouseEvent e*), i quali ricevono come argomento un evento legato al movimento del mouse, come si intuisce dal nome dell'argomento



stesso. Questi metodi vengono chiamati nel momento in cui, rispettivamente, il mouse è mosso sulla finestra grafica o trascinato all'interno di essa, ma non contengono istruzioni già definite. Il programmatore può, quindi, inserire delle istruzioni al loro interno. Tale operazione è chiamata “ridefinizione”.

```
class SpectrumDraw extends ViewWindowBase implements MouseMotionListener {  
    ...  
}
```

**Codice 7.7** Definizione della classe *SpectrumDraw*. È visibile l'istruzione per ereditare la classe *ViewWindowsBase* e quella per implementare la classe *MouseMotionListener*.

I metodi contenuti dalla classe *SpectrumDraw* servono per la visualizzazione degli spettri e, indirettamente, per la generazione dei profili di riga o colonna. Il metodo *MouseMoved* è stato ridefinito proprio per essere utilizzato allo scopo di visualizzare i profili. Esso contiene le istruzioni per recuperare le coordinate del mouse all'interno della finestra grafica, coordinate che vengono poi passate ad uno dei metodi della classe *VisProfileDraw*, la quale si occupa di visualizzare il profilo selezionato.

L'altro metodo rilevante della classe in esame è il metodo *paint* (*Graphics g*), il quale si occupa di visualizzare lo spettro. Esso riceve come argomento l'oggetto *Graphics*, il quale contiene tutti i metodi necessari per qualunque operazione grafica. La prima operazione effettuata durante il processo di visualizzazione dello spettro è quello di recuperare, dall'oggetto cubo, lo spettro da visualizzare. Questa azione è eseguita utilizzando il metodo *getSpectrum2D(int line)* del cubo stesso, già visto in precedenza.

Una volta recuperato lo spettro ne vengono calcolati i valori minimo e massimo, operazione necessaria per definire i livelli di grigio da associare ai pixel presenti nello spettro stesso. Il colore di ogni pixel, infatti, è calcolato secondo una semplice formula che utilizza i valori massimo e minimo, del segnale, presenti all'interno

di una *line*, oltre, ovviamente, al valore del segnale presente all'interno del pixel stesso. Tale formula è riportata qui di seguito.

```
col = Math.round(((val - minVal)/(maxVal-minVal))*255);  
g.setColor(new Color(col,col,col));
```

**Codice 7. 8 Formula per il calcolo del valore relativo al colore da associare ad un pixel dello spettro e successiva assegnazione del colore stesso.**

Il colore “col” è calcolato moltiplicando per una costante, pari a 255, il rapporto tra due valori: il numeratore è il valore del segnale in un pixel meno il valore minimo presente nello spettro in esame; il denominatore è pari alla differenza tra i valori massimo e minimo, del segnale, presenti nello spettro. L'istruzione seguente serve ad assegnare al pixel un colore. Ciò viene fatto mediante l'istruzione `setColor(Color(col, col, col))`, metodo presente nella classe *Graphics*.

Terminata questa fase il programma recupera lo spettro ingrandito secondo la procedura spiegata nel paragrafo 7.6.1, e, infine, ogni pixel è colorato. Al termine di questa operazione lo spettro è visibile all'utente.

### **7.6.3) ImageDraw.java**

Questa classe, come la precedente, deriva dalla classe padre *ViewWindowBase* e si occupa della visualizzazione delle immagini monocromatiche contenute all'interno del cubo. Come la classe descritta precedentemente, inoltre, implementa la classe `MouseListener()`, per gestire il movimento del mouse sulla finestra grafica. Tale movimento genera la visualizzazione di un profilo lungo le *bands*, analogo al profilo di riga sullo spettro. Il comportamento di questa classe è del tutto analogo a quanto visto per la classe precedente, a parte le opportune differenze dovute alle diversità fra spettri ed immagini monocromatiche. Una descrizione

approfondita di tale classe, pertanto, sarebbe una ripetizione di quanto detto nel paragrafo precedente.

### 7.6.4) ProfileWindowBase.java

È la classe padre delle due che si occupano di visualizzare i vari tipi di profili, *VisProfileDraw* e *IRProfileDraw*. Essa contiene alcuni metodi comuni alle due suddette classi e fornisce ad esse un costruttore comune che inizializza alcune variabili utili ad entrambe. Analogamente alla classe *ViewWindowBase*, anche questa classe estende la classe *Canvas*, in maniera da essere adatta a visualizzare immagini o grafici di qualunque tipo. Il costruttore di questa classe

```
ProfileWindowBase (Cubo qube, String profileType, String profileColumnType) {
    this.qube = qube;
    this.samples = qube.getSamples();
    this.lines = qube.getLines();
    setProfileColumnType(profileColumnType);
    setProfileType(profileType);
    profileSource = "spectrum";
}
```

#### Codice 7. 9 Costruttore della classe ProfileWindowBase.

riceve tre argomenti: l'oggetto cubo, dal quale estrae in seguito alcune informazioni; il tipo di profilo, riga o colonna; la scala da utilizzare in caso il profilo selezionato sia quello di riga, per il quale, come già spiegato, è possibile avere sulla scala delle x sia i numeri di colonna che le  $\lambda$  associate alle colonne stesse. Di *default*, inoltre, imposta al valore "spectrum" la variabile che segnala quale sia la finestra dalla quale è generato il profilo. Tale variabile è aggiornata ogni volta che il mouse è portato sulla finestra dello spettro o su quella dell'immagine monocromatica.

I metodi presenti in questa classe sono per lo più utilizzati per impostare le variabili già citate riguardanti il tipo di profilo, la scala o la finestra "sorgente" del profilo o per recuperare tali valori.

Oltre a ciò, però, sono presenti altri due metodi: `getCuted-Value(String value)`, e `setMousePosition(int xWin, int yWin, int line, String bandType)`.

Il primo metodo è utilizzato durante la visualizzazione dei profili di riga generati dalla finestra relativa allo spettro o del profilo spettrale generato a partire dalla finestra relativa all'immagine monocromatica. La sua funzione è sostanzialmente quella di arrotondare il valore della  $\lambda$  visualizzata sulla scala delle  $x$ , per non occupare un eccessivo spazio.

Il secondo metodo è utilizzato per recuperare la posizione del mouse sull'immagine o sullo spettro. Tale operazione deve tener conto del fatto che sia l'immagine che lo spettro sono mostrati ingranditi e che l'asse delle  $y$  della finestra grafica è invertito rispetto al verso tradizionalmente utilizzato. Ciò comporta che le coordinate del mouse sullo spettro o sulla immagine monocromatica derivino da quelle sulle rispettive finestre mediante due semplici formule visualizzate qui di seguito:  $xWin$  e  $yWin$  sono le coordinate del mouse sulla finestra grafica,  $x$  e  $y$  le coordinate cercate.




```
x = (int) Math.floor(xWin/2);  
y = (int) Math.floor((samples-1)-(yWin/2));
```

**Codice 7. 10 Calcolo delle coordinate del mouse sullo spettro o sulla immagine monocromatica a partire dalle coordinate sulle rispettive finestre grafiche.**

### 7.6.5) VisProfileDraw.java

Al momento della creazione di questa classe il costruttore si occupa, come prima azione, di impostare le dimensioni e lo sfondo della finestra grafica che ospiterà i profili. Successivamente prepara una serie di *array* delle dimensioni e del tipo appropriato alla visualizzazione dei profili stessi. I metodi contenuti nella classe in esame sono tutti volti alla creazione ed alla visualizzazione dei

profili. Il metodo base, che regola le attività degli altri, è `paint(Graphics g)`. Tale metodo è quello utilizzato per lo svolgimento di tutte le “attività grafiche” di una classe. Gli altri metodi utilizzati sono, nell’ordine:

-  `setXarr(int factor);`
-  `getProfile();`
-  `setVisDrawnXScale().`

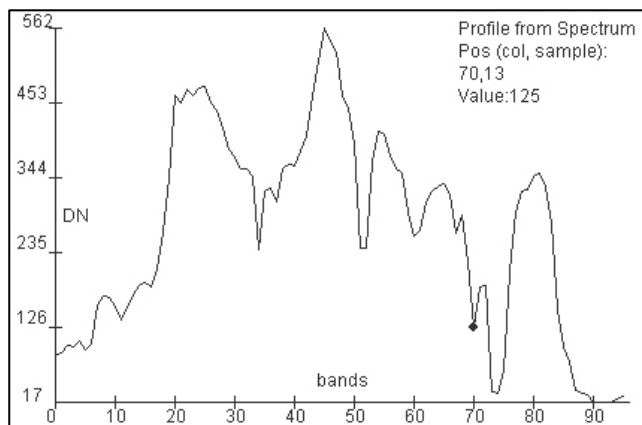
Ognuno di questi svolge un ruolo ben preciso all’interno di un lungo processo per la visualizzazione non solo del profilo vero e proprio ma anche per la visualizzazione delle scale *x* e *y*, dei valori sulle scale, e della scrittura delle informazioni ausiliarie sul profilo stesso. Qui di seguito sarà descritto ora il processo di creazione del profilo e della sua successiva visualizzazione.

Il primo metodo chiamato, `setXarr(int factor)`, serve a definire il punto sulla finestra per la generazione dell’indicatore della posizione del mouse sul profilo. Come è visibile dalla Figura 7.3, infatti, sul profilo è visibile un pallino nero che segna, all’interno del profilo generato a partire da una riga o da una colonna, la posizione del mouse. L’intero passato come argomento è un fattore di moltiplicazione per la posizione, avendo il profilo visualizzato una lunghezza pari a 4 volte quella originale.

Immediatamente dopo il termine dell’esecuzione del metodo precedente viene chiamato il secondo citato, `getProfile()`. Il compito di questo metodo, come si intuisce dal nome, è quello di recuperare il profilo selezionato dall’utente. Mediante l’utilizzo delle variabili, menzionate in precedenza, atte a memorizzare sia il tipo di profilo che la finestra sorgente, il metodo in esame si serve del metodo dell’oggetto cubo adatto a recuperare il profilo desiderato e lo memorizza in un apposito *array*. Ovviamente, in tale operazione si deve anche tener conto del tipo di dato, calibrato o meno. Una volta recuperato l’*array*, ha inizio il ciclo per individuare il valore minimo e quello massimo appartenenti all’*array* stesso. Tali valori sono utilizzati in seguito per determinare i valori della scala verticale da visualizzare. Una volta individuati i suddetti valori, essi vengono

inseriti all'interno dell'*array* contenente il profilo, rispettivamente in penultima e ultima posizione.

Il terzo metodo, `visDrawnXScale()`, è utilizzato per determinare i valori che vanno mostrati, successivamente, sulla scala delle *x*. Durante questa fase viene popolato un *array* di 10 elementi nel quale vengono inseriti i numeri associati alle colonne della CCD o le relative lunghezze d'onda  $\lambda$ , a seconda della scala selezionata dall'utente.



**Figura 7. 3 Esempio di porzione nel canale visibile di un profilo di riga.**

Alla fine di tale ciclo il controllo viene restituito al metodo `paint()` che prosegue il lavoro popolando l'*array visXArray*, contenente la coordinata *x* dei punti della finestra grafica che saranno utilizzati per rappresentare i punti del profilo. È stato detto in precedenza, infatti, che la lunghezza del profilo visualizzato è pari a 4 volte la lunghezza del profilo originale. Ciò è fatto per consentire una migliore visualizzazione del profilo stesso. I punti della finestra grafica che corrispondono ai punti del profilo sono quindi calcolati partendo da un determinato punto, utilizzato come *offset*, e proseguendo selezionando un pixel ogni quattro. Le due ultime operazioni, ovviamente, non sono in contrasto, visto che la prima si occupa di determinare i valori delle coordinate, in riferimento alla

CCD, mostrati all'utente, mentre la seconda calcola le coordinate sulla finestra grafica dei punti del profilo.

Il passo seguente consiste nel calcolo delle coordinate y dei punti della finestra grafica corrispondenti ai punti del profilo. Tale calcolo è eseguito mediante un ciclo su tutti i valori dell'*array* contenente il profilo. Durante tale ciclo vengono eseguite le operazioni mostrate qui di seguito: ad ogni valore dell'*array* viene

```
val = visQubeProfile[i]-visProfileMin;
delta = visProfileMax-visProfileMin;
rapp = (float)val/delta;
visDrawnProfile[i]=260-(int)(Math.round(rapp*250));
```

**Codice 7. 11 Operazioni per il calcolo delle coordinate y dei punti del profilo.**

sottratto il valore minimo dello stesso *array*. Il valore risultante viene poi diviso per la differenza tra il massimo e il minimo valore dell'*array*. Infine il risultato viene scalato per tenere conto della dimensione verticale della finestra grafica e dell'orientazione del suo asse delle y, invertito rispetto al verso tradizionale.

Una volta calcolate le coordinate dei punti del profilo resta da creare tutto il contorno, ovvero le due scale dei valori e le informazioni mostrate nella finestra, visibile nella Figura 7.3.

Riguardo la scala delle y c'è da dire che essa è divisa in 5 intervalli uguali. Lo scopo è creare dei valori di riferimento, mostrati lungo la scala stessa per facilitare l'individuazione grossolana dei valori dei punti del profilo. Il programma procede, quindi, a calcolare i valori da mostrare nei punti intermedi individuati. Tali valori sono poi posizionati lungo la scala. La scala lungo l'asse delle x, successivamente, viene riempita con i valori calcolati precedentemente.

A questo punto si ha la visualizzazione sia del profilo che delle scale. Le operazioni grafiche, infatti, sono svolte tutte insieme in maniera da non avere effetti di ritardo. Contemporaneamente è anche visualizzato il pallino che indica la posizione del mouse sul

profilo. Successivamente a questa operazione sono visualizzate le informazioni ausiliarie. Tali informazioni, per la maggior parte, sono costituite da stringhe pre-determinate. Ciò che deve essere recuperato sul momento è semplicemente il valore del segnale sul pixel sul quale è posizionato il mouse. Tale semplice operazione è eseguita sfruttando il metodo `getElementValue(int x, int y, int line)` dell'oggetto `cubo`. La posizione del puntatore sullo spettro o sull'immagine, infine, è un valore già noto, utilizzato per recuperare il profilo, e quindi deve solamente essere mostrato.

### 7.6.6) `IRProfileDraw.java`

La classe *IRProfileDraw*, che si occupa di visualizzare la porzione nel canale infrarosso di un profilo di riga, lavora allo stesso modo della classe descritta nel paragrafo precedente. Le differenze nei due metodi derivano dal fatto di dover gestire due *array* di dimensioni differenti e dalla caratteristica della finestra relativa al visibile di visualizzare anche i profili di colonna. Inoltre sulla finestra del canale infrarosso non sono visualizzate informazioni ausiliarie, che rimangono sempre visualizzate sull'altra finestra. Analogamente a quanto fatto in precedenza, quindi, non sarà qui riportata la descrizione del funzionamento di questa classe, in quanto si tratterebbe di una ripetizione di quanto già detto.



## Conclusioni

Il Sistema Solare è stato oggetto, nel corso degli ultimi anni, di intensa attività di esplorazione. Tale attività è stata effettuata per mezzo di sonde interplanetarie le quali, dopo un viaggio di qualche mese o alcuni anni, si inseriscono in orbita attorno al loro *target* ed eseguono, mediante gli strumenti che ospitano, un enorme numero di acquisizioni di dati che poi sono inviati sulla Terra per essere analizzati. Questo genere di operazioni sono molto costose, da un punto di vista economico, e richiedono, generalmente, anni di preparazione, durante i quali è coinvolto un elevato numero di persone tra tecnici e ricercatori.

In virtù di quanto appena detto si capisce, quindi, quanto cruciale sia il ritorno scientifico di queste missioni, ovvero la capacità, da parte dei ricercatori, di estrarre tutte le informazioni possibili dai dati inviati a terra dalle sonde, al fine di ottenere un sostanziale miglioramento nella comprensione della natura e della evoluzione del corpo osservato.

Tale capacità dipende in maniera non trascurabile dalla capacità di progettare e realizzare un sistema di archiviazione dei dati il quale permetta di facilitare a chi si trovi ad eseguire un lavoro di ricerca di discriminare, in maniera facile e veloce, tutti i dati di cui necessita per il proprio lavoro da tutti gli altri. Ciò è dovuto, in maniera rilevante, all'elevato numero di file, contenenti i dati delle acquisizioni, che vengono prodotti nel corso di una missione, ovvero dai test a terra prima del lancio alla fase di stazionamento in orbita attorno al corpo oggetto di studio. Riuscire ad organizzare e ricercare in maniera efficiente i file utili in una tale situazione può essere fondamentale per riuscire a mettere a frutto gli anni di lavoro passati a preparare la missione.

Proprio allo scopo di mettere in condizione la comunità italiana dei planetologi di avere accesso a tutti i dati derivanti dagli strumenti italiani a bordo di sonde europee o americane, si è deciso di dare il via ad una collaborazione ASI – IFSI allo scopo di creare, presso il centro ASI denominato ASDC, un archivio *on-line* di dati derivanti da strumenti presenti a bordo di alcune missioni quali *Cassini*, *Mars Express*, *Rosetta* e *Venus Express*.

Il lavoro svolto durante l'attività di progettazione e realizzazione di tale archivio *on-line* ha portato alla creazione di uno strumento che offre, agli utenti, diverse possibilità di ricerca e differenti modalità di visualizzazione dei risultati.

Gli strumenti ospitati a bordo di sonde volte all'esplorazione del Sistema Solare i cui dati sono conservati presso ASDC sono, al momento, cinque, anche se non tutti ancora disponibili per le ricerche così come illustrato nella presente tesi. Il criterio generale di funzionamento degli archivi relativi ai diversi strumenti è sostanzialmente lo stesso, con le dovute differenze generate dalla differente natura e/o dalle diverse caratteristiche tecniche degli strumenti medesimi. Nello sviluppo delle pagine web che ospitano i *form* di ricerca e le tabelle risultato nonché degli *script* che operano le ricerche, infatti, si è tentato di creare degli strumenti il più possibile versatili ed adatti a tutti gli strumenti ospitati nell'archivio. Per questo motivo, al solo scopo di evitare la ripetizione di concetti simili, si è scelto di presentare solo il caso di *VIMS-V*, spettrometro ad immagini a bordo della sonda *Cassini-Huygens*.

Come accennato sopra, vengono messi a disposizione degli utenti diverse modalità di ricerca e di presentazione dei dati. Relativamente alle ricerche, nel caso di *VIMS-V*, sono stati presentati i quattro *form* per esso realizzati. Essi sono i seguenti:

- i. *single parameter*;
- ii. *multi-parameter*;
- iii. *session*;
- iv. *time*.

Il primo consente semplici ricerche mediante la selezione di una tra le *keyword* disponibili e l'inserimento del relativo valore. Il sistema provvede a selezionare tutti i file contenenti il valore inserito per la *keyword* scelta dall'utente. Il secondo consente la creazione di *query* di ricerca più complesse, utilizzando una combinazione di più parametri con i rispettivi valori. Il terzo criterio è un sistema di ricerca basato sulla organizzazione per sessioni e sotto-sessioni dei file di dati. Il quarto e ultimo *form* dà all'utente la possibilità di definire un intervallo temporale e ordina al sistema di ricercare tutti i file per i quali il valore della *keyword* "START\_TIME" è contenuto all'interno del suddetto intervallo temporale.

Le tabelle di visualizzazione dei risultati messe a disposizione nel caso di *VIMS-V* sono le seguenti:

- i. *rows table*;
- ii. *pictures table*;
- iii. *download table*.

La prima è la classica tabella nella quale in ciascuna riga sono presentati i valori delle *keyword* relative ad un determinato file, oltre, ovviamente, al nome ed alla sessione di appartenenza del file stesso. La seconda è una tabella creata per presentare le immagini ricavate da ciascun file e relative sia al canale visibile che a quello infrarosso. La terza ed ultima tabella è stata ideata per permettere il *download* di grosse quantità di file, i quali vengono presentati per gruppi appartenenti alla stessa sessione e sotto-sessione.

Oltre a ciò è stata sviluppata un'applet java la quale consente agli utenti di aprire i file ed osservarne il contenuto, sotto forma di spettri ed immagini monocromatiche. Tale applicazione da modo, inoltre, di generare in tempo reale profili di diverso tipo i quali forniscono un'idea più precisa delle informazioni spettrali contenute nel file aperto.

Le possibilità di ricerca e visualizzazione dei dati sopra descritte sono rese possibili solo al termine di un processo di inserimento dei dati stessi in un database relativo allo strumento.

Tale database contiene un elenco di tutti i file disponibili per l'analisi o il *download* oltre al percorso per ritrovarli. Il suddetto processo di inserimento consiste in una serie di script i quali creano un elenco dei file inseriti sul server e ne estraggono tutte le informazioni necessarie al database e, successivamente, le inseriscono nel db stesso. Durante questa fase, inoltre, viene eseguito un programma, appositamente realizzato utilizzando il linguaggio IDL, per estrarre da ogni file una coppia di immagini le quali, al termine del processo, sono copiate sul server e, di conseguenza, disponibili per essere mostrate agli utenti.

Per concludere si vogliono, in questa sede, citare due obiettivi da perseguire in futuro. Il primo di essi è l'ampliamento del database mediante l'aggiunta di ulteriori strumenti, sia da missioni già supportate che da nuove. Un simile obiettivo va inquadrato in un'ottica di consolidamento e di rafforzamento del valore del lavoro fin qui portato avanti attraverso un ampliamento della platea dei planetologi interessati all'archivio.

Il secondo obiettivo consiste nella implementazione di un sistema di ricerca che non sia limitato ad un singolo strumento da selezionare all'inizio ma che sia in grado di eseguire una ricerca all'interno di tutti i database relativi agli strumenti presenti nell'archivio. È noto, infatti, che strumenti presenti a bordo della stessa sonda sono in grado di eseguire acquisizioni in differenti bande spettrali. Capita spesso, quindi, che diversi strumenti operino in contemporanea effettuando acquisizioni della stessa regione spaziale, ciascuno nella banda spettrale di propria competenza. È comprensibile, da quanto detto, quindi, che sarebbe molto utile per chi si trova ad investigare su una determinata regione o su un corpo in particolare, recuperare dati e informazioni provenienti da strumenti differenti, al fine di ottenere un quadro più completo della natura del *target* esaminato.

## Appendice A

# La missione Cassini-Huygens e lo strumento VIMS-V

## A.1) La missione Cassini-Huygens

### A.1.1) La nascita della missione

La sonda “*Cassini*” è la prima ad essere stata posta in orbita attorno a Saturno, posizione che le sta consentendo una osservazione

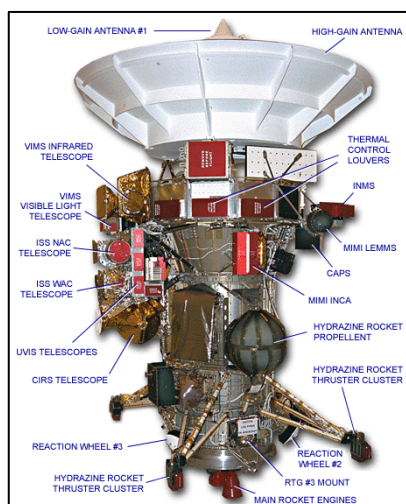


Figura A.1 Immagine della sonda Cassini

dettagliata del pianeta e del suo sistema di anelli e di satelliti. Lo scopo della missione è di studiare questo complesso insieme di corpi, caratterizzati da diversa composizione e, probabilmente, origine.

Questa missione nasce dalla collaborazione tra la NASA e l'ESA. In particolare quest'ultima si è occupata di fornire il Probe mentre l'Orbiter è stato realizzato dalla NASA. L'Agenzia Spaziale Italiana (ASI) si è occupata di costruire alcuni sottosistemi dell'orbiter e strumenti sia per l'orbiter che per la sonda.

### A.1.2) Le fasi della missione

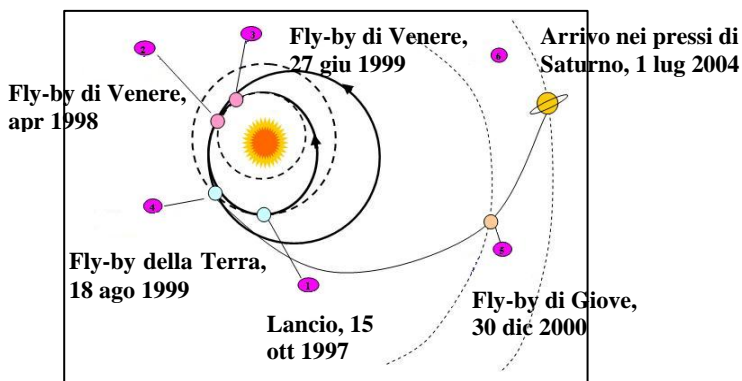


Figura A.2 Schema del percorso tragitto effettuato dalla Sonda Cassini-Huygens nel suo viaggio dalla Terra a Saturno.

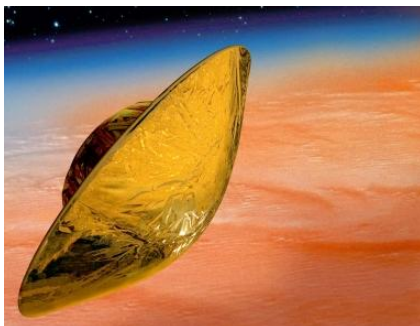
Il 15 ottobre 1997 è avvenuto il decollo del razzo vettore destinato a portare nello spazio la sonda. La traiettoria di *Cassini* è stata costruita in modo da orbitare per ben due volte attorno al Sole prima di dirigersi verso Saturno, passando due volte in prossimità di Venere e una volta nei pressi della Terra. Il motivo di questa scelta risiede nella volontà di sfruttare, come spiegato in precedenza, l'“effetto fionda”, che permette di far variare direzione e velocità alle sonde stesse senza usare i motori.

Il primo incontro con Venere è avvenuto il 26 aprile 1998, il secondo il 24 giugno 1999. Entrambe le occasioni sono state sfruttate sia per effettuare prove di calibrazione degli strumenti sia per acquisire nuovi dati sul pianeta. In particolare sono stati messi in funzione gli strumenti destinati a studiare il campo magnetico, quello per la ricerca di bagliori dovuti ad eventuali fenomeni atmosferici, l'analizzatore di particelle per lo studio della composizione e dell'abbondanza della polvere nello spazio attorno a Venere e lo spettrometro per lo studio dell'interazione tra Venere ed il vento solare.

Passati 55 giorni dal secondo incontro con Venere, la sonda "*Cassini*" ha operato un passaggio ravvicinato alla Terra, fornendo così la possibilità di "puntare" alcuni strumenti in direzione della Terra stessa e della Luna. Le attività legate all'avvicinamento alla Terra hanno avuto una durata di circa sette settimane, dalla metà di luglio ai primi giorni di settembre del 1999; queste attività consistevano in correzioni di traiettoria, manutenzione degli strumenti e misurazioni di vario tipo. Durante l'incontro con la Terra, "*Cassini*" ha ottenuto una quantità di dati superiore a quella ottenuta nei precedenti passaggi in prossimità di Venere. Parte delle informazioni raccolte, come accennato in precedenza, riguardavano la Luna. La sonda è riuscita, infatti, a compiere un'osservazione dell'intero disco lunare utilizzando gli strumenti ottici, evitando di essere disturbata dalla luce proveniente dalla Terra. Ciò è accaduto circa un'ora e mezza dopo il massimo avvicinamento al nostro pianeta, avvenuto alle 3:28 GMT del 18 agosto 1999, ed ha permesso a diversi sottosistemi, tra i quali *VIMS*, di ottenere dati per la calibrazione. Osservazioni effettuate nel periodo successivo al passaggio hanno riguardato anche la "coda" del campo magnetico terrestre.

La fase più critica della missione è stata, comunque, l'immissione nell'orbita di Saturno, avvenuta il 1 luglio del 2004. Oltre ad essere cruciale, è stata anche una fase molto intensa dal punto di vista dell'attività scientifica poiché la navicella, in quel momento, si trovava anche alla minima distanza dal pianeta.

Uno dei principali eventi della missione è stato sicuramente il rilascio dell'orbiter *Huygens*. Tale evento ha avuto luogo nel dicembre del 2004, al termine della terza orbita. *Huygens* ha affrontato un viaggio di 22 giorni, diretto verso Titano, per poi, una volta arrivato, il 14 gennaio 2005, discendere verso la superficie del pianeta.



**Figura A.3** Rappresentazione della discesa di Huygens nell'atmosfera di Titano.

La discesa, durata circa 3 ore, ha avuto inizio con l'apertura del paracadute, all'altezza di 180 km, che ha diminuito la velocità dell'orbiter, portandola a circa 400 m/s. Sceso a 160 km di altezza è stato rilasciato lo scudo termico e, subito dopo, sono stati accesi gli strumenti che hanno effettuato una ininterrotta serie di acquisizioni, proseguite fino all'arrivo alla superficie. Tutte le informazioni raccolte, a questo punto, sono state inviate alla "*Cassini*" che ha provveduto, a sua volta, ad inoltrarle sulla Terra.

La missione, comunque, ha avuto in questo evento solo uno dei numerosi momenti importanti che hanno caratterizzato tutto lo svolgimento della stessa. La possibilità, infatti, di osservare da distanze relativamente piccole molti dei satelliti principali di Saturno, nonché il suo complesso sistema di anelli, è fonte di una considerevole mole di dati per il team di "*Cassini*" innanzitutto e per la comunità scientifica in generale. Osservazioni eseguite da spettrometro e camere nella banda del visibile e nell'infrarosso, ma anche rilevamenti eseguiti mediante il radar, nonché le raccolte di dati eseguite dagli altri strumenti presenti a bordo della sonda stanno



dando un contributo notevole alla conoscenza di questa parte del nostro sistema, ponendo al tempo stesso nuovi quesiti.

Tra tutte quelle fatte, alcune scoperte si sono rivelate particolarmente significative. Tra queste vanno citate sia il rinvenimento di acqua liquida su Enceladus, sia l'osservazione della superficie di Titano. Entrambe hanno costituito una svolta nella storia dell'esplorazione del Sistema Solare.

La durata della missione "nominale", ovvero il periodo di attività minimo previsto se la sonda e tutti gli strumenti in essa alloggiati funzioneranno correttamente, è di 4 anni. Ciò significa che la missione dovrebbe essere attiva fino alla metà del 2008. Come spesso accade in questo genere di missioni, però, è ritenuta molto probabile un'estensione delle attività di "*Cassini*" per un periodo che potrebbe arrivare ad un massimo di altri 4 anni dal termine della fase nominale.

## A.2) Gli obiettivi scientifici

### A.2.1) Atmosfera di Saturno

Lo studio dell'atmosfera di Saturno è uno degli obiettivi primari della missione *Cassini* in generale e di *VIMS* in particolare.

Precedenti passaggi ravvicinati di Saturno, durante le missioni *Voyager*, hanno permesso di rilevare solamente piccoli cambiamenti atmosferici, ciò a causa dei lunghi tempi che caratterizzano i fenomeni meteorologici di questo pianeta. Infatti le missioni *Voyager* hanno compiuto solamente dei passaggi a distanza ravvicinata da Saturno ma senza mettersi in orbita. Il tempo di osservazione è stato, quindi, limitato (Conrath e Pirraglia, 1983). Con la missione *Cassini*, invece, ci sono buone possibilità di ottenere dati più completi grazie ad una osservazione che si prolungherà nel tempo ed alla capacità di raccogliere dati sia da orbite polari che

equatoriali. Ciò permetterà di studiare variazioni latitudinali, longitudinali e temporali di molte strutture atmosferiche.

Tra le principali finalità di queste osservazioni troviamo:

determinazione della distribuzione verticale, al variare di latitudine e longitudine, di differenti specie gassose quali ammoniaca e acqua o di specie chimiche non in equilibrio, quali orto e para idrogeno. Variazione temporale delle suddette specie (Drossart, 1998; Gierasch, 1983).

Determinazione dei vincoli sui processi dinamici e chimici derivanti dalla variabilità spaziale e temporale dei fenomeni atmosferici. Studio delle relazioni tra le variazioni dei gas e le variazioni di opacità delle nubi (Taylor *et al.*, 1998). Determinazione delle variazioni temporali dei venti (Beebe, Ingersoll, Garneau, 1982).

Determinazione dei profili di temperatura nella stratosfera.

Studio dell'illuminazione e determinazione dei flussi; determinazione delle caratteristiche spettrali e della distribuzione 3-D nell'atmosfera.

Nella parte visibile dello spettro di Saturno sono state osservate tre specie gassose, in particolare: H<sub>2</sub>, NH<sub>3</sub> e CH<sub>4</sub> (Danehy, 1982; West *et al.*, 1982). Tra queste, H<sub>2</sub> è il più abbondante ma, sfortunatamente, la risoluzione spettrale di *VIMS-V* non permette di osservarlo. Le molecole di NH<sub>3</sub> risultano difficili da osservare a causa della bassa intensità delle bande di assorbimento dovuta alla ridotta concentrazione del gas. Tuttavia *VIMS-V* è stato in grado di rilevare la presenza di NH<sub>3</sub> nell'atmosfera di Giove (Coradini *et al.*, 2002; Brown *et al.*, 2002).

Il metano mostra forti bande di assorbimento nell'intervallo di lunghezze d'onda comprese tra 0.4  $\mu\text{m}$  e 0.6  $\mu\text{m}$ . Queste bande possono essere studiate utilizzando *VIMS-V* e, dai dati raccolti, risulta possibile derivare il contenuto di metano negli stati superiori dell'atmosfera (Karkoschka e Tomasko, 1989).

La capacità di *VIMS-V* di produrre immagini porterà alla creazione di mappe delle abbondanze delle specie volatili e, presumibilmente, darà un importante contributo allo studio delle strutture nuvolose presenti nell'atmosfera di Saturno.

In particolare sarà importante ricavare una dettagliata descrizione delle variazioni, in funzione dello spazio e del tempo, e le variazioni nella composizione delle nuvole stesse. Tutto ciò al fine di comprendere meglio la dinamica dei processi che caratterizzano l'atmosfera e di poter quindi creare un valido modello.

Utilizzando la massima risoluzione spaziale (25 km in alta risoluzione, 74 km in modo nominale alla minima distanza dal pianeta) sarà possibile studiare la stratificazione dei livelli più alti dell'atmosfera, profondità ottiche a differenti altezze e velocità dei venti.

### **A.2.2) Anelli di Saturno**

L'analisi spettroscopica degli anelli di Saturno nel visibile possono offrire la possibilità di studiare forme, dimensioni, composizioni e proprietà di diffusione delle particelle. Precedenti osservazioni hanno, infatti, messo in luce notevoli differenze tra i vari anelli (Porco et al, 1999; Porco, 1988). L'anello C, per esempio, si è rivelato molto più scuro degli anelli A e B (Estrada, 1996; Cooke, Nicholson, Showalter, 1991). *VIMS-V* darà un contributo alla comprensione dei motivi di queste differenze, chiarendo in che misura tali diversità dipendano dalla composizione e alle proprietà di diffusione delle particelle che formano gli anelli.

### **A.2.3) Titano**

La missione *Cassini – Huygens* rappresenta una grande opportunità per lo studio dell'atmosfera di Titano.

L'uso combinato del canale visibile e di quello infrarosso, permetterà di determinare la composizione e la distribuzione delle strutture atmosferiche. L'esistenza di alcune "finestre" spettrali nell'intervallo di lunghezze d'onda tra i 0.5  $\mu\text{m}$  e i 2  $\mu\text{m}$  darà la possibilità di ottenere una mappa, nel vicino infrarosso, della superficie di Titano. La risoluzione spaziale di 0.17 mrad nel visibile e di 0.50 mrad nell'infrarosso consentirà una risoluzione "al suolo" inferiore ad 1 km (Capaccioni et al., 1998).

Anche nel caso dell'osservazione di Titano la lunga durata delle osservazioni permetterà di determinare l'evoluzione temporale delle strutture atmosferiche e la loro dipendenza longitudine, latitudine ed altitudine. Le attività principali di *VIMS-V* saranno:

misurare la distribuzione alle varie altitudini delle specie attive di gas quali metano, etano, acetilene. Verrà condotto, inoltre, uno studio della loro distribuzione spaziale e delle variazioni nel tempo (Lebonnois, 2001; Gautier e Raulin, 1997).

Studiare la composizione e della formazione di aerosol e loro proprietà ottiche (Raulin et al., 2000).

Osservare e studiare la composizione della superficie (Gibbard et al., 1987)

Determinare le presenza di tramite lo spostamento di formazioni nuvolose. Dall'analisi della struttura delle bande di assorbimento atmosferico sarà possibile ricostruire il profilo di temperatura della stratosfera. (Griffith, 2000).

Studiare l'abbondanza del CH<sub>4</sub>. Nell'intervallo di lunghezze d'onda tra 0.6  $\mu\text{m}$  e 0.9  $\mu\text{m}$  il metano produce una serie di bande di assorbimento la cui profondità dipende dell'abbondanza del metano stesso negli strati più alti dell'atmosfera (Karkoschka e Tomasko, 1989). Assumendo un appropriato modello dell'atmosfera, della densità del metano in funzione dell'altezza, è possibile eseguire una inversione e determinare il profilo di concentrazione atmosferica.

Studiare le proprietà ottiche degli aerosol. Diverse osservazioni hanno messo in evidenza che gli effetti degli aerosol sono significativi particolarmente nel range tra  $0.3 \mu\text{m}$  e  $0.6 \mu\text{m}$  (Rages e Pollack, 1980). Questa osservazione evidenzia l'esistenza di precisi limiti alla loro dimensione ed anche alla distribuzione. Inoltre lo studio dello spettro nel visibile, a diversi angoli di fase, permetterà la determinazione delle proprietà di diffusione degli aerosol stessi.

### **A.2.4) Satelliti ghiacciati**

I satelliti ghiacciati di Saturno mostrano una struttura superficiale molto complessa che ne rispecchia la storia, caratterizzata da eventi che ne hanno modificato profondamente la superficie.

Nel caso di Enceladus, per esempio, si può parlare di attività tettonica che ha contribuito a cancellare la superficie originaria; Mimas e Thetys, invece, hanno subito l'impatto di corpi di grandi dimensioni come testimoniato dai crateri da impatto di diametri confrontabili con le dimensioni del satellite. Per i satelliti ghiacciati si è ipotizzato che essi siano stati distrutti e si siano riaccresciuti successivamente (McKinnon, 1985; Squyres e Croft, 1986).

La percentuale di crateri da impatto di questi satelliti varia considerevolmente da zona a zona. Ciò è probabilmente dovuto alle modificazioni subite dalla crosta primordiale, intensamente craterizzata per effetto di processi endogeni. Infatti, a causa del decadimento degli elementi radioattivi di lunga vita i satelliti sono stati caratterizzati, in un periodo più o meno lungo della loro storia, dalla presenza di un mantello fluido oppure dalla presenza di attività vulcanica di frattura. L'evidenza geologica dimostra che varie zone sono state ricoperte da materiale fluido, successivamente solidificatosi e che questi eventi sono avvenuti in una fase susseguente all'intenso bombardamento meteoritico che ha caratterizzato le fasi finali dell'accrescimento.

La presenza di materiali volatili quali ammoniaca, metano e CO, intrappolati nel ghiaccio d'acqua, è fondamentale per spiegare la complessa storia geofisica dei satelliti in questione (Coradini et al., 1995). Alcuni fenomeni di vulcanismo ghiacciato, che presumibilmente coinvolgono materiali ricchi di ammoniaca, sono presi in considerazione per spiegare l'abbondanza di pianure osservate sulla superficie di Encelado (Coradini et al., 1995).

Gli obiettivi fondamentali di *VIMS* saranno:

determinazione della composizione della superficie dei satelliti, utilizzando la massima risoluzione spaziale possibile (Denk, Jaumann, Neukum, 1993). La risoluzione spaziale, infatti, dipende fortemente dalla geometria e dalla velocità del passaggio ravvicinato. L'orbita della sonda non permetterà, però, sempre una buona osservazione dei satelliti a causa della distanza fortemente variabile dagli stessi tra un passaggio e l'altro e la risoluzione ottenibile sarà, comunque, già nel modo nominale, tra le 3 e le 10 volte migliore di quella ottenuta dalle sonde Voyager. Nelle migliori condizioni sarà inoltre possibile ottenere immagini spettrali di un intero emisfero dei satelliti per differenti angoli di osservazione. Si riuscirà così ad ottenere una maggiore conoscenza dei costituenti (minerali, ghiacci e composti organici) presenti, le loro concentrazioni e le dimensioni dei grani (Cruikshank, 1999), poiché queste ultime influiscono sulle proprietà fotometriche dei materiali.

Determinazione della composizione di ogni materiale scuro presente sui satelliti, con particolare attenzione a Giapeto. Questi materiali, in precedenti osservazioni, hanno mostrato, infatti, similarità, dal punto di vista chimico, con alcuni composti presenti nelle condriti carbonacee (Cruikshank, 1999). Si tenterà di mettere in relazione la composizione di questo materiale con gli studi riguardanti i corpi (meteoriti e comete), appartenenti al Sistema Solare, sui quali ci si aspetta di trovare materiali organici (Yang e Epstein, 1983). Questo tipo di studi potrebbe risultare notevolmente utile per la comprensione dell'origine del Sistema Solare.

Creazione di una mappa mineralogica delle superficie dei satelliti.

All'interno di questi obiettivi generali *VIMS-V* avrà dei compiti ben precisi. Questo strumento, infatti, è particolarmente adatto a tali indagini; la sua buona risoluzione spaziale e la risoluzione spettrale permetteranno di ottenere una dettagliata mappa mineralogica dei satelliti di Saturno. Particolare importanza sarà data ai seguenti studi:

studio del ghiaccio d'acqua e delle impurità in esso presenti. Il cammino ottico medio di un fotone all'interno del ghiaccio dipende dalla sua lunghezza d'onda; maggiore è il cammino ottico (e quindi minore l'assorbimento da parte del ghiaccio stesso), maggiore è la possibilità di rilevare eventuali impurità. Le righe di assorbimento a 0.81, 0.90, 1.04  $\mu\text{m}$ , tutte nel range di *VIMS-V*, sono particolarmente importanti poiché il coefficiente di assorbimento del ghiaccio è piccolo e ciò permette di rilevare con facilità anche piccole concentrazioni di impurità presenti (Clark e Lucey, 1984).

Studio della composizione e delle caratteristiche spettrali nel visibile e nella regione ultravioletta della superficie. I satelliti ghiacciati di Saturno non sono schermati dal bombardamento di particelle energetiche provenienti dallo spazio. Il ghiaccio d'acqua presente, sottoposto a tale bombardamento, subisce delle modificazioni nella microstruttura, mostrando così un notevole arrossamento dello spettro nella regione visibile e nell'UV, oltre ad una rugosità superficiale (Sack et al., 1991). Alcune caratteristiche specifiche potrebbero spiegare l'aumento di assorbimento nella regione UV dello spettro: per esempio la presenza di SO e di OH (derivante dalla fotodissociazione dell'acqua causata dai raggi ultravioletti del Sole) può determinare l'assorbimento a 280 nm (Sack et al., 1992; Johnson e Quickenden, 1997). Diverse precedenti osservazioni, inoltre, hanno evidenziato che l'esposizione di H<sub>2</sub>O, NH<sub>3</sub>, H<sub>2</sub>S e NH<sub>4</sub>HS a radiazioni UV diminuisce la riflettività spettrale sotto i 0.5  $\mu\text{m}$  (Lebofsky e Fegley, 1976).

Studio dei solidi organici non volatili e dei minerali. L'intervallo di lunghezze d'onda al quale è sensibile *VIMS-V* sarà molto utile per lo studio di composti di materiali organici macromolecolari quali materiali carboniosi anche se la loro bassa

riflettanza pone dei seri limiti alla loro rilevazione. La regione spettrale tra 0.8 e 1.05  $\mu\text{m}$  contiene le bande dei silicati che, mescolati tra loro o con altri minerali, possono subire delle modificazioni di forma e posizione (Pieters e Englert, 1993).

## A.3) VIMS-V

Lo strumento denominato *VIMS* (*Visible Infrared Mapping Spectrometer*) è stato progettato per fornire immagini multispettrali.

*VIMS* è in grado di coprire un intervallo spettrale che va da 0.30  $\mu\text{m}$  a 5.10  $\mu\text{m}$  ed è composto di due canali indipendenti, sensibili a due differenti intervalli di lunghezze d'onda: il primo, *VIMS-V*, opera tra 0.30  $\mu\text{m}$  e 1.05  $\mu\text{m}$ ; il secondo, *VIMS-R*, tra 0.85  $\mu\text{m}$  e 5.10  $\mu\text{m}$ . Il disegno, i materiali usati e le tecniche di fabbricazione impiegate hanno permesso di ottenere uno strumento compatto, leggero, a basso consumo e caratterizzato da un'elevata sensibilità.

### A.3.1) Caratteristiche tecniche

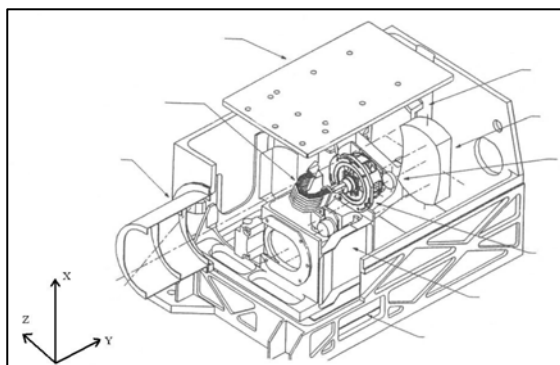


Figura A. 4 Schema interno di VIMS-V. Sono mostrati i componenti interni e la loro disposizione relativa.



*VIMS-V* è costituito da una “testa ottica” accoppiata elettricamente alla “scatola dell’elettronica”; entrambe le parti sono montate su un *pallet*, un sostegno di alluminio, situato accanto al canale infrarosso *VIMS-IR*.

La testa ottica, mostrata in Figura A.4, comprende due parti: il telescopio e lo spettrometro, sviluppati e verificati indipendentemente e parallelamente. Questi due componenti sono, a loro volta, composti da componenti separabili e collaudati individualmente. Un approccio modulare di questo genere è stato utilizzato per abbreviare i tempi di sviluppo e permettere di individuare più efficacemente i problemi durante le fasi di integrazione dei vari componenti e dei successivi test.

	<b>REQUISITI MINIMI</b>	<b>CAPACITA' DI VIMS-V</b>
<b>S/N</b>	100	Da 275 a 425
<b>IFOV</b>	500 $\mu$ rad	167 $\mu$ rad
<b>TFOV</b>	1.8° x 1.8°	2.4° x 2.4°
<b>Risoluzione spettrale (FWHM)</b>	7.5 nm	2nm
<b>Banda spettrale</b>	350÷1000 nm	300÷1050 nm

**Tabella A. 1** Requisiti scientifici necessari per la missione e valori ottenuti nella realizzazione di *VIMS-V*. **IFOV** (*Instantaneous Field Of View*) indica il campo di vista “istantaneo”, in altre parole quello del singolo pixel. **TFOV** (*Total Field Of View*) indica il campo di vista “totale”, vale a dire il campo di vista corrispondente a tutti i pixel letti.

### **A.3.2) Sistema ottico**

L’ottica di *VIMS-V* consiste in un telescopio di tipo Shafer, di uno spettrometro di tipo Offner e di un’unità di calibrazione.

*VIMS-V* è stato realizzato utilizzando esclusivamente specchi sferici. La radiazione esterna penetra tramite una fenditura di ingresso. La combinazione data dal telescopio Shafer più lo

spettrometro Offner è particolarmente efficace poiché permette di ridurre la radiazione fuori campo anche in presenza di un campo di vista grande, quale è il caso proprio di *VIMS-V*. In Fig.A.2 è riportato uno schema del sistema ottico completo: risulta evidente che tutti gli specchi, tranne due, hanno un taglio ret tangolare, ciò è dovuto alla necessità di ridurre il più possibile massa e volume di ogni componente. Solamente i due “*aperture stop*” hanno una forma circolare.

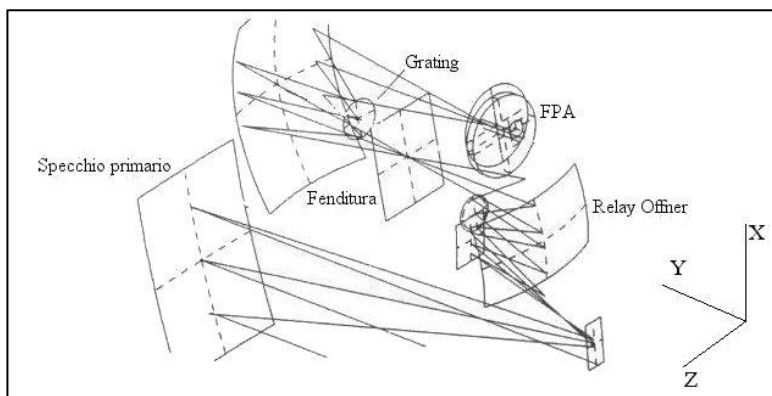


Figura A. 5 Schema ottico completo di VIMS-V.

La riduzione delle dimensioni degli specchi è stata possibile grazie al fatto che il campo di vista nella direzione perpendicolare alla slit, asse Z, è di soli  $167 \mu\text{rad}$ . Un'altra caratteristica del disegno ottico di *VIMS-V* è l'utilizzo del principio secondo il quale, quando gli elementi di un sistema ottico sono simmetrici, coma e distorsione possono essere praticamente eliminati in un apprezzabile campo. Nel caso di *VIMS-V* sia lo spettrometro sia il telescopio sono dotati di un *relay* Offner.

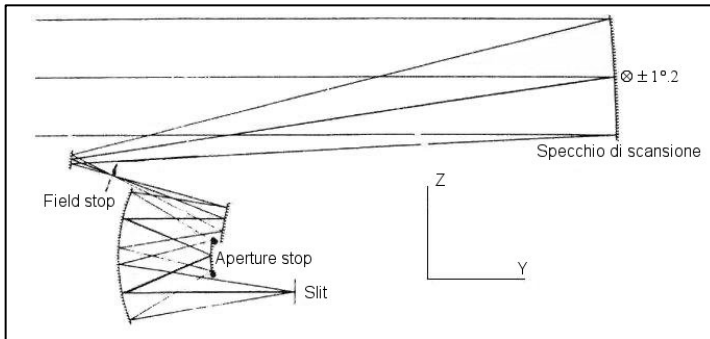
Tali *relay* sono simmetrici e coniugati l'uno rispetto all'altro, fatto questo che permette di avere un'ulteriore compensazione dell'aberrazione e una riduzione della radiazione fuori campo. Il *relay* dello spettrometro è posto, inoltre, all'esterno rispetto al *relay*

del telescopio; anche questo posizionamento è dovuto all'esigenza di contenere gli spazi.

Diametro della pupilla di ingresso	45 mm
F/#	F/3.2
Focale	143.2 mm
AΩ	$4.42 \cdot 10^{-7} \text{ cm}^2 \text{ ster}$
Angolo di scansione	$\pm 1.2^\circ$

**Tabella A. 2** Caratteristiche del sistema ottico di VIMS-V.

### A.3.3) Telescopio Shafer



**Figura A. 6** Schema del telescopio Shafer.

Il telescopio di tipo Shafer è una elaborazione dello schema studiato, in precedenza, da Burch il quale, nello schema originale, aveva previsto due specchi sferici concentrici. Sebbene corretto per aberrazione, coma e astigmatismo il telescopio Burch presenta, comunque, un campo convesso ed uno specchio secondario più grande del primario. Nello schema Shafer il secondario e il primario sono invertiti. Quest'ultimo è il più grande e l'immagine è virtuale e

dietro il secondario. Viene utilizzato, inoltre, un *relay* Offner, consistente in due specchi sferici e concentrici fuori asse che forniscono un'immagine piatta, reale, non ingrandita e priva di distorsioni.

Nella Figura A.6 viene mostrato uno schema bidimensionale del telescopio di *VIMS*: raggi luminosi quasi collimati entrano a  $0^\circ$ , o entro un angolo pari a  $\pm 1.2^\circ$ , e si riflettono sullo specchio primario, concavo, dirigendosi verso un piccolo e inclinato specchio piatto.

Il fascio convergente, a questo punto, viene diretto, attraverso il primo "*stop*", fino al secondario sferico e convesso. Dal secondario dello Shafer la luce entra nel *relay* Offner, andando a riflettersi sul primario, sferico e concavo, e poi sul secondario, sferico e convesso, che costituisce l'"*aperture stop*".

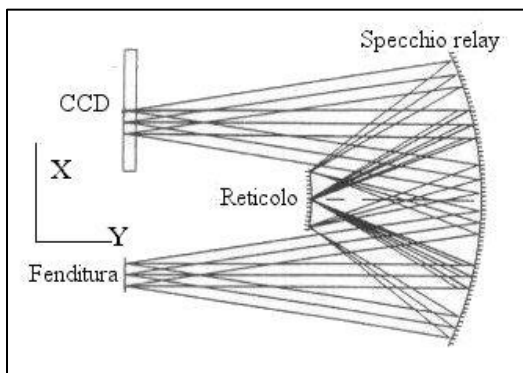
Successivamente la luce torna sul primario del *relay* e, finalmente, arriva sulla fenditura. Nel progettare questo telescopio sono stati adottati alcuni accorgimenti: per rendere il telescopio telecentrico è stato necessario posizionare la pupilla a 680 mm dal primario, frontalmente. Ciò ha causato un aumento di 15 mm nelle dimensioni dello specchio, lungo l'asse X, rispetto al caso in cui la pupilla fosse stata posta sul primario. La qualità dell'immagine risulta praticamente uniforme a tutti gli angoli e l'unica aberrazione presente è quella sferica. Nel caso di *VIMS-V* l'"*aperture stop*" si trova sul secondario del *relay* Offner, abbastanza vicina al centro di curvatura da ridurre considerevolmente le aberrazioni.

### **A.3.4) Spettrometro Offner**

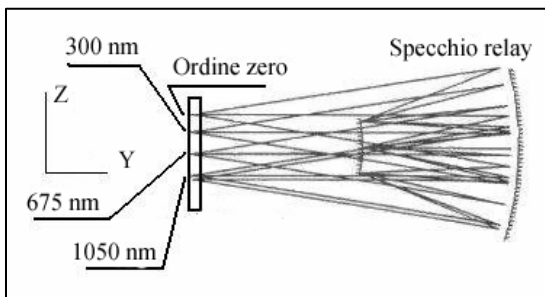
Lo spettrometro Offner offre prestazioni notevoli. Molti spettrometri si basano su ingombranti sistemi ottici comprendenti collimatori ed elementi riflettenti per giungere ad avere notevoli prestazioni sul piano della risoluzione. L'Offner è, invece, costituito da due soli elementi. La struttura è praticamente la stessa del *relay* di cui abbiamo parlato poc'anzi, con una differenza: al posto dello

specchio secondario, convesso e sferico, in questo caso si trova un reticolo con caratteristiche geometriche simili.

Nelle Figure A.7 e A.8 è illustrata la stessa situazione da due differenti punti di vista. Viene visualizzato il percorso di tre raggi luminosi che entrano dalla fenditura secondo gli angoli  $+1.2^\circ$ ,  $0^\circ$ ,  $-1.2^\circ$ . La luce, dopo aver subito una prima riflessione sul primario, arriva sul reticolo dove viene diffranta; da qui torna sul primario che la manda, infine, sul rivelatore. In Fig.A.7 la scena è vista lateralmente, in Figura A.8 è vista dall'alto.



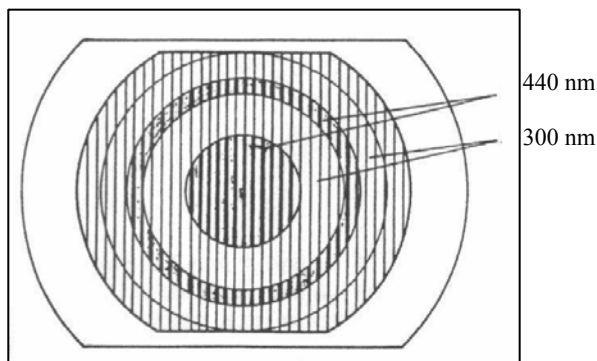
**Figura A. 7 Percorso della radiazione dalla fenditura alla CCD, vista dall'alto.**



**Figura A.8 Percorso della radiazione dalla fenditura alla CCD, vista laterale.**

### A.3.5) Il reticolo

Il reticolo utilizzato in *VIMS-V* è stato realizzato dalla Zeiss ed è inciso olograficamente su un supporto convesso. La distanza tra le incisioni, se la si considera proiettata lungo una corda della superficie sferica, è costante. Si è scelto di creare un profilo delle incisioni rettangolare poiché, rispetto a tutti gli altri tipi di profilo, questo è quello che presenta le minor difficoltà di realizzazione su una superficie convessa. Questo profilo, inoltre, riduce la radiazione fuori campo al prim'ordine a meno di una parte su diecimila; al second'ordine a meno dell'1% e al terz'ordine a meno del 10%. Per *VIMS-V* questa è proprio la luce che deve essere bloccata. Un altro

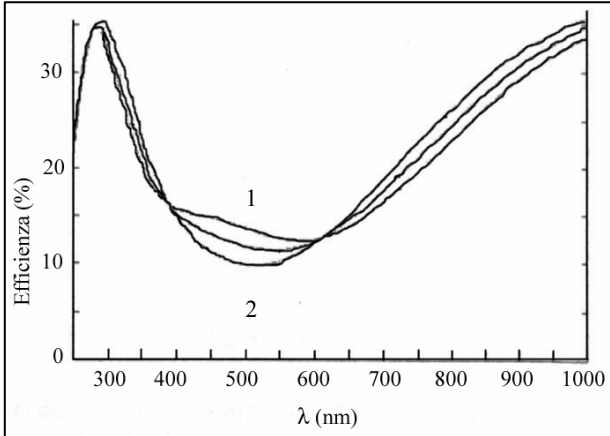


**Figura A. 9** Schema del reticolo di diffrazione di *VIMS-V*. Si distinguono le due zone di differenti profondità delle incisioni.

vantaggio del profilo laminare è dato dalla possibilità di correggere l'efficienza, lungo lo spettro, a seconda delle necessità della missione.

A causa dell'andamento della curva di emissione dello spettro solare (valori bassi agli estremi ed alti nella zona centrale, in assenza di un qualche sistema di compensazione) si avrebbero valori del segnale molto alti nella porzione centrale dello spettro e troppo bassi, anche sotto la soglia di rumore, nelle zone laterali. Per rimediare a questo problema si è operato in modo da avere una risposta complessiva dello strumento la più piatta possibile, così da

avere valori uniformi del segnale lungo tutto l'intervallo spettrale. Il metodo studiato per realizzare questo obiettivo consiste nell'adottare



**Figura A. 10** Curve di efficienza del reticolo di diffrazione. La curva 1 è relativa alla profondità maggiore, la 2 alla profondità minore. La curva centrale è l'efficienza complessiva.

due differenti profondità di incisioni in regioni adiacenti del reticolo. In pratica si è suddiviso il reticolo in anelli concentrici nei quali si sono incise, alternativamente, incisioni di profondità minore, 300 nm, e maggiore, 440 nm. Uno schema del reticolo è mostrato in Figura A.9.

Una complicazione nasce dalla natura stessa del fenomeno della diffrazione. L'equazione mostrata sotto descrive tale fenomeno; in essa compaiono diverse grandezze:  $d$  è la distanza tra le incisioni del reticolo;  $\theta$  è l'angolo tra la normale al reticolo e la direzione di propagazione del raggio di luce incidente;  $m$  è l'intero che contraddistingue i diversi ordini formati;  $\lambda$  è la lunghezza d'onda della luce diffranta.

$$d \sin \theta = m \lambda$$

Tale equazione prevede la creazione di un numero infinito di ordini, individuati, come già detto, dal valore  $m$ , che vanno ad incidere sul rivelatore. *VIMS-V* è stato studiato per utilizzare solo il primo ordine, gli altri costituiscono, invece, un disturbo che deve essere eliminato. Più avanti sarà spiegato come tale inconveniente è stato risolto.

### **A.4.6) La CCD**

La CCD utilizzata per *VIMS-V* è un dispositivo NMOS, del tipo ad “trasferimento d'immagine”, e richiede un'illuminazione frontale. Dopo l'acquisizione, il contenuto della zona illuminata, costituita da una matrice di 512 per 256 elementi, viene trasferito in quella che è chiamata “zona di memoria”, dove avviene la lettura. Entrambe le zone sono orientate in maniera tale da avere lo spettro lungo il lato maggiore ed il trasferimento dei fotoni, dall'una all'altra, avviene lungo la colonna.

La CCD è stata costruita in maniera tale da ridurre la corrente di buio ad un valore di meno di 15 elettroni/s a  $-20^{\circ}$ . Il chip della CCD è alloggiato su uno zoccolo isolante, caratterizzato da un basso degassamento, composto di  $\text{BeO}_2$ . Entrambi sono inseriti in un contenitore di kovar che subisce solamente piccole dilatazioni al variare della temperatura e che, inoltre, ha la funzione di gabbia di Faraday e costituisce, quindi, una protezione per la CCD stessa.

La CCD è ricoperta da una finestra in *suprasil*, spessa 178  $\mu\text{m}$ , e da un filtro ottico. Come è stato spiegato in precedenza, esiste un problema dovuto alla creazione, nel processo di dispersione della luce da parte del reticolo, di ordini superiori, i quali, se non bloccati, andrebbero ad incidere sul rivelatore dando luogo a notevoli disturbi.

Per impedire ciò è stato predisposto il già citato filtro ottico, il quale è costituito da due zone separate da un divisore largo 40  $\mu\text{m}$ . La prima zona taglia le frequenze tra 150 e 276 nm e quindi blocca, per esempio, la radiazione al second'ordine tra 300 e 552 nm. La seconda taglia le  $\lambda$  tra 275 e 526 nm, fermando il second'ordine tra i



600 nm e i 1104 nm. Tra le due zone esiste, come già detto, un divisore che blocca la luce diffusa alla giunzione tra i due filtri. Questa “zona morta” è posizionata in corrispondenza del punto nel quale va ad incidere la radiazione con  $\lambda$  pari a 600 nm, una lunghezza d’onda per la quale non ci si aspettano righe importanti per la missione di *VIMS-V*. La finestra si trova tra il filtro e la CCD e ha il compito di ridurre la diffusione verso la CCD stessa ed il filtro ottico. Un’altra funzione della finestra è quello di limitare la sublimazione del deposito chimico della CCD nel caso in cui la temperatura dovesse superare gli 80 °C. Solo le colonne in corrispondenza delle lunghezze d’onda tra 300 e 490 nm sono trattate con il *coating*, un deposito chimico, necessario poiché il rivelatore non è sensibile alle suddette lunghezze d’onda.

Il *coating* utilizzato risolve tale problema assorbendo le frequenze che non sarebbero rivelate dalla CCD ed emettendo, di conseguenza, radiazione ad altre frequenze le quali, invece, generano la produzione di foto-elettroni da parte del rivelatore. Poiché il materiale del quale è costituita la CCD ha un’efficienza minore dello 0.1% per le  $\lambda < 360$  nm, le colonne non trattate con il già citato deposito chimico non sono sensibili alla radiazione UV e quindi non c’è pericolo di produzione di elettroni da parte di questa radiazione nelle colonne destinate alla rivelazione delle lunghezze d’onda nel visibile. Dalla parte opposta, rispetto ai filtri ottici, della finestra, si trovano due segmenti anti-riflessione ottimizzati per le lunghezze d’onda 350 nm e 1050 nm. La giunzione tra i due segmenti è larga 10  $\mu\text{m}$  e si trova in corrispondenza della divisione tra i due filtri ottici



## **Bibliografia**

ADAMS J. B., McCORD T. B. e PIETERS C. M. (1986). Apollo 16 landing site absolute reflectance.

<http://www.planetary.brown.edu/pds/database.html>

Autori Vari. (2001). Introduction to IDL. R.S.I. Inc.

Autori Vari. (2001). Intermediate Programming and Analysis in IDL. R.S.I. Inc.

Autori Vari (2004). PHP5 & MySQL, McGraw-Hill

Autori Vari (2001). Practical Java2. Jackson Libri

Autori Vari (2002). Java2 – La guida completa. McGraw-Hill

Autori Vari (2004). Javascript – La guida. McGraw-Hill

Autori Vari (2001). Linux – La guida completa. McGraw-Hill

BEEBE R., INGERSOLL A. P. e GARNEAU G. (1982). A Summary of the Winds of Jupiter and Saturn. *Bulletin of the American Astronomical Society*, 14:722.

BERRY R. e BURNELL J. (2000). The handbook of astronomical image processing / Willmann-Bell

CLARK R. N. e LUCEY P. G. (1984). Spectral properties of ice particulate mixtures and implications for remote sensing. 1. Intimate mixtures. *Journal Geophysical Research.*, 89:6341.

CONRATH B.J. e PIRRAGLIA J. A. (1983). Thermal structure of Saturn from Voyager infrared measurements: implications for atmospheric dynamics. *Icarus*, 53:286.

COOKE M. L., NICHOLSON P. D. e SHOWALTER M. R. (1991). Photometric studies of Saturn's C ring. *Bulletin of the American Astronomical Society*, 23(3):1180.

CRUIKSHANK D. P., ROUSH T. L., OWEN T. C., GEBALLE T. R., DALLE ORE C. M., KHARE B. N. e DE BERGH C. (1999). Saturn's Icy Satellites: Infrared Reflectance Spectra of Rhea, Tethys, Dione, and Iapetus at  $\lambda \geq 2.5 \mu\text{m}$ . *American Astronomical Society, DPS meeting n.31, #03.02*

DANEHY R. G. (1982). Detection of the 5520Å NH<sub>3</sub> band and the 8260Å S(O) 3-0 H<sub>2</sub> dipole band on Saturn. *Bulletin of the American Astronomical Society*, 14:731.

DENK T., JAUMANN R. e NEUKUM G. (1993). Comparison of the Surface Composition on Different Regions of Saturn's Satellites. *American Astronomical Society, 25th DPS Meeting, n.31.10-P; Bulletin of the American Astronomical Society, 25:1114.*

DROSSART, P. (1998). Saturn Tropospheric Water Measured with ISO/SWS. *American Astronomical Society, DPS meeting n.30, #27.01; Bulletin of the American Astronomical Society*, 30:1060.

ESTRADA P. R. e CUZZI J. N. (1996). Voyager Observations of the Color of Saturn's Rings. *Icarus*, 122:251.

GAUTIER D. e RAULIN F. (1997). Chemical Composition of Titan's Atmosphere. *Huygens: Science, Payload and Mission, Proceedings of an ESA conference. Edited by A. Wilson*, 359.

GIBBARD S., MACINTOSH B., GAVEL D., MAX C., DE PATER I., YOUNG E. e MCKAY C. (1997). Very High Resolution IR Imaging of Titan's Surface and Atmosphere. *American Astronomical Society, DPS meeting n.29, #33.01; Bulletin of the American Astronomical Society*, 29:1037.

GIERASCH P. J. (1983). Dynamical Consequences of Orthohydrogen-Parahydrogen Disequilibrium on Jupiter and Saturn. *Science*, 219:847.

GRIFFITH C. A. (2000). Titan's atmosphere (clouds and composition): new results. *Highlights of Planetary Exploration from Space and from Earth, 24th meeting of the IAU, Joint Discussion 12, August 2000, Manchester, England*.

JOHNSON R. E. e QUICKENDEN T. I. (1997). Photolysis and Radiolysis of water ice on outer solar system bodies. *J. Geophys. Res.*, 102(E5):10985.

KARKOSCHKA E. e TOMASKO G. M. (1989). Methane-Band Spectroscopy of Saturn. *Bulletin of the American Astronomical Society*, 21:952.

LEBOFSKY L. A. e FEGLEY M. B. (1976). Laboratory reflection spectra for the determination of chemical composition of icy bodies. *Icarus*, 28:379.

LEBONNOIS S., TOUBLANC D., HOURDIN F. e RANNOU P. (2001). Seasonal Variations of Titan's Atmospheric Composition. *Icarus*, 152(2):384.

McKINNON W. B. (1985). Geology of icy satellites. In *Ices in the Solar System*, ed. J. Klinger, D. Benest, A. Dollfuss e R. Smoluchowski, 829. NATO-ASI Series v. 156, Reidel Pub. Comp.

McCORD T. B., CHARETTE M. P., JOHNSON T. V., LEBOFSKY L. A. e PIETERS C. M. (1972). Spectropotometry (0.3 to 1.1  $\mu\text{m}$ ) of visited and proposed Apollo lunar landing sites. *The Moon*, 5:52.

McCORD T. B. e ADAMS J. B. (1973). Progress in remote optical analysis of lunar surface composition. *The Moon*, 7:453.

McCORD T. B., CHARETTE M. P., JOHNSON T. V., LEBOFSKY L. A. e PIETERS C. M. (1995). Lunar spectral types. *Journal Geophysical Research*, 77(8):1349.

PIETERS C. M. (1986). Composition of the lunar highland crust from near infrared spectroscopy. *Reviews of Geophysics*, 24(3):557.

PIETERS C. M. e ENGLERT P. A. J. (1993). *Remote geochemical analysis: elemental and mineralogic composition*. Cambridge University Press.

PORCO C. C. (1988). Observational studies of Saturn's rings. *NASA Tech. Memo.*, NASA TM-4041:11.

PORCO C. C. , PANTAZOPOULOU M. J. , RICHARDSON D. C. , QUINN T. e KEHOE T. J. J. (1999). Light scattering in planetary rings: the nature of Saturn's particle disk. *Bulletin of the American Astronomical Society*, 31(4):1140.

RAGES K. e POLLACK J. B. (1980). Titan aerosols: optical properties and vertical distribution. *Icarus*, 41:119.

RAULIN F., COLL P., RAMIREZ S., NAVARRO-GONZALEZ R., DA SILVA A., LAFAIT J., LABORATOIRE INTERUNIVERSITAIRE DES SYSTEMES ATMOSPHERIQUES TEAM, INSTITUTO DE CIENCIAS NUCLEARES TEAM e LABORATOIRE D'OPTIQUE DES SOLIDES TEAM. (2000). Titan's aerosols : chemistry and optical behaviour. *American Astronomical Society, DPS meeting n.32, #17.09*

SACK N. J., BORING J. W., JOHNSON R. E., BARAGIOLA R.A. e SHI M. (1991). Alteration of the UV-visible reflectance spectra of water ice by bombardment. *J. Geophys. Res.*, 96(E2):17535.

SACK N. J., BORING J. W., JOHNSON R. E. e BARAGIOLA R.A. (1992). The effect of magnetospheric ion bombardment on the reflectance of Europa's surface. *Icarus*, 100:534.

SQUYRES S.W. e CROFT S. K. (1986). The tectonics of icy satellites. In *Satellites*, ed J. A. Burns e M. S. Matthews, 293. The University of Arizona Press, Tucson.

TAYLOR F. W., CALCUTT S. B., IRWIN P. G. J., NIXON C. A., READ P. L., SMITH P. J. C. e VELLACOTT T. J. (1998). Investigation of Saturn atmosphere by Cassini. *Planetary and Space Science*, 46(9-10):1315.

WEST R. A., TOMASKO M. G., SMITH B. A., WIJESINGHE M. P., DOOSE L. R., REITSEMA H. e LARSON S. (1981). Methane Band Images of Saturn. *Bulletin of the American Astronomical Society*, 13:723.

YANG J. e EPSTEIN S. (1983). Interstellar organic matter in meteorites. *Meteoritics*, 18(4): 429.