UNIVERSITÀ DEGLI STUDI DI NAPOLI
**"PARTHENOPE"**

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica


Elaborato di Laurea

# Sviluppo di Componenti Grid per la Condivisione di Strumenti di Acquisizione Dati

Relatore:
Prof. Raffaele Montella

Presentata da:
Roberto Di Lauro
LI/534

Anno Accademico 2007-2008

# Abstract

L'obiettivo di questo elaborato di Tesi di Laurea è la creazione di un sistema software per la condivisione sicura e l'aggregazione di strumenti d'acquisizione dati, geograficamente distribuiti, per applicazioni scientifiche ed ingegneristiche. A tale fine è stata scelta la tecnologia messa a disposizione dalle griglie computazionali *web-service based* adoperando il software toolkit Globus Toolkit 4. Il primo obiettivo è stato quello di sviluppare un framework per la virtualizzazione di strumenti di acquisizione dati al fine di creare un sistema di interfaccia comune incentrato su componenti standard interoperabili. Successivamente è stato progettato un *grid-web-service* che combinato al framework consente l'accesso, il controllo e l'acquisizione dati provenienti da strumenti in un ambiente sicuro e condiviso qual è la Grid da cui è ereditata anche la capacità di pubblicizzare le risorse disponibili, sia in quanto strumenti sia in quanto dati acquisiti, su appositi servizi indice adoperati dai servizi di resource brokering. La soluzione proposta è estremamente innovativa in quanto consente allo sviluppatore di usufruire di un supporto concreto, in termini di adattabilità e riusabilità, nella virtualizzazione e nell' integrazione di strumenti su Grid. Infine, è stato sviluppato come *Test-Case* un Osservatorio Astronomico Virtuale il quale dimostra che gli obiettivi prefissati sono stati raggiunti con successo. Le componenti sviluppate in questo lavoro di Tesi contribuiscono alla nuova frontiera di ricerca riguardo l'integrazione di strumenti scientifici in ambiente Grid ormai vista come tecnologia per l'interoperabilità di componenti distribuite piuttosto che mero mezzo aggregativo di risorse computazionali ormai disponibili in quantità e a basso costo. Questo lavoro di tesi è così suddiviso: il capitolo 1 rappresenta una

rapida introduzione a quanto effettuato oltre che un inquadramento scientifico e metodologico agli argomenti trattati con particolare riferimento alla tecnologia grid e la sua evoluzione con cenni sulla differenze e similitudini fra questa tecnologia e l'emergente cloud computing. Nel capitolo 2 è descritto lo stato dell'arte nel campo di questo tipo di applicazioni facendo riferimento a progetti analoghi e ad una delle più significative applicazioni di questa tipologia: il Large Headron Collider del CERN. Nel capitolo 3 sono descritti gli strumenti adoperati con particolare riferimento a quelli relativi alle tecnologie grid e per la gestione di dati multidimensionali. Nel capitolo 4 è descritto quanto sviluppato da zero in questo lavoro di tesi: l'abstract instrument framework e l'instrument service. Nel capitolo 5 è descritta l'applicazione usata come banco di prova e relativa all'implementazione di un laboratorio virtuale per applicazioni di astronomia. Infine, nel capitolo 6, oltre alle conclusioni, alcuni cenni a possibili sviluppi futuri.

# Indice

# Capitolo 1

# Introduzione

Gli esperimenti scientifici odierni hanno raggiunto livelli di complessità tali da rendere fondamentale l'utilizzo di molteplici strumenti tecnologicamente all'avanguardia.

Come progetto di tesi di laurea è stato sviluppato un insieme di soluzioni software per la gestione, attraverso la tecnologia offerta dal Grid Computing, di dispositivi d'acquisizione dati geograficamente distribuiti [1].

L'esperienza maturata riguardo queste tecnologie è relativa all'integrazione di strumenti geograficamente distribuiti in una griglia computazionale tramite un interfaccia basata su (grid) Web Service. L'implementazione e tutte le problematiche inerenti al controllo ed all'acquisizione on-line dei dati prodotti dagli strumenti devono essere del tutto trasparenti all'utente mediante l'utilizzo di tecniche per la virtualizzazione delle risorse seguendo un approccio il più possibile orientato alla futura frontiera del Cloud Computing [10,11] (decomponentizzazione, approccio multi-tier, separazione delle componenti computazionali da quelle di memorizzazione). Mediante una computing grid è possibile avere accesso agli strumenti d'acquisizione e a

ciascuno dei sensori che li compongono in modo tale che possono essere gestiti come risorse al pari di quelle per la memorizzazione o per il calcolo ad alte prestazioni.

L'integrazione di strumenti di acquisizione dati in ambiente di griglia è una sfida strategicamente rilevante poiché una risorsa rara, costosa e di difficile gestione è adoperata in modo più efficiente ed efficace, grazie ad una migliore interazione con altri tipi di elementi di calcolo e di memorizzazione massimizzando, così, il *throughput* dell'intero sistema.

L'uso di tecnologia di griglia per controllare strumenti di acquisizione e recuperare i dati acquisiti implica la necessità di sviluppare una metodologia standard di interfaccia tra i diversi tipi di strumenti hardware. Durante lo sviluppo del progetto di tesi di laurea è stato implementato il framework **AbstractInstrument** (AIF), usato per la virtualizzazione degli strumenti, mediante l'uso di interfacce standard che forniscono un alto livello di interazione comune a tutti gli strumenti. Grazie a questo approccio, qualsiasi strumento può essere gestito tramite un device driver di alto livello: questo rappresenta un primo elemento di innovazione.

Perché uno strumento virtualizzato sia gestibile tramite griglia computazionale, è stato sviluppato, facendo uso del Globus Toolkit versione 4 (GT4), sviluppato presso la Mathematics and Computer Science divisione dell'Argonne National Laboratory (MCS/ANL) ed il Computation Institute dell'University of Chicago (CI-UoC) istituzioni scientifiche di rilevanza mondiale con cui sono in corso collaborazioni, il *secure grid-web-service* **Instrument Service** (IS) che, attraverso le funzionalità offerte dall'AIF permette l'accesso, il controllo e la condivisione degli strumenti virtualizzati attraverso la

Grid. L'IS interfaccia un qualsiasi strumento alla griglia pubblicando automaticamente sull'Index Service, componente standard del GT4, i metadata relativi a ciascuno strumento ed eventualmente i valori delle misure correntemente acquisite dai sensori. Questa funzionalità, completamente configurabile in termini di informazioni pubblicate, consente al Resource Broker Service (RBS), componente sviluppata presso il Dipartimento di Scienze Applicate, di cercare strumenti al pari di altre risorse di griglia mediante query scritte con il linguaggio di descrizione di risorse ClassAd, adoperato in Condor e in gLite e considerato lo standard de facto in questo tipo di applicazioni.

Durante lo sviluppo di AIF prima e di IS dopo sono stati fatti numerosi test su strumenti di tipologie differenti come centraline meteo, webcam, GPS, schede TV. Tuttavia, al fine di mostrare quanto effettivamente possibile mediante l'uso delle componenti sviluppate, grazie alla collaborazione con l'Osservatorio Astronomico di Capodimonte (OAC), nella persona dell' Ing. Enrico Cascone dell'Istituto Nazionale Astrofisica (INAF), ottenuta grazie alla Prof.ssa Alessandra Rotundi del Dipartimento di Scienze Applicate dell'Università degli Studi di Napoli *"Parthenope"*, è stato sviluppato un Laboratorio Virtuale dedicato ad applicazioni di tipo astronomico (AVL).

Attualmente AVL supporta telescopi robotizzati e centraline meteorologiche che possono essere utilizzate in applicazioni di griglia computazionale integranti anche altre componenti come i servizi per la distribuzione di dati ambientali multidimensionali.

Sul sito internet http://lmncp.uniparthenope.it/grid/IS è possibile effettuare il download delle componenti software sviluppate.

## 1.1 Il grid computing

Con le tecnologie odierne per i vari enti coinvolti nelle IT risulta particolarmente svantaggioso investire le proprie risorse economiche in sistemi di calcolo complessi dall'elevato costo di produzione e di gestione.

Di conseguenza, nella ricerca di soluzioni alternative per far fronte a questa tipologia di problematiche, oggi l'attenzione è rivolta alle tecnologie basate sulle Griglie Computazionali [1,2,5].

Mediante un approccio basato sul Grid Computing enti diversi con scopi comuni o anche differenti possono instaurare rapporti di collaborazione mediante la realizzazione di *Virtual Organization* (VO) [12]. Le VO consentono una condivisione reciproca delle risorse computazionali che ogni singola entità possiede. In questo modo per tutte le organizzazioni coinvolte è possibile accedere alle risorse condivise e quindi disporre facilmente ed in modo economicamente vantaggioso le risorse computazionali necessarie per il raggiungimento dei propri scopi. Il Grid Computing offre la possibilità di integrare e condividere risorse computazionali geograficamente distribuite. Utilizzando un approccio basato sulle griglie computazionali è possibile avere accesso in modo omogeneo a risorse eterogenee sia da un punto di vista hardware che software, realizzando delle applicazioni indipendenti dalle piattaforme operative, progettate e implementate in modo da essere compatibili con la Grid.

L'idea centrale del Grid Computing è l'elaborazione intesa come servizio. Ad esempio, oggi possiamo attingere a risorse primarie come l'acqua potabile o l'elettricità senza considerare da dove queste risorse vengano prelevate. Secondo questo paradigma la distribuzione geografica delle ri-

sorse computazionali non condiziona in modo significativo il risultato delle elaborazioni.

Il Grid Computing è una tecnologia ideata per consentire l'accesso e la condivisione a risorse di calcolo distribuite, inoltre possiamo condividere grandi quantità di dati e gestire i dispositivi di acquisizione utilizzati.

L'unione tra le Tecnologie Web con la prima generazione di Grid Computing ha spostato l'attenzione nelle collaborazioni tra differenti organizzazioni nel rispetto della sicurezza individuale. La Grid, inoltre, facilita l'individuazione, il recupero, l'elaborazione e la distribuzione di dati distribuiti attraverso lo sviluppo di software che permettono il processo di astrazione di dati eterogenei e dei relativi meta-dati.

Mediante il processo di virtualizzazione delle risorse il Grid Computing si è rapidamente evoluto dando luogo ad una nuova concezione nota oggi con il nome di *Cloud Computing*. Il *Cloud Computing* [10,14] permette un accesso alle risorse di calcolo di tipo astratto e altamente scalabili. Le Tecnologie di virtualizzazione comprendono una varietà di meccanismi e tecniche affinché l'utente non percepisca tutte le problematiche riguardanti le componenti hardware e software delle risorse condivise. Nella comunità scientifica del calcolo distribuito ad alte prestazioni è in corso un acceso dibattito riguardo alla relazione fra il grid ed il cloud computing. Un buon compromesso è rappresentato dal considerare le tecnologie grid come una componente di basso livello su cui, grazie al supporto alla virtualizzazione e paravirtualizzazione delle risorse di calcolo offerto da software come XEN [15], KVM e VMware [10], sono costruite applicazioni di tipo cloud (cloud basate su grid). In questo modo, attraverso gestori di nuvole di calcolo come Nimbus [10] (utilizzato

da Amazon EC2), OpenNebula [11] ed Haizea [13], è possibile effettuare deployment dinamici di cluster virtuali (cluster on demand). Tuttavia, questa stessa tecnologia permette di fare deployment dinamico di un'intera griglia computazionale (grid on cloud). Il reale punto di discussione è lo sviluppo di linee guida che determina l'efficacia di un'applicazione distribuita su griglia o su nuvola ("gridable" or "cloudable" applications), mentre un settore di ricerca interessante è rappresentato dall'uso dell'approccio cloud nel calcolo ad alte prestazioni dove la comunicazione fra le CPU rappresenta il collo di bottiglia che limita le performance.

# Capitolo 2

# Stato dell'Arte

Sebbene la condivisione degli strumenti di acquisizione dati sia stata indicata come una delle killing application delle griglie computazionali sin dagli albori di questa tecnologia, non molto è stato effettivamente sviluppato per la produzione di massa fatta eccezione per grandi progetti come il Large Headron Collider del CERN e tentativi più o meno riusciti di componenti a basso livello.

## 2.1 Large Hadron Collider

Il LHC (*Large Hadron Collider*) [18] sviluppato dal CERN di Ginevra è il più grande strumento scientifico del pianeta e produce annualmente circa 15 Petabytes (15 milioni di GigaBytes).

I dati prodotti verranno analizzati dai ricercatori in tutto il mondo. L'obiettivo del progetto Worldwide LHC Computing Grid (WLCG) è quello di costruire un' infrastruttura per lo storage e l' analisi dei dati, accessibile all'intera comunità *High Energy Physics*.

Figura 2.1: Il Large Hadron Collider (LHC)

I dati provenienti dagli esperimenti saranno distribuiti in tutto il mondo, secondo un modello a 4 livelli. Un primo backup, sarà memorizzato su nastro al CERN, il "Tier 0", successivamente questi dati saranno distribuiti a grandi centri di calcolo con sufficiente capacità di storage e un supporto round-the-clock per la griglia "Tier-1".

I centri "Tier-1" rendono disponibili i dati ai centri "Tier-2", ciascuno costituito da uno o più impianti informatici, in grado di memorizzare sufficientemente i dati e fornire un'adeguata potenza di calcolo per analisi specifiche. Singoli scienziati potranno accedere a queste strutture attraverso le risorse di calcolo "Tier-3", che consistono ad esempio in cluster di dipartimenti universitari o singoli PC.

Il *Worldwide LHC Computing Grid* aggrega risorse computazionali provenienti da più di 140 centri di calcolo in 33 paesi, al fine di sfruttare la potenza di 100.000 CPU per elaborare, analizzare e archiviare i dati prodotti, rendendoli accessibili in egual misura a tutti i partner, indipendentemente dalla loro ubicazione fisica. L'accesso ai dati è previsto per oltre 5000 scienziati in circa 500 istituti di ricerca e università di tutto il mondo che partecipano

alla sperimentazione. Inoltre tutti i dati devono essere disponibili oltre i 15 anni di vita stimati per LHC.

Un tradizionale approccio sarebbe quello di centralizzare e localizzare tutte queste capacità; Tuttavia nel caso di LHC è stato scelto un nuovo modello di distribuzione mondiale (computing/data-grid) per lo storage e l'analisi dei dati che fornisce numerosi vantaggi.

*LHC Computing Grid* (LCG) fornisce Tools e Computing Services per aiutare gli utenti nelle loro attività di ricerca. Gli obiettivi del progetto LCG includono:

- Sviluppo di componenti per il supporto alle applicazioni, come librerie, tools per la gestione e l'accesso ai dati, adeguamento dei framework per le simulazioni, interfacce batch e un sistema di analisi interattiva in ambiente Grid.

- Sviluppo e *deployement* dei servizi Grid basati su modello di griglia distribuita utilizzando risorse provenienti da più di un centinaio di centri di calcolo in tutto il mondo.

- Gestione degli utenti e dei loro diritti in un ambiente Grid internazionale, eterogeneo e distribuito.

## 2.2 CIMA

Il progetto CIMA [19] sviluppato dal *National Science Foundation Middleware Iniziative*, ha l'obiettivo di facilitare la realizzazione di strumenti integrati in grado di essere gestiti mediante i servizi offerti da una Griglia Computazionale. Per raggiungere questo obiettivo, CIMA è stato sviluppato

mediante un approccio orientato ai Web Services e quindi applicando gli standard OGSA (Open Grid Services Architecture) e CCA (Common Component Architecture).

Risulta fondamentale ai fini della progettazione la differenza tra due scenari:

1. Accesso remoto
2. Operazioni distribuite

L'accesso da remoto consente ad un utente della Griglia di usare una rete di strumenti geograficamente distribuita dalla propria postazione di lavoro, mentre per operazioni distribuite vogliamo intendere la possibilità di condividere le funzioni degli strumenti distribuiti attraverso una *Virtual Organization*. Mediante l'approccio utilizzato per il progetto CIMA si ottengono i seguenti vantaggi:

- Flessibilità nell'invio e ricezione dei dati.
- Adattabilità a nuovi strumenti.
- Integrazione degli strumenti, risorse di calcolo e memorizzazione come nel caso dei Web/Grid Service.
- Connessione ai Web Service tramite plug-in.

Tra i requisiti importanti nella progettazione di applicazioni basati sul progetto CIMA troviamo : l'interoperabilità ovvero la capacità di cooperare e di scambiare informazioni o servizi con altri sistemi; trasferimento dei dati efficiente tramite tecniche di buffering che risultano molto utili quando il data-rate degli strumenti è superiore alla capacità di trasferimento della rete

e infine l'accurata descrizione di ogni funzionalità dello strumento da implementare, in modo da sviluppare un modello di operazioni da un minimo di conoscenze esterne.

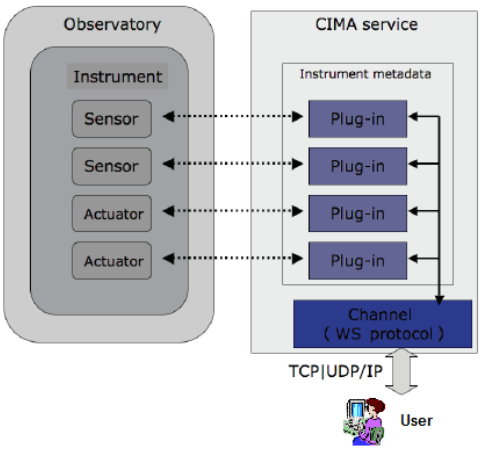


Figura 2.2: Corrispondenza tra CIMA Service & Instruments

Alcuni degli approcci utilizzati per raggiungere gli obiettivi citati sono:

1. Il Layered Specification che assicura l'interoperabilità e il riutilizzo delle interfacce per gli strumenti tramite una programmazione a strati;

2. Sviluppo di plug-in per l'implementazione delle funzioni specifiche di ogni strumento;

3. Un approccio gerarchico di questi ultimi senza limiti alla profondità di annidamento e alla locazione del singolo componente.

## 2.3 Instrument Element

L'Instrument Element [21] è un componente di Grid/Cloud Computing che fornisce un'infrastruttura computazionale e di dati, attraverso un' astrazione

di strumenti reali e un' interfaccia interattiva per controllarli.

L'Instrument Element offre la possibilità di esporre le funzionalità di uno strumento generico utilizzando un approccio orientato ai Web Service. Le applicazioni distribuite, come Grid e infrastrutture di tipo service-oriented, sono una raccolta di software eterogenei che devono essere installati e configurati che richiedono inoltre una manutenzione continua da parte di personale specializzato. In una situazione ideale le macchine che devono cooperare insieme dovrebbero recuperare informazioni riguardo l'ambiente di lavoro e adeguare le loro funzionalità in base alle loro esigenze, senza limite all'interazione umana. Le reti P2P (*Peer-to-Peer*) sono state proposte come una possibile soluzione alle problematiche incontrate. Seguendo questo approccio possiamo configurare gli strumenti in modo tale da condividere informazioni e cooperare al fine di ottimizzare le performance dell'intero sistema.

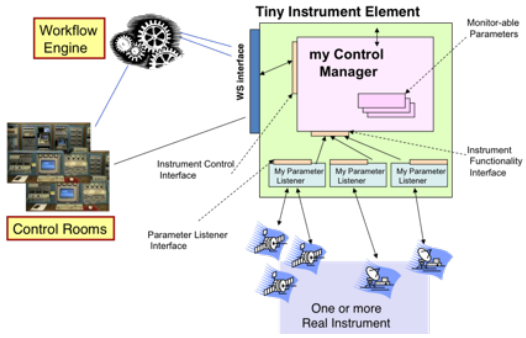


Figura 2.3: Instrument Element design

L'Instrument Element in una rete dinamica di strumenti può agire come un Super-Peer dove semplici dispositivi possono apparire e scomparire, di conseguenza è necessario una sorta di adattamento automatico all'infrastruttura esistente. Questa particolare funzionalità è tipica dei sistemi P2P in cui la rete può adattarsi automaticamente ai cambiamenti dei Peer. In un

tipico sistema P2P quando uno strumento viene aggiunto, si annuncia alla rete, dando ad altri Peer la possibilità di eseguire query o lo scambio di dati.

In questo scenario , le informazioni non sono più centralizzate ma distribuite nel sistema. Per scoprire un nuovo strumento l'Index Service della Grid esegue periodicamente la ricerca di nuovi dispositivi , al fine di rilevare lo stato di tutto il sistema, o in alternativa gli strumenti di una stessa tipologia interagiscono tra loro comunicando il proprio status.

# Capitolo 3

# Metodologie e Strumenti

Per la realizzazione delle componenti software sviluppate sono stati selezionati degli strumenti per massimizzare sia l'efficienza operazionale sia quelle della produzione e manutenzione del codice stesso e delle interfacce con gli strumenti. A tale scopo è stato scelto di adoperare il *Globus Toolkit* in versione 4 poichè middleware di grande diffusione e basato sui web service. Per lo sviluppo rapido di servizi grid web service basati sul Web Service Resource Framework si è adoperato il software *Introduce Toolkit* sviluppato da *Informatics and Information Technology*, mentre *Abstract Instrument Framework* e *Instrument Service* sono stati sviluppati in questo lavoro di laurea e rappresentano una componente fondamentale ed innovativa.

## 3.1 Il Globus Toolkit

Il Globus Toolkit [5,23] è il software middleware divenuto uno standard de facto per l'implementazione delle Grid. Esso è stato sviluppato come strumento di supporto in ambito accademico e fornisce tutti gli strumenti necessari alla

realizzazione di una Grid. Inoltre fornisce un numero considerevole di servizi ad alto livello per lo sviluppo di applicazioni di tipo Grid. I componenti fondamentali del Globus Toolkit sono:

- Information Service;
- Resource Management;
- Data Management;
- Grid Security Infrastucture.

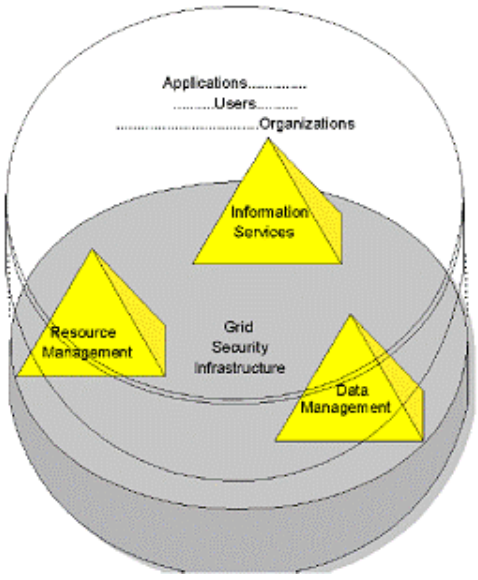


Figura 3.1: I componenti fondamentali del GT.

Nella figura 3.1 ogni piramide rappresenta un componente del GT che costituisce parte della griglia computazionale. Il componente principale del Resource Management è il GRAM (Grid Resource Allocation Manager) basato sul principio del message-passing, esso comunica con il client aggiornando lo stato della coda di esecuzione dei job; l'Information Service è costituito

dal MDS (Monitoring and Discovery Service) che fornisce l'accesso allo stato delle risorse, siano esse statiche o dinamiche. L'MDS a sua volta può essere suddiviso in due unità il GRIS (Grid Resource Information Service) che mantiene le informazioni sulle risorse locali e il GIIS (Grid Index Information Service) che contiene gli indici delle risorse registrate. I vari elementi della griglia comunicano tra loro mediante un sistema di message-passing: il GRIS comunica con l'information provider che interroga direttamente la risorsa, ottenendo le informazioni richieste e alle quali il GIIS associa un indice per la loro identificazione sulla Grid; per il Data Management il GT si avvale del GridFTP che si occupa del trasferimento e la condivisione dei file fra i vari nodi della griglia in totale sicurezza, mentre il framework OGSA-DAI (Data Access Integration) viene impiegato per rendere i database disponibili in differenti formati.

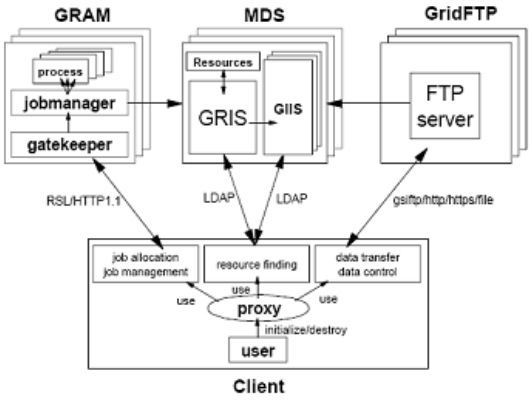


Figura 3.2: I componenti fondamentali del GT.

Tra le varie funzionalità offerte dal GT abbiamo la possibilità di utilizzare il *Web Services Resource Framework* (WSRF) mediante il quale è possibile sviluppare Web Services secondo lo standard W3C.

In una Grid il problema della sicurezza rappresenta uno dei maggiori punti di vulnerabilità di un sistema distribuito ed eterogeneo. Per risolvere l'insieme di problematiche legate alla sicurezza nel progetto Globus è stato necessario sviluppare un ulteriore strato software: il Grid Security Infrastructure.

### 3.1.1 Grid Security

La sicurezza del proprio lavoro all'interno di una Grid ha un ruolo fondamentale nel progresso di tali tecnologie e gli sviluppatori del Globus Toolkit sono costantemente impegnati su tale fronte.

Le componenti base, fornite dal Globus Toolkit, prevedono un meccanismo per l'autenticazione, l' autorizzazione e la riservatezza delle comunicazioni tra risorse Grid. Le Grid Resources prima di compiere qualsiasi operazione o richiesta devono essere autenticate. Per autenticazione si intende il processo di verifica dell' identità della Grid Resource che esegue una richiesta. In seguito possono essere concessi alcune autorizzazioni per l' accesso o utilizzo di una risorsa. Questa operazione garantisce che l'utente o il servizio goda dei privilegi necessari affinché la richiesta possa essere soddisfatta. Tuttavia, questo non esclude che le informazioni scambiate possano essere catturate, contraffatte o alterate. Il servizio che esclude questa ipotesi è la crittografia.

Il *Grid Security Infrastructure* (GSI) [1,6] fornisce il supporto necessario nell' implementazione di queste funzionalità. GSI utilizza la public key cryptography conosciuta anche come asymmetric cryptography. Questo sistema crittografico a differenza dei precedenti non si basa su una singola chiave (password) ma su due chiavi. Queste ultime consistono in numeri matema-

ticamente correlati in modo che se una chiave viene utilizzata per cifrare un messaggio, l'altra deve essere utilizzata per decifrarlo. Le nostre attuali conoscenze matematiche, e la potenza di calcolo disponibile rendono impossibile sia ottenere la seconda chiave a partire dalla prima che ottenere entrambe le chiavi a partire dal messaggio.

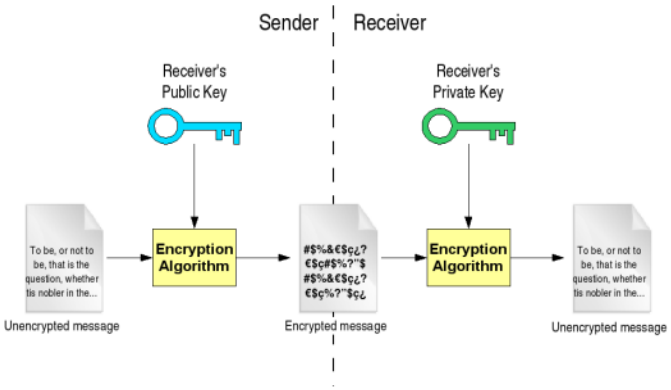


Figura 3.3: Public Key Criptography

Un concetto fondamentale nel GSI Authentication è il certificato, il quale identifica ogni utente e servizio della Grid. Un certificato GSI contiene quattro informazioni principali :

- Subject Name, che identifica la persona o il servizio rappresentato dal certificato.
- La chiave pubblica appartenente al soggetto.
- L'identità della Certificate Authority (CA) la quale garantisce l'appartenenza della chiave pubblica al Subject Name.
- La firma digitale della CA.

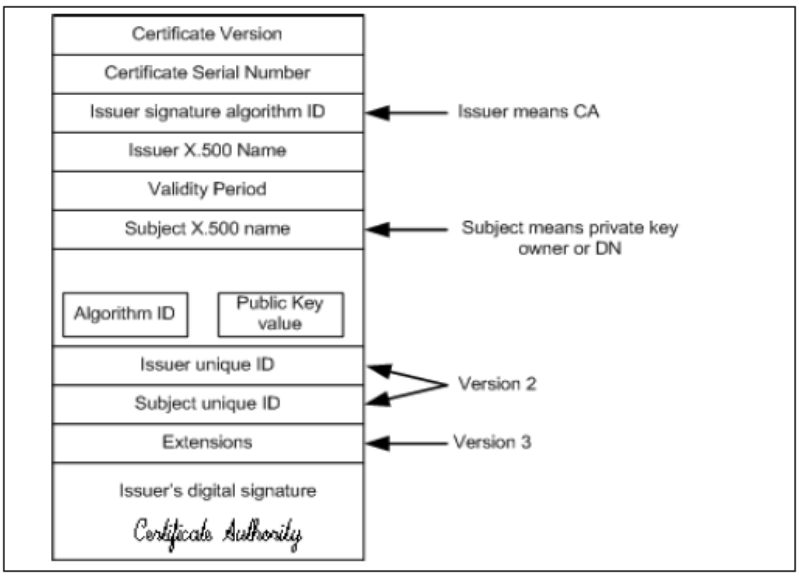


Figura 3.4: Certificato digitale

La figura 3.4 mostra una rappresentazione grafica del certificato digitale. Grazie allo sviluppo delle componenti descritte le griglie computazionali sono considerate ambienti estremamente sicuri. E' possibile quindi condividere in totale sicurezza risorse hardware e software geograficamente distribuite per realizzare progetti ed esperimenti altrimenti irrealizzabili.

## 3.2 Network Common Data Form

Il *Network Common Data Form* [9] (NetCDF) è sviluppato e distribuito sotto licenza GPL dalla *University Corporation for Atmospheric Research* (UCAR), a partire dal modello concettuale della NASA CDF, è ampiamente utilizzato per la gestione di dati multidimensionali organizzati in forma matriciale e rappresenta uno standard de-facto aperto e funzionale.

NetCDF è costituito da un insieme di librerie software e formati di dati,

indipendenti dalla piattaforma utilizzata, che supportano la creazione, l'accesso e la condivisione di dati scientifici array-oriented. Il NetCDF fornisce un'astrazione dei dataset come una collezione di oggetti auto-descritti gestiti mediante una semplice interfaccia; Inoltre consente un accesso diretto ai dati senza necessariamente conoscere i dettagli relativi alla sua memorizzazione. Correlati ai dati ci sono informazioni ausiliari quali ad esempio l'unita di misura relativa, questo consente a qualsiasi applicazione o utility di accedere ai datasets di tipo NetCDF al fine di elaborare, combinare o visualizzare i dati o parte di essi.

Il NetCDF fornisce una rappresentazione astratta dei dati e di conseguenza consente operazioni su dati di qualsiasi natura utilizzando un set di funzioni comuni senza preoccuparsi dei dettagli implementativi di basso livello.

Uno degli obiettivi del NetCDF è quello di consentire un accesso ai dati efficiente indipendentemente dalla dimensione del datasets avvalendosi di un accesso diretto piuttosto che sequenziale. Tale caratteristica risulta particolarmente vantaggiosa nel caso in cui i dati vengano letti in ordine diverso da quello di memorizzazione. Tuttavia, l' overhead può dipendere da fattori esterni quali:

- Tipologia dei dati utilizzati;
- Work-load della macchina utilizzata;
- La granularità di accesso ai dati.

In questo lavoro di tesi di laurea per gestire i dati prodotti dagli strumenti di acquisizione è stato scelto il formato NetCDF per le sue proprietà di por-

tabilità, efficienza e le sue caratteristiche di standardizzazione. Di seguito è riportato un esempio di dataset in formato NetCDF acquisito dalla centralina meteo Davis Vantage Pro 2 installata dal Dipartimento di Scienze Applicate presso la sede di Villa Doria D'Angri dell' Università degli Studi di Napoli "Parthenope".

```
Data:
netcdf DavisVantagePro2.20090128Z163500 {
 variables:
   int UTC;
   double LON;
     LON:unit = "deg";
   double LAT;
     LAT:unit = "deg";
   float BSP;
     BSP:unit = "knt";
   float SOG;
     SOG:unit = "knt";
   float HDG;
     HDG:unit = "N";
   float COG;
     COG:unit = "N";
   float AWA;
     AWA:unit = "deg";
   float AWS;
     AWS:unit = "knt";
   float AWD;
     AWD:unit = "N";
   float TWA;
     TWA:unit = "deg";
   float TWS;
     TWS:unit = "knot";
   float TWD;
     TWD:unit = "N";
   char MAIN;
   char FORE;
```

```
// global attributes:
:Generator = "it.uniparthenope.dsa.weatherstations.davis.VantagePro2";
:DateStamp = "20090128Z163500";
:InstrumentName = "davisVantagePro2";
:InstrumentType = "Weather station";
:InstrumentDescription = "";
:InstrumentPositionX = 14.26676275;
:InstrumentPositionY = 40.81970638;
:InstrumentPositionZ = NaN;
data:
UTC =95601
LON =14.26676275
LAT =40.81970638
BSP =3.494781
SOG =3.167826
HDG =217.8652
COG =227.7634
AWA =-19.75652
AWS =8.870433
AWD =198.3522
TWA =-31.86602
TWS =5.712814
TWD =186.2428
MAIN =M
FORE =G
}
```

## 3.3 The Web Service GUI Support Develop: Introduce

Introduce [30] è un toolkit *open source* distribuito con licenza GPL, concepito come elemento di supporto allo sviluppo di Web Service che rispettano lo standard WSRF compatibile con i servizi Grid.

L'obiettivo è ridurre le problematiche riguardanti l'implementazione e il deployement dei Web Service, nascondendo i dettagli di basso livello del Globus Toolkit. Esistono molti tool il cui obiettivo è di agevolare l'implementazione dei Web/Grid Service, tuttavia la maggior parte di essi sono specifici per il supporto a basso livello, in particolare per la gestione di Grid Service, risorse Grid e il deployement. L'Introduce Toolkit invece consente al Grid Developer l'utilizzo di tecnologie Grid senza necessariamente conoscere i dettagli di basso livello, inoltre Introduce consente anche un'interoperabilità sintattica affinché due entità Grid possano reciprocamente interagire.

Introduce è stato progettato per fornire supporto nelle tre principali fasi di sviluppo di un Web Service:

- Creazione dell'infrastruttura di base del Web Service, una volta definite le Service Configuration Properties, Introduce creerà l'implementazione base a cui lo sviluppatore aggiungerà metodi specifici e opzioni di sicurezza tramite le varie fasi di configurazione.

- Service Modification. Consente allo sviluppatore di aggiungere, rimuovere e modificare i metodi, le proprietà, le risorse, i service context ed i servizi/metodi riguardo la sicurezza.

- Deployement. Lo sviluppatore può effettuare il deployment del Web Service con l'ausilio di Introduce nel Grid Service Container.

Le funzioni necessarie per eseguire queste 3 fasi sono accessibili dall'ambiente di sviluppo grafico (GDE) di Introduce. Il supporto in modalità runtime per consentire le principali fasi di sviluppo è fornito da Introduce Engine. Tra le componenti principali troviamo :

1. Service creator. Esso è composto da una serie di template, i quali utilizzano le componenti di Java Emitter Templates (JET), per generare codice sorgente, file di configurazione e la struttura del package necessari per la generazione del Grid Service.

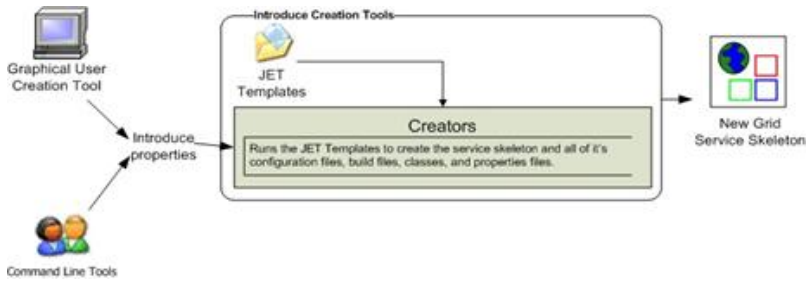


Figura 3.5: Service Creator componente di Introduce Engine

I template del codice sorgente e dei file di configurazione sono usati per creare codice JAVA personalizzato, sia per il Web Service che per il Client, e per generare i file richiesti da Globus Toolkit. Un Web Service creato tramite l' ausilio di Introduce è composto da:

- Moduli Ant per la creazione, il deploy, e le operazioni di test.
- File di configurazione personalizzati per l'integrazione con l'IDE (Eclipse).
- Interfacce standard per Client e Web Service.
- APIs client.
- Implementazione di Stub Service.
- Configurazioni per l'utilizzo dei metadati e Resource Properties.
- Configurazioni per il deployement di un secure service e la gestione delle autorizzazioni.

2. Service resynchronization. Il processo mediante il quale i tools di configurazione/generazione di Introduce toolkit analizzano l'implementazione corrente del web-service al fine di renderla coerente con la descrizione del servizio desiderato. Questo processo può aggiungere, rimuovere e modificare metodi del servizio, resource properties, impostazioni del web-service.

3. Service Deployer. Il deployement di Introduce attualmente è previsto sia per il container del Globus sia per quello di Tomcat. La struttura di riferimento al fine di rendere deployable il Web Service è il Grid Archive (GAR).

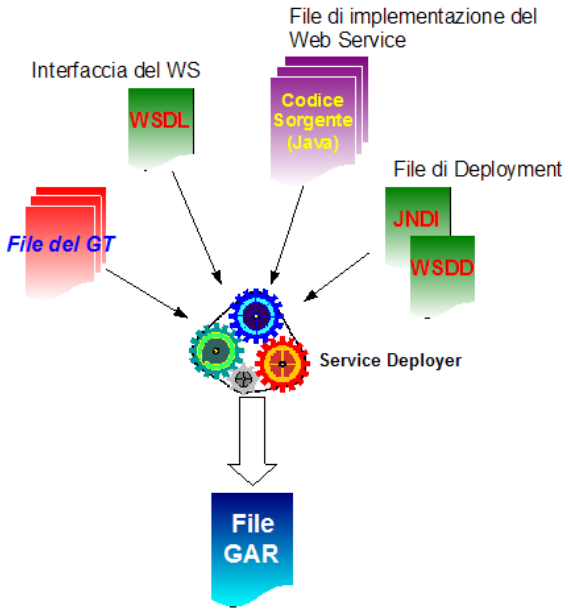


Figura 3.6: Service Deployer, componente di Introduce Engine

# Capitolo 4

# Componenti Software Sviluppate

Per rendere possibile la condivisione di strumenti su griglie computazionali è stato necessario, in prima istanza, omogeneizzare l'interfaccia di strumenti differenti analizzando le caratteristiche comuni e le azioni che un software utilizzatore può compiere su di uno strumento di tipo astratto. Realizzato un modello ad oggetti che permette l'interazione con uno strumento indipendentemente dalla specifica natura dello stesso, è stato realizzato un grid web service che ne effettua l'esposizione sulla griglia. Le componenti Grid sviluppate in questo progetto di tesi di laurea sono reperibili sul sito internet http://lmncp.uniparthenope.it/grid/IS .

## 4.1 The Abstract Instrument Framework

L' *Abstract Instrument Framework* (AIF) consente di virtualizzare gli strumenti utilizzati per l'investigazione scientifica attraverso l'implementazione

di interfacce logiche che forniscono un alto livello di interazione comune a tutti gli strumenti.

L'AIF è stato sviluppato utilizzando il linguaggio di programmazione orientato agli oggetti JAVA [24,25] e grazie ad uno stile di programmazione modulare consente con estrema facilità di espanderne le funzionalità mediante l'implementazione di *plug-in* in modo tale da consentire la gestione di qualsiasi strumento di acquisizione. Il framework AbstractInstrument si pone come componente intermedio tra gli strumenti reali e la componente software preposta all' elaborazione e/o allo storage dei dati acquisiti.

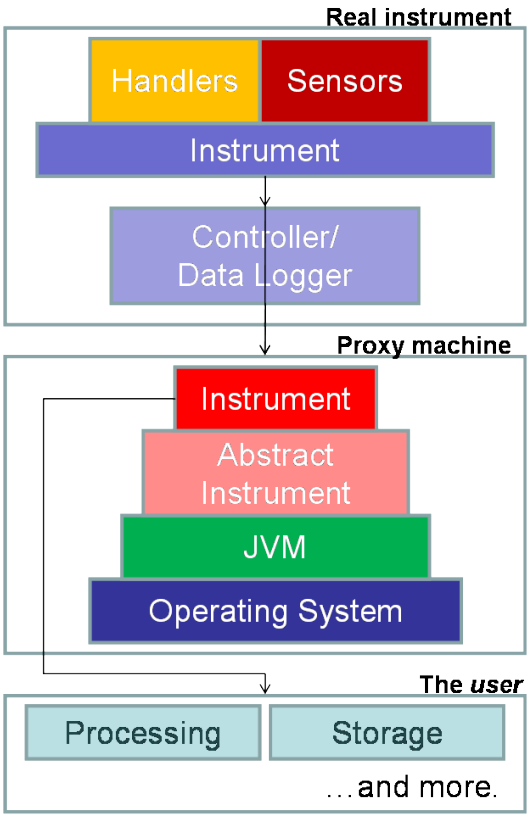


Figura 4.1: Architettura Abstract Instrument Framework

L'Abstract Instrument Framework è stato progettato mediante l'utilizzo

del *Unified Modeling Language* (UML). L'UML [26] è un linguaggio semantico che fornisce un insieme di metodologie per rappresentare in modo uniforme qualsiasi tipo di processo di sviluppo software. L'UML è caratterizzato da diverse tipologie di diagrammi utili per descrivere aspetti del progetto, tra questi hanno notevole importanza quelli dedicati alla Programmazione Orientata agli Oggetti che si avvale dei Diagrammi di Classe nei quali gli elementi forniscono una corrispondenza effettiva con gli oggetti implementati. In figura 4.2 è riportato il Diagramma di Classe per l'AIF, per semplicità sono rappresentate solo le classi più significative.

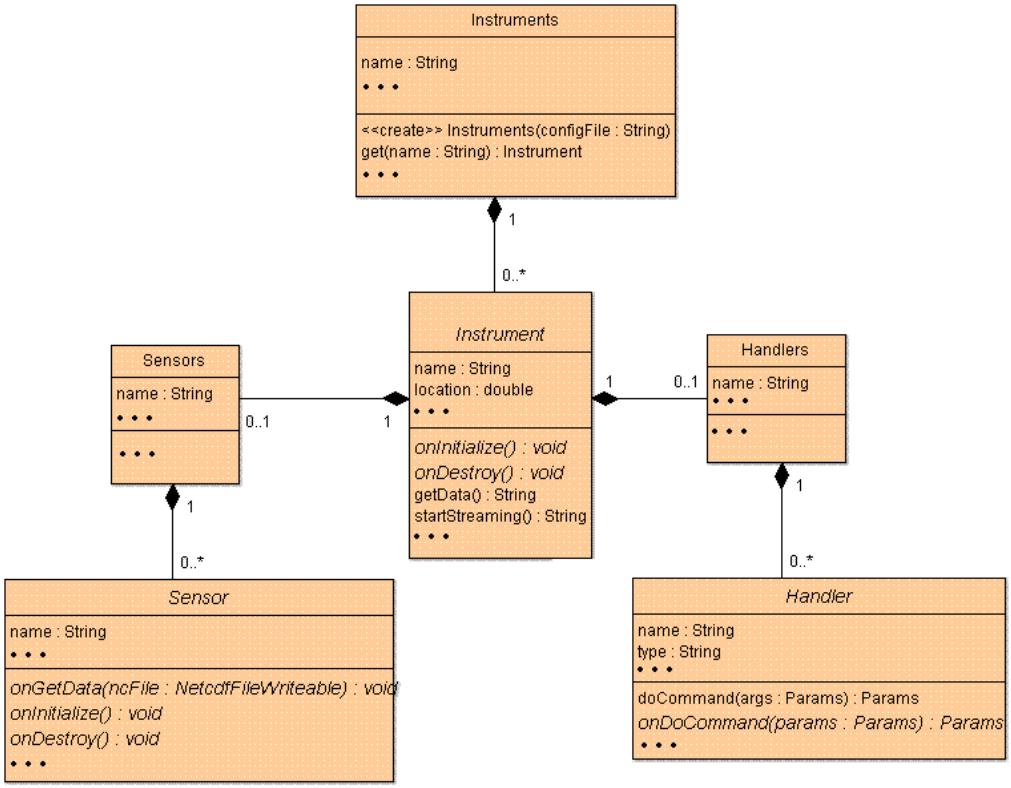


Figura 4.2: Class Diagram di Abstract Instrument

Secondo la filosofia di sviluppo ogni strumento che costituisce il Labora-

torio Virtuale idealizzato è composto da una serie di sensori, con il compito di acquisire dati, e una serie di attuatori (handler) che hanno il compito di impartire i comandi allo strumento. La classe *Instruments* raccoglie la collezione di strumenti che compone il laboratorio virtuale, tale classe estende le funzionalità della classe Vector del package java.util. In ogni locazione del vettore verrà inserito lo strumento virtualizzato con gli handler e sensor ad esso associati. Il costruttore di tale classe prevede la specifica del percorso (path) del file di configurazione dove sono presenti tutti gli strumenti del Laboratorio Virtuale e per ognuno di essi i suoi sensori ed attuatori con i relativi parametri, quali le coordinate riguardanti la posizione dello strumento, il baud-rate, lo sleep-time, e altri ancora. Il file di configurazione instruments.xml è stato implementato utilizzando il metalinguaggio *AIDL* (*Abstract Instrument Description Language*) che è stato appositamente sviluppato derivandolo dal XML (eXtensible Markup Language) [27].

La classe Instruments è una collezione di oggetti della classe Instrument. La classe Instrument è stata progettata e implementata in modo tale da nascondere i dettagli implementativi agli sviluppatori delle sotto-classi. Tra i metodi esposti dalla classe astratta Instrument hanno particolare importanza i metodi “getData” e “startStreaming”. Il metodo *getData* si occupa di interrogare i sensori associati allo strumento, ne acquisisce i dati e li restituisce alla classe chiamante. Il metodo *startStreaming* crea l’infrastruttura necessaria affinché sia possibile lo streaming dei dati e restituisce alla classe chiamante la stringa di connessione.

L’ AIF rappresenta una grande innovazione nel settore del Grid Instruments Service e grazie ad esso il Grid-developer può avvantaggiarsi di un

supporto concreto per lo sviluppo di plug-in al fine di integrare gli strumenti di acquisizione all'interno di una infrastruttura Grid mediante il processo di virtualizzazione offerta dall' AIF. Lo sviluppatore infatti potrà facilmente acquisire o inviare dati in streaming invocando i metodi dell'AIF senza curarsi minimamente dei dettagli implementativi.

Il Grid-developer potrà utilizzare le funzionalità esposte dalle classi *Sensor* e *Handler* per modellare all'interno della Grid gli strumenti che costituiscono il suo Laboratorio Virtuale. Le classi Sensors e Handlers seguono una logica di progettazione simile alla classe Instruments essi infatti sono una collezione rispettivamente di sensori e attuatori. La classe astratta Sensor fornisce il supporto necessario alle classi che si occuperanno dell'acquisizione dei dati ed espone l'insieme dei metodi necessari per l'implementazione dei sensori.

I dati generati in output dal sistema sono in formato NetCDF [9], acronimo di *Network Common Data Form*. Il formato NetCDF è sviluppato e distribuito sotto licenza GPL dalla *University Corporation for Atmospheric Research* (UCAR), a partire dal modello concettuale della NASA CDF, e rappresenta uno standard de-facto aperto e funzionale alla gestione di dati multidimensionali organizzati in forma matriciale. I file NetCDF generati sono organizzati su 5 dimensioni, x e y (le coordinate), z (altezza), il tempo (su base oraria), e p (generico parametro).

Ogni sensore legge dal file di configurazione instruments.xml i parametri relativi alla sua auto-configurazione, uno di questi parametri è lo sleeptime il quale indica per quanti millisecondi il sensore assume uno stato di attesa, dopodiché si attiva e inizia l'acquisizione. Tale comportamento è rappresentato

attraverso lo "State Diagram" in figura 4.3.

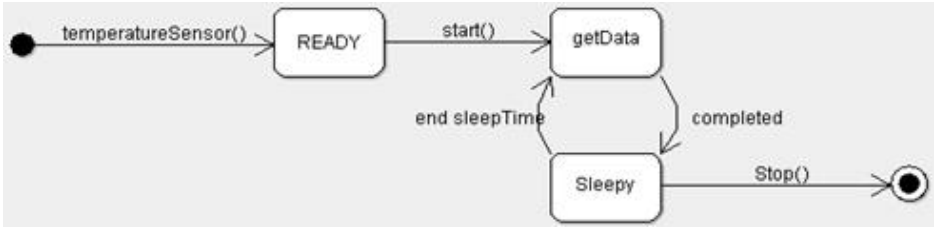


Figura 4.3: State Diagram del sensore di temperatura

Per ogni strumento, inoltre, viene data la possibilità di acquisire i dati in base alle proprie esigenze, come ad esempio acquisizioni singole o ad intervalli regolari, acquisizioni di dati compressi, acquisizione mediante streaming dei dati attraverso l'utilizzo del protocollo RTP.

La classe Handler fornisce il supporto alle classi che impartiranno i comandi allo strumento. Il metodo astratto OnDoCommand può essere utilizzato in due modi diversi e lo sviluppatore può ridefinire il metodo per:

1. Invocare una particolare funzionalità del driver dello strumento;

2. Passare a DoCommand, metodo nativo implementato nella classe Handler, la stringa necessaria per impartire l'ordine allo strumento.

Abstract Instrument è stato progettato come componente di *Instrument Service* con lo scopo di integrare in modo facile e veloce gli strumenti virtualizzati su griglia fornendo una metodologia standard di interfaccia tra i diversi tipi di strumenti hardware. Tuttavia Abstract Instrument può essere utilizzato anche in modo totalmente indipendente da Instrument Service. Come possiamo notare dal *Diagramma di Caso d'Uso* in figura 4.4 l'attore principale è rappresentato dallo sviluppatore che implementa plug-in al fine di utilizzare

lo strumento generico all'interno del Laboratorio Virtuale di Griglia, ma ciò non preclude la gestione dello strumento virtualizzato anche in locale.

Quindi, per creare un plug-in per un generico strumento basterà estendere le classi Instrument, Sensor e Handler e implementare le funzioni specifiche dello strumento, notiamo poi che grazie al supporto fornito da Abstract Instrument implementare uno strumento reale diventa un operazione intuitiva ed immediata.

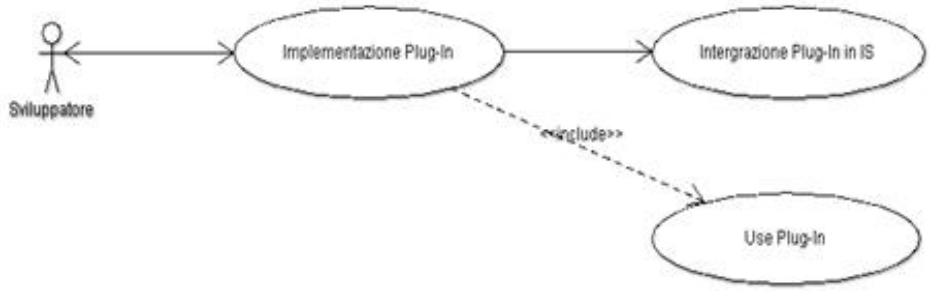


Figura 4.4: Use Cases Abstract Instrument

## 4.2 Instrument Service

In questo lavoro di tesi di laurea utilizzando la tecnologia del Grid Computing è stato realizzato un "Secure Web Service". Instrument Service (IS) è un applicazione software sviluppata nell'ambito della tecnologia Grid che permette l'accesso, il controllo e la condivisione degli strumenti virtualizzati grazie all' Abstract Instrument Framework.

Per realizzare Instrument Service sono state adottate le specifiche di sviluppo del *Web Service Resource Framework* (WSRF) [14,15], che esortano gli sviluppatori nella realizzazione di Web Service di tipo *stateful* (in grado di conservare il proprio stato) in accordo con le specifiche richieste dall'

*Open Grid Services Architecture* (OGSA) [31]. Inoltre sono state utilizzate le caratteristiche offerte dal Globus Toolkit per poter meglio integrare l'IS all'interno di una Griglia Computazionale. L'uso di questi strumenti conferiscono l'indipendenza di IS dalla piattaforma Hardware/Software utilizzata e rendono IS estremamente stabile in quanto queste tecnologie sono tutte ampiamente verificate e di larga diffusione.

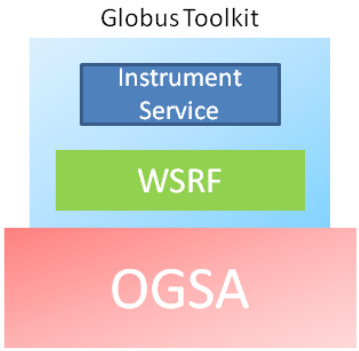


Figura 4.5: Relazione fra i vari componenti necessari per la realizzazione di Instrument Service

L'Instrument Service è stato sviluppato con lo scopo di fornire un' interfaccia ad *Abstract Instrument Framework* permettendo alle applicazioni client la gestione di strumenti scientifici attraverso i numerosi vantaggi forniti dall' utilizzo del Grid Computing come infrastruttura. Tra le caratteristiche di IS è presente la capacità di pubblicare gli strumenti di acquisizione disponibili e i dati acquisiti dai relativi sensori, automaticamente sull' *IndexService*, componente di GT4. Questa funzionalità consente al Resource Broker Service (RBS), componente sviluppata presso il Dipartimento di Scienze Applicate, di cercare gli strumenti scientifici al pari di altre risorse di griglia mediante query scritte con il linguaggio di descrizione di risorse ClassAd. Le relazioni

tra i vari componenti descritti sono riportati in figura 4.6.

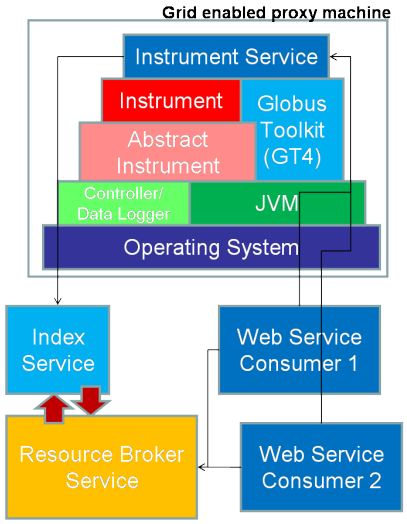


Figura 4.6: Architettura InstrumentService

Utilizzando Abstract Instrument Framework, realizzare un plug-in per integrarlo in Instrument Service con lo scopo di ampliare le funzionalità operative nella Grid è un operazione che si può definire immediata. Infatti, non bisogna far altro che realizzare un archivio JAR del framework comprensivo di plug-in, importarlo come libreria in IS e utilizzarne le funzionalità. Di seguito è riportato un frammento di codice sorgente di IS che dimostra come sia semplice e intuitivo realizzare lo streaming dei dati grazie al supporto fornito da AIF .

```
1 public java.lang.String startStreming(){
2   ResourceContext resourceContext = ResourceContext.getResourceContext();
3   InstrumentServiceContextResource thisResource = resourceContext.getResource();
4   it.uniparthenope.dsa.Instrument instrument = thisResource.getInstrument();
```

Nelle righe da 2 a 4 del metodo “startStreaming” sono implementate delle operazioni comuni a tutti i metodi che intendiamo esporre e mediante esse è

possibile il recupero della risorsa che nel nostro caso per l'IS corrisponde allo strumento.

```
5  try {
6     return instrument.startStreaming();
7   } catch (Exception e) {
8     throw new RemoteException("Troubles with start streaming: "+e.getMessage());
9 }
```

Nelle righe 5, 7, 8 è implementato un blocco try - catch il quale può essere utile nel caso in cui vengano lanciate delle eccezioni. La riga 6 rappresenta l' invocazione dello streaming e possiamo notare che un' operazione complessa come quella dell'inizio di uno streaming grazie all'utilizzo combinato del IS e del AIF è racchiusa in una sola riga di codice.

### 4.2.1 Invocazione Instrument Service

Il processo di invocazione di Instrument Service da parte di un applicazione client si svolge in tre passaggi:

1. Quando il client vuole connettersi ad IS il primo scambio di messaggi avviene fra l'applicazione ed il Discovery Service che ha il compito di informare il client su dove localizzare IS;

2. Una volta ottenuto l'indirizzo, il client lo interroga per conoscere le modalità appropriate affinché riesca a comunicare con esso, quindi IS invia al client il Web Services Description Language (WSDL), un file che segue le direttive dell'XML Schema mediante il quale IS espone la sua forma;

3. Quando il client ha le conoscenze necessarie per dialogare con esso invia la richiesta (request) che può essere un'acquisizione dati o un comando da impartire ad un handler dello strumento. La request viene elaborata da IS e il risultato dell'operazione (response) viene rispedito al client.

Tutti i passaggi comprendono l'utilizzo dei protocolli HTTP/HTTPS che al suo interno custodiscono i vari messaggi di comunicazione opportunamente organizzati mediante il Simple Object Access Protocol (SOAP) [29] che rappresenta il protocollo di riferimento per le tecnologie basate sui Web Services, sviluppato appositamente per facilitare lo scambio dei messaggi di tipo client/server.

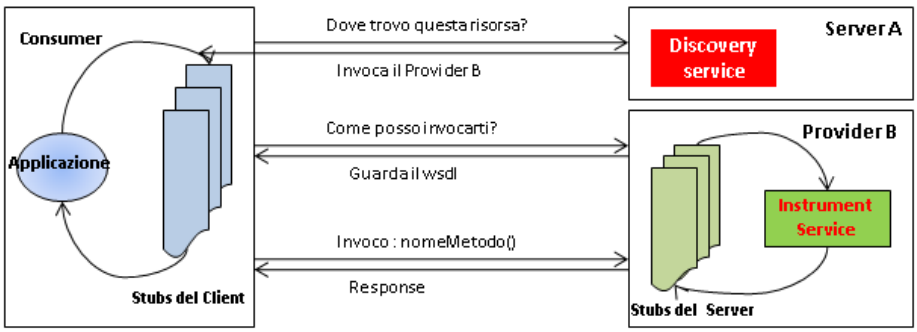


Figura 4.7: Passi necessari per invocare Instrument Service

### 4.2.2 Gestione delle risorse

I Web Service in genere sono progettati per essere stateless (senza stato) e quindi non conservano alcuna informazione sul proprio stato prima e dopo l'invocazione corrente. Instrument Service, invece, è stato implementato per fornire un servizio di tipo stateful. Questo obiettivo è ottenuto mediante la separazione del servizio dalle informazioni sullo stato. L' entità preposta a

conservare queste informazioni è detta resource (risorsa). Ad ogni risorsa viene associata una chiave univoca di identificazione chiamata *EPR (EndPoint Reference)* [23]. Ogni chiave EPR è costituita da due elementi:

*EPR = URI WS + IDchiave numerica della risorsa*

In IS ogni resource corrisponde ad uno strumento virtualizzato. Il componente di IS preposto alla gestione delle risorse è la Resource Home. L'implementazione di questo fondamentale componente è estremamente semplice grazie al supporto fornito dal Globus Toolkit.

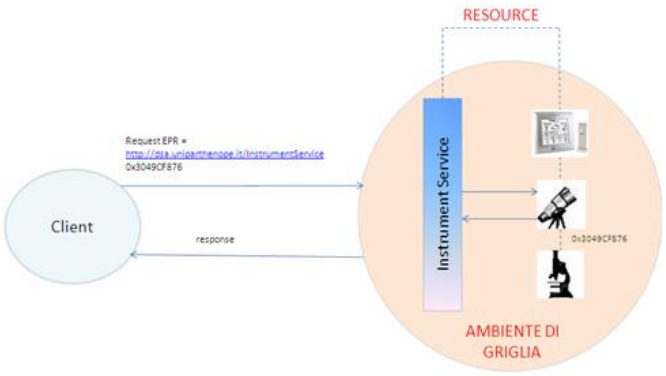


Figura 4.8: Instrument Service integrato nell'ambiente di Griglia

### 4.2.3 Instrument Service Security

Instrument Service è incentrato sull'uso condiviso e coordinato di strumenti scientifici tra diverse *Virtual Oganization*. La sua natura implica la necessità di implementare un Secure Web Service in quanto bisogna stabilire dei vincoli di autorizzazione alle diverse categorie di utenti. L'IS oltre a determinare chi è autorizzato a eseguire determinate azioni (*authorization*) consente anche

di stabilire l'identità degli utenti o dei servizi che interagiscono con esso (*authentication*) inoltre, grazie all'utilizzo del protocollo https, le comunicazioni tra il client e IS avvengono in un canale di comunicazione criptato al fine di garantirne la riservatezza del contenuto. Il componente di Globus Toolkit che fornisce un supporto nell' implementazione delle funzionalità riguardo la sicurezza è il *Grid Security Infrastructure* (GSI) [1,3,6]. GSI utilizza la public key cryptography conosciuta anche come asymmetric cryptography.

Il cuore di un secure web-service è il suo security descriptor, e in esso verranno specificati i parametri di configurazione riguardo la sicurezza. Il parametro principale da configurare è il meccanismo di autenticazione utilizzato. Instrument Service utilizza il *GridMapFile* nel quale sono riportati gli utenti autorizzati ad accedere ad IS. Inoltre è necessario stabilire, per ogni metodo, il level-protection desiderato. GSI offre due message-level schemes e un transport-level scheme. Le differenze tra i tre schemi è riassunta nella tabella 4.1.

| | **GSI Security Conversation** | **GSI Secure Message** | **GSI Transport** |
|---|---|---|---|
| **Technology** | **WS - SecureConversation** | **WS-Security** | **TLS** |
| **Privacy (Encrypted)** | **YES** | **YES** | **YES** |
| **Intregrity (Signed)** | **YES** | **YES** | **YES** |
| **Anonymus authentication** | **YES** | **NO** | **YES** |
| **Delegation** | **YES** | **NO** | **NO** |
| **Performance** | **Good if sending many message** | **Good if sending few message** | **Best** |

Tabella 4.1: Tabella comparativa dei level-protection utilizzabili

Oltre agli schemi appena descritti è possibile utilizzare come level-protection le modalità :

1. Host Authorization, grazie al quale è possibile stabilire per ogni metodo quali sono gli utenti autorizzati ad invocarlo.

2. Custom Authorization Mechanisms, sarà consentito allo sviluppatore, attraverso l'implementazione della classe InstrumentServiceAuthorization, una completa personalizzazione delle politiche di gestione dei metodi.

### 4.2.4 Componenti di Instrument Service

I file che costituiscono il nucleo centrale di Instrument Service sono molteplici. Esistono file che svolgono le funzioni di descrizione e configurazione, quali il WSDL, WSDD, JNDI, build.xml [12]. Questi file si basano sull'XML Schema, una variante dell'XML. Abbiamo poi i file che costituiscono il codice sorgente di IS. Una volta implementato IS, viene compilato ed eseguito il deployement nel Grid Service Container di Globus, in modo semplice e veloce grazie alle funzionalità offerte da Introduce toolkit.

### 4.2.5 Files di Descrizione e Deployement

Il *WSDL* (Web Services Description Languange) è sicuramente il file più importante, in esso infatti mediante le direttive del XML Schema, viene definita l'interfaccia di Instrument Service e le operazioni a cui un client può accedere. Quando il client si collega per la prima volta a IS non conosce come dialogare con quest' ultimo, allora esso viene incontro al client mostrandogli il

proprio WSDL. Il file .wsdl si presenta suddiviso in quattro sezioni mediante dei tag specifici:

```
<type>
nella prima parte si definiscono i metodi e quello che restituiscono,
 nella seconda definiamo gli attributi che stiamo utilizzando.
</type>

<message>
contiene i riferimenti agli elementi che permettono di visualizzare
lo stato del processo di comunicazione fra client e WS.
</message>

<portType>
 questa sezione è la più importante perché in essa viene definito il Web Service
  indicando le classi e le interazioni che avvengono fra di esse.
</portType>

<bilding>
in esso vengono esposti i collegamenti fra i vari componenti.
</bilding>
```

WSDD (Web Services Deployment Description)

In questo file vengono inserite le informazioni necessarie al container di Instrument Service per esporre correttamente il servizio. Nel tag *service* si specifica dove si trova IS. Se si combina questo con l' indirizzo del WS container otteniamo l' URI completo.

JNDI (Java Naming Directory Interface)

il JNDI è un API di Java, il cui compito è quello di fornire una interfaccia standard alle directory. Nel caso dei WS un servizio deve conoscere la posizione della sua RH in modo che il JNDI possa lavorare in background per risolvere queste problematiche.

build.xml

Se si utilizza ANT per compilare il servizio nel file build.xml vengono inserite tutte le informazioni necessarie affinché il WS sia compilato correttamente.

### 4.2.6 Il Codice Sorgente

Instrument Service è stato sviluppato utilizzando il linguaggio di programmazione orientato agli oggetti JAVA [22,23]. Il codice sorgente di IS è composto da numerose classi, di seguito sono riportate quelle più significative con una breve descrizione:

**InstrumentServiceImpl.java**

Costituisce l'interfaccia del servizio e rappresenta l' API del Grid Service accessibile da parte del client. E' la classe più importante perchè in esso vengono sviluppati i metodi con i quali il client interagisce con IS.

**InstrumentServiceConstants.java**

Contiene i riferimenti agli oggetti costanti di tipo QName (Qualifield Name) costituiti da due elementi: un namespace racchiuso fra le parentesi graffe rappresentato dall' URI del servizio ed un local name che identifica il nome di un attributo passato ad un metodo del servizio e quindi di tipo locale.

**InstrumentServiceResource.java**

In questa classe vengono implementate le resource. Questa è una delle tante operazioni resa immediata dall' unione di Abstract Instrument Framework e Instrument Service. Infatti basta invocare i costruttori dei plug-in realizzati grazie ad Abstract Instrument Framework.

### InstrumentServiceResourceHome.java

Si occupa della gestione delle risorse. Tale classe eredita le funzionalità di SingletonResourceHomeImpl classe del Globus Toolkit. Ciò rende l'implementazione di questo fondamentale componente semplice e veloce.

### InstrumentServiceAuthorization.java

Questa classe implementa l'interfaccia PDP [23] (Policy Decision Point) del Globus Toolkit. Per ogni metodo esposto da InstrumentService ci sarà un metodo del tipo *authorizeMethodName*, nel quale verranno implementate le politiche di gestione delle autorizzazioni del metodo.

Di seguito è riportato il diagramma di classe dell' IS.

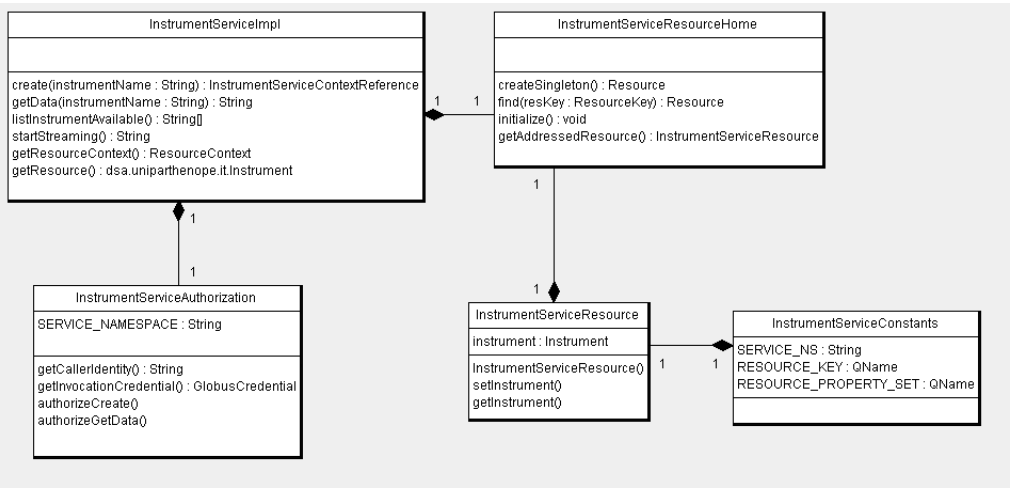


Figura 4.9: Diagramma di Classe IS

## 4.2.7 Compilazione e Deployement

L'opzione Deploy Service accessibile dall'ambiente di sviluppo grafico (GDE) di Introduce consente di compilare ed eseguire il deployement del Web Ser-

vice, nel Grid Container di Globus. Eseguite queste due operazioni si può lanciare il GT Container al fine di esporre agli utenti della Griglia Computazionale autorizzati, i metodi di Instrument Service. Il comando necessario per eseguire il GT Container è :

```
$ GLOBUS_LOCATION/bin/globus-start-container -containerDesc  -security-descriptor.xml
```

A questo punto tra i servizi elencati nel GT Container comparirà anche Instrument Service.

```
Starting SOAP server at: https://193.205.230.9:8443/wsrf/services/
With the following services:

[1]:https://193.205.230.9:8443/wsrf/services/AdminService
[2]:https://193.205.230.9:8443/wsrf/services/AuthzCalloutTestService
[3]:https://193.205.230.9:8443/wsrf/services/ContainerRegistryEntryService
[4]:https://193.205.230.9:8443/wsrf/services/ContainerRegistryService
[5]:https://193.205.230.9:8443/wsrf/services/CounterService
[6]:https://193.205.230.9:8443/wsrf/services/JWSCoreVersion
[7]:https://193.205.230.9:8443/wsrf/services/ManagementService
[8]:https://193.205.230.9:8443/wsrf/services/NotificationConsumerFactoryService
[9]:https://193.205.230.9:8443/wsrf/services/NotificationConsumerService
[10]:https://193.205.230.9:8443/wsrf/services/NotificationTestService
[11]:https://193.205.230.9:8443/wsrf/services/PersistenceTestSubscriptionManager
[12]:https://193.205.230.9:8443/wsrf/services/SampleAuthzService
[13]:https://193.205.230.9:8443/wsrf/services/SecureCounterService
[14]:https://193.205.230.9:8443/wsrf/services/SecurityTestService
[15]:https://193.205.230.9:8443/wsrf/services/ShutdownService
[16]:https://193.205.230.9:8443/wsrf/services/SubscriptionManagerService
[17]:https://193.205.230.9:8443/wsrf/services/TestAuthzService
[18]:https://193.205.230.9:8443/wsrf/services/TestRPCService
[19]:https://193.205.230.9:8443/wsrf/services/TestService
[20]:https://193.205.230.9:8443/wsrf/services/TestServiceRequest
[21]:https://193.205.230.9:8443/wsrf/services/TestServiceWrongWSDL
[22]:https://193.205.230.9:8443/wsrf/services/Version
[23]:https://193.205.230.9:8443/wsrf/services/WidgetNotificationService
[24]:https://193.205.230.9:8443/wsrf/services/WidgetService
[25]:https://193.205.230.9:8443/wsrf/services/dsa/InstrumentService
```

[26]:https://193.205.230.9:8443/wsrf/services/dsa/InstrumentServiceContext

[27]:https://193.205.230.9:8443/wsrf/services/gsi/AuthenticationService

# Capitolo 5

# Casi di Studio ed Applicazioni

In questo capitolo verrà descritta l'applicazione Test Case realizzata : *Astronomical Virtual Laboratory* (AVL). AVL è stata resa possibile grazie alla collaborazione dell' Istituto Nazionale di Astrofisica - Osservatorio Astronomico di Capodimonte (INAF-OAC) che oltre a mettere a disposizione gli strumenti scientifici utilizzati, ha fornito un valido supporto nel risolvere tutte le problematiche inerenti a questioni puramente astronomiche.

L' osservatorio virtuale è composto dai seguenti strumenti scientifici :

- Telescopio 400mm/f8 RC.
- Camera CCD Finger Lakes Instrument.
- Stazione meteo Davis Vantage Pro 2.

Questi strumenti una volta virtualizzati con Abstract Instrument Framework potranno essere condivisi dagli utenti della Griglia Computazionale grazie ad Instrument Service. A questo punto è possibile utilizzarli attraverso l'applicazione client realizzata.

## 5.1 Il Telescopio 400mm/f8 RC

Il telescopio 400mm/f8 RC in configurazione equatoriale (figura 5.1) in dotazione al OAC è uno dei telescopi più utilizzati in ambito di ricerca scientifica soprattutto per le sue qualità ottiche. Si tratta di uno strumento in configurazione ottica Ritchey Chretien, installato su una montatura equatoriale a forcella, caratterizzata dall'avere uno dei due assi, quello denominato asse orario, puntato verso la stella polare. In questo modo lo si rende parallelo all'asse di rotazione terrestre, col vantaggio di poter inseguire gli oggetti nel loro moto apparente sulla volta celeste, movimentando un solo asse, quello di Ascensione Retta. Il tubo ottico è sostenuto da una forcella, che completa il secondo grado di libertà dello strumento, consentendo il puntamento in ogni punto desiderato.

Figura 5.1: Telescopio 400mm/f8 RC in configurazione equatoriale

Al fine di virtualizzare lo strumento, è stato sviluppato un *plug-in* che grazie al contributo innovativo offerto da Abstract Instrument Framework è stata un operazione semplice e veloce. Tale operazione è possibile schematizzarla nei seguenti passi.

1. Creare le classi Telescope, DataSensor, e PointTelescopeHandler le qua-

li ereditano rispettivamente le funzionalità delle classi Instrument, Sensor e Handler.

2. Implementare i metodi astratti delle classi genitrici al fine di sviluppare le funzioni specifiche di ogni componente.

3. Integrazione del Plug-In in Instrument Service.

Il primo punto è di importanza rilevante perché definisce le linee guida che lo sviluppatore dovrà seguire al fine di realizzare la virtualizzazione dello strumento. Il secondo punto consiste nell' overriding dei metodi ereditati, i più importanti per la classe Telescope sono :

- *onInitialize()*. Tale metodo, come suggerisce il nome, si occupa dell'inizializzazione dello strumento. Esso ha due scopi principali. Il primo è quello di stabilire una connessione diretta con lo strumento; Il secondo è quello di leggere dal file di configurazione Instruments.xml scritto in metalinguaggio Abstract Instrument Description Language i parametri relativi alla sua configurazione.

```
ON-INITIALIZE (void)
1  scope <- ESTABLISH-CONNECTION()
2  scope.latitude <- GET-PARAMS("latitude")
3  scope.longitude <- GET-PARAMS ("longitude")
4  scope.elevation <- GET-PARAMS ("elevation")
5  scope.mountType <- GET-PARAMS ("mount type")
6  scope.sleepTime <- GET-PARAMS ("sleep time")
```

- *OnDestroy*. Effettua il release delle risorse istanziate nel metodo OnInitialize.

```
ON-DESTROY()
1  scope.RELEASE()
```

- OnTimer. Questo metodo viene invocato ad intervalli di tempo regolari in base allo sleepTime, parametro definito nel file di configurazione. Esso invoca il metodo nativo di AIF, GetData, il quale si occupa di interrogare i sensori e restituire il risultato dell' acquisizione.

```
ON-TIMER()
1  data <- instrument.GET-DATA()
```

Il plug-in, è inoltre composto da :

- La classe DataSensor, acquisisce i dati del telescopio relativi alla sua posizione corrente e al suo obiettivo da puntare. Il metodo più importante da ridefinire è OnGetData. L'obiettivo di quest' ultimo è acquisire i dati dal telescopio, e scriverli su file netCDF, in modo che possano essere inviati al client ed essere elaborati.

```
ON-GETDATA(NetcdfFile)
1   double rightAscension <- scope.GET("right ascension")
2   double declination <- scope.GET ("declination")
3   double targetRightAscension <- scope.GET ("targetRightAscension")
4   double targetDeclination <- scope.GET ("targetDeclination")
5   NetcdfFile.WRITE("right ascension", rightAscension)
6   NetcdfFile.WRITE ("declination",declination)
7   NetcdfFile.WRITE ("targetRightAscension", targetRightAscension)
8   NetcdfFile.WRITE ("targetDeclination", targetDeclination)
```

- La classe PointTelescopeHandler, eredita dalla classe astratta Handler ed ha il compito di impartire i comandi allo strumento. Alcuni dei comandi sono :
  - SlewToTarget / SlewToAltAz, ordinano allo strumento di puntare all' oggetto celeste che si trova nelle coordinate, rispettivamente ascensionali o azimutali, passate in input.

- AbortSlew, ordina allo strumento di arrestare l'operazione di puntamento.
- Park, muove il telescopio in park position.

OnDoCommand è il metodo preposto a compiere queste operazioni.

```
ON-DO-COMMAND(Params)
1   If ( params.GET("methodName") = "slew to target")
2     ra <- params.GET ("rightAscension")
3     dec <- params.GET("declination")
4     scope.SLEW-TO-TARGET(ra,dec)
5   else If (params.GET ("methodName") = "slew to alt az")
6     az <- params.GET ("azimuth")
7     alt <- params.GET ("altitude")
8     scope.SLEW-TO-TARGET (az,alt)
9   else If (params.GET ("methodName") = "abort slew")
10    scope.ABORT-SLEW ( )
```

E' stato dimostrato come realizzare un plug-in grazie ad Abstract Instrument Framework. Il contributo innovativo sta nel fatto che lo sviluppatore dovrà soltanto eseguire queste semplici operazioni di ridefinizione per virtualizzare lo strumento, ma soprattutto può utilizzare i metodi nativi che AIF mette a disposizione. I principali sono :

- GetData . Per eseguire un'acquisizione dati lo sviluppatore invocherà questo metodo senza alcuna ridefinizione, come visto ad esempio nel metodo OnTimer della classe Telescope. Esso si occuperà di interrogare tutti i sensori e impacchettare i dati, pronti per essere inviati al client in caso di richiesta.
- StartStreaming. Per avviare questa operazione e quindi costruire tutta l'infrastruttura necessaria, lo sviluppatore invocherà questo metodo, il

quale restituirà la stringa di connessione a cui il client dovrà collegarsi per acquisire i dati. Tutta l'operazione risulta trasparente dal punto di vista dello sviluppatore. Ovviamente sono nativi anche i metodi PauseStreaming, PlayStreaming, TearDownStreming.

In figura 5.2 è rappresentato il diagramma di classe del plug-in sviluppato.

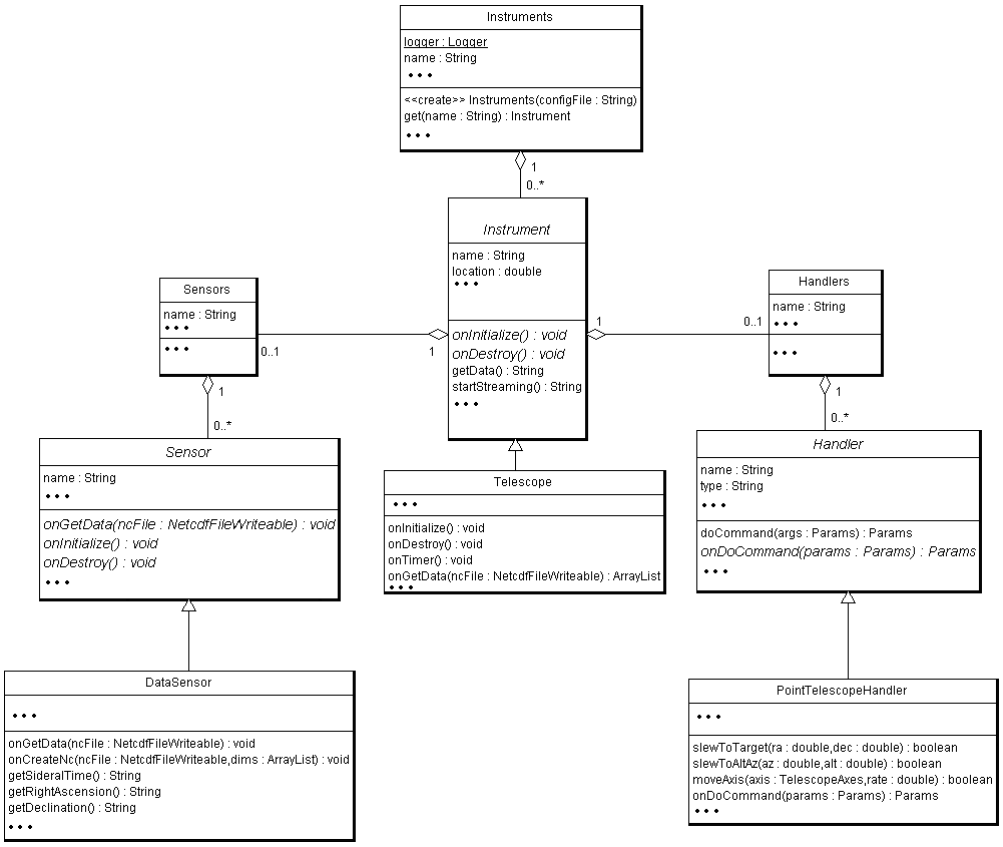


Figura 5.2: Diagramma di Classe Telescope

## 5.2 Camera CCD Finger Lakes Instrument

*Finger Lakes Instrument* (figura 5.3) è la camera in dotazione con il telescopio 400mm/f8 RC. Viene utilizzata collegandola meccanicamente sull'asse ottico del telescopio. Durante il periodo di esposizione, il sensore Charge Couple Device (CCD) registra nelle microscopiche cellette di silicio di cui costituito, il campo stellare ripreso. Terminata l'esposizione l'immagine è pronta per essere acquisita.

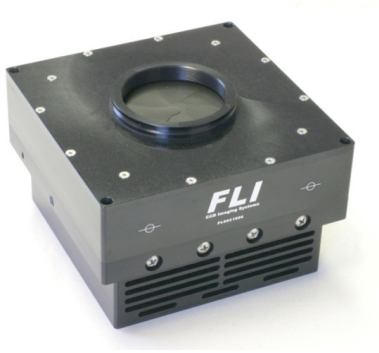


Figura 5.3: Finger Lakes Instrument

Lo sviluppo del plug-in della camera è molto simile a quello del telescopio. Esso è composto da una classe CameraFLI che estende Instrument e dalla classe CCDSensor che estende Sensor. Anche in questo caso bisogna effettuare l' overriding dei metodi visti in precedenza, ed è possibile usufruire dei metodi nativi offerti da Abstract Instrument Framework. Tra i vari metodi da ridefinire è di particolare importanza OnGetData, della classe CCDSensor, il quale memorizza in formato NetCDF, informazioni quali l'ora, la data di acquisizione, le coordinate ascensionali e l'immagine in formato raw.

## 5.3 La Stazione Meteo Davis Vantage Pro 2

La stazione meteo *Davis Vantage Pro2* (figura 5.4) consente di monitorare le condizioni atmosferiche del luogo dove è situato il telescopio. Tale strumento è di fondamentale importanza nella composizione di un Osservatorio Astronomico in quanto la qualità delle immagini acquisite dipende dalle condizioni atmosferiche. Essa è composta da:

- Barometro, per l'acquisizione dei dati di pressione atmosferica.
- Igrometro, raccoglie i dati sulla quantità di vapore acqueo presente nell'atmosfera al fine di valutarne l'umidità.
- Pluviometro, Misura la quantità di pioggia caduta.
- Anemometro, misura la velocità e direzione del vento.
- Test handler, verifica il corretto funzionamento della stazione meteo e riceve informazioni su di essa.

Figura 5.4: Davis Vantage Pro 2

Lo sviluppo del plug-in è molto simile a quelli visti in precedenza. Uno dei contributi innovativi di Abstract Instrument Framework risiede proprio in questo. Infatti la procedura di virtualizzazione suggerita, come è stato

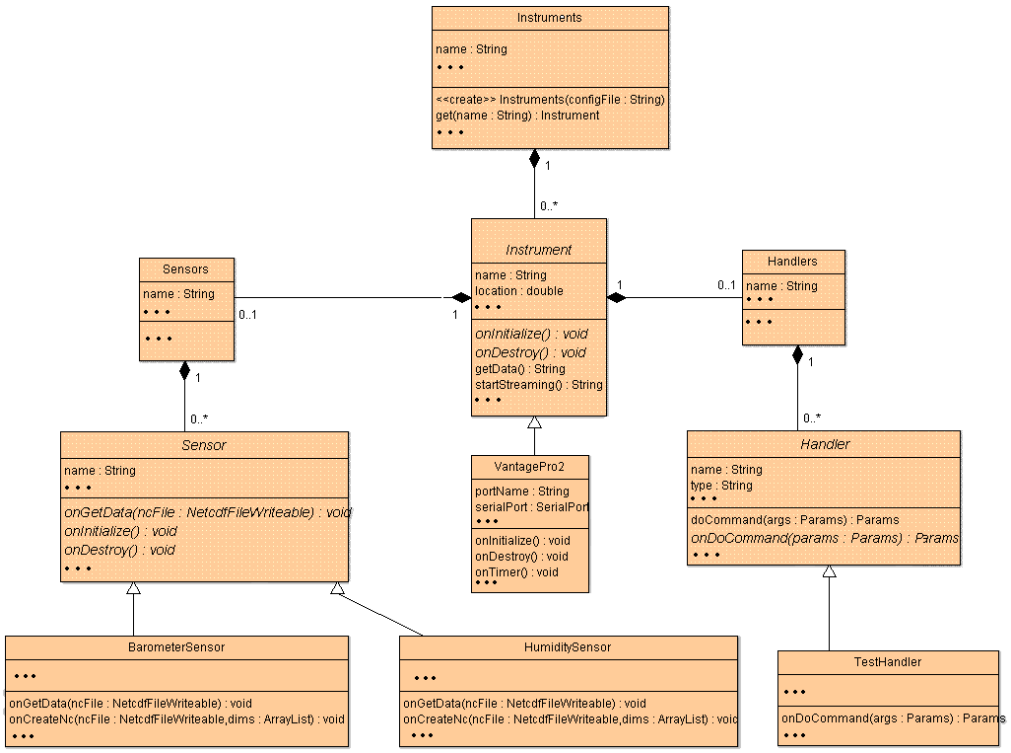


Figura 5.5: Diagramma di classe Davis Vantage Pro 2

dimostrato in questo test-case, si adatta perfettamente a qualsiasi strumento scientifico, sarà quindi possibile realizzare plug-in di strumenti profondamente diversi in modo simile. In figura 5.5 è riportato il diagramma di classe del plug-in Davis Vantage Pro 2, per semplicità non sono riportate tutte le classi.

## 5.4 Integrazione Plug-in in Instrument Service

Integrare un plug-in realizzato grazie ad Abstract Instrument Framework in Instrument Service è un operazione che è possibile definire immediata, ed è

possibile schematizzarla in queste 5 operazioni :

1. Realizzare un archivio JAR del plug-in realizzato e importarlo come libreria in IS.
2. Creare la resource associata allo strumento.
3. Creare il ServiceContext associato allo strumento.
4. Aggiungere in IS i prototipi dei metodi del plug-in che vogliamo esporre.
5. Implementare il corpo del metodo.

La prima operazione è molto semplice da realizzare, soprattutto se in fase di sviluppo, viene utilizzato un IDE. Nella seconda fase creiamo la resource associata allo strumento. Questa è una delle tante operazioni resa immediata dall'utilizzo combinato di IS e AIF. Infatti basta invocare nella classe InstrumentServiceResource i costruttori dei plug-in realizzati.

```
INSTRUMENT-SERVICE-RESOURCE()
1  instruments <- new INSTRUMENTS()
2  instrument <- new VANTAGE-PRO2-INSTRUMENT()
3  instruments.ADD(instrument)
4  instrument <- new TELESCOPE(driverId)
5  instruments.ADD(instrument)
6  instrument <- new  FLI-CAMERA(driverId)
7  instruments.ADD(instrument)
```

Nella terza fase viene creato il ServiceContext associato allo strumento, il quale definisce un interfaccia per la gestione della resource. Quest'operazione va realizzata una sola volta ed è valida per tutti gli strumenti.

```
CREATE ()
    ~ Creazione  Resource ~
1   ResourceKey <- CREATE-RESOURCE()
```

```
  ~ Creazione  Service Context
2   InstrumentServiceContextResource  thisResource  <-  FIND (resourceKey)
  ~ Recupera lo strumento dalla resourceHome
3   instrument ? resourcehome.GET(instrumentName
  ~ Associa il Service Context alla Resource
4   thisResource.SET-INSTRUMENT (instrument)
```

Nella quarta fase vengono aggiunti al Web Service i prototipi dei metodi che vogliamo esporre. Grazie alle funzionalità offerte da Introduce Toolkit quest' ultima è un operazione molto semplice come possiamo vedere in figura 5.6 . L'ultima operazione consiste nell'implementare il corpo del metodo. Di

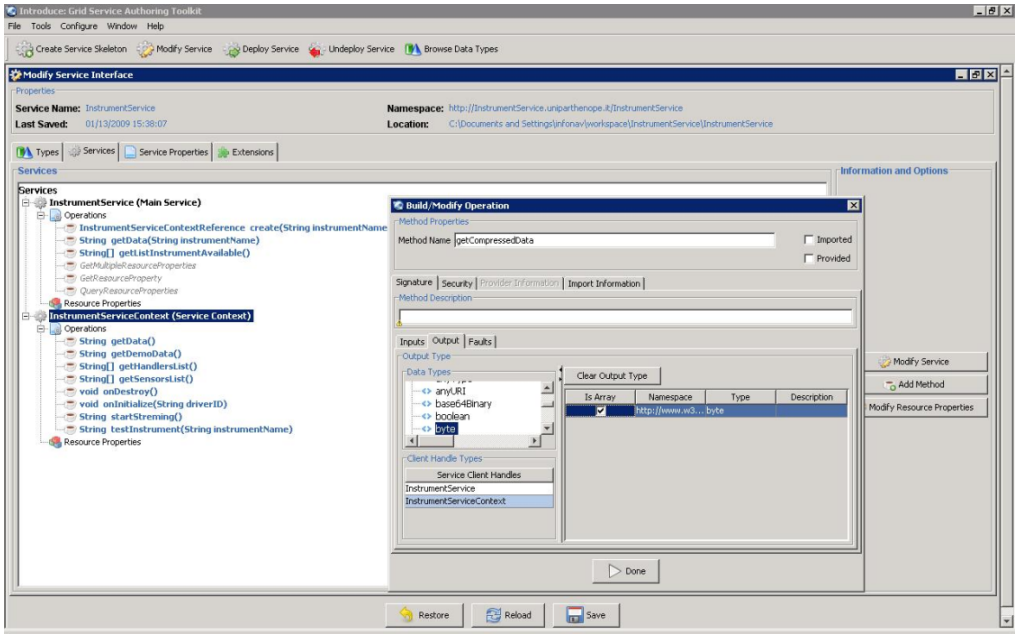


Figura 5.6: Introduce Toolkit, add method

seguito è riportato il frammento di codice relativo all'implementazione di *GetData*. Le righe 1-3 restano invariate per tutti i metodi e consistono nel recupero del riferimento allo strumento virtualizzato. Nella riga 4 avviene l'acquisizione.

```
GET-DATA ( )
1   resourceContext <- GET-RESOURCE-CONTEXT( )
2   thisResource <- resourceContext. GET-RESOURCE()
3   instrument <- thisResource.GET-INSTRUMENT( )
4   data <- instrument.GET-DATA()
5   return data
```

L'obiettivo primario è stato quello di semplificare operazioni che risultavano essere molto complesse. È stato dimostrato che l'obiettivo è stato raggiunto con successo. Tra le parole chiave di questo lavoro di tesi di laurea, c'è riusabilità del codice sorgente, infatti sarà possibile condividere qualsiasi tipo di strumento scientifico, apportando solo minime modifiche al codice.

## 5.5 Astronomical Virtual Laboratory

*Astronomical Virtual Laboratory* (AVL) è un software client progettato per essere un osservatorio astronomico virtuale [6] ed ha come obiettivo il controllo, la gestione, e l'acquisizione dati degli strumenti scientifici virtualizzati. Grazie ad AVL l'utente avrà la possibilità di svolgere tutte le operazioni che egli svolgerebbe in loco, ed inoltre può gestire una rete geograficamente distribuita di strumenti dalla sua postazione di lavoro (figura 5.7). Il client realizzato invoca i metodi che InstrumentService offre. IS quindi si pone come componente intermediario tra AVL e i plug-in relativi agli strumenti.

AVL è dotato di una interfaccia grafica intuitiva e funzionale sviluppata con l'obiettivo di soddisfare le esigenze degli astronomi che lo utilizzeranno(figura 5.8). Una volta eseguito AVL, Instrument Service verificherà se il client è in possesso dei certificati digitali necessari per comunicare con esso. Se l'operazione ha avuto successo apparirà una dialog in cui viene chiesto se

si vuole prenotare l'osservatorio per un utilizzo futuro o se si vuole utilizzarlo immediatamente, ovviamente nel caso in cui fosse disponibile o precedentemente prenotato. Successivamente si può interagire con l'osservatorio astronomico virtuale.

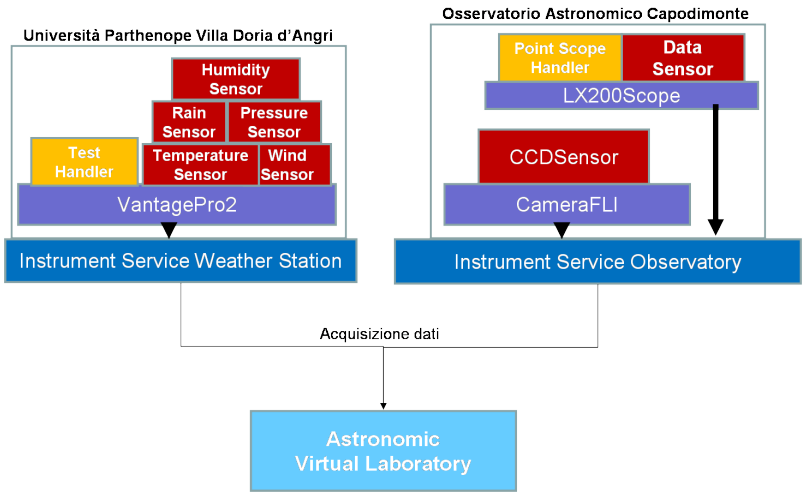


Figura 5.7: Astronomical Virtual Laboratory Design


Figura 5.8: Astronomical Virtual Laboratory GUI

Un generico caso d'uso del software realizzato potrebbe prevedere :

- Verifica delle condizioni atmosferiche. Dalla barra dei menu bisogna selezionare la voce Weather ' Connect. Apparirà un frame in cui sono presenti i dati acquisiti dalla stazione meteo (figura 5.9 b).

- Inserimento delle coordinate dell' oggetto celeste che si vuole osservare e attendere che il telescopio abbia completato la fase di puntamento (figura 5.9 a).

- Acquisire l'immagine dalla camera collegata al telescopio. Dalla barra dei menù selezionare la voce Camera' Connect. Dal frame visualizzato (figura 5.9 c) cliccare su start exposure per iniziare l'acquisizione. Al termine di quest' operazione sarà visualizzata l' immagine acquisita, pronta per essere memorizzata o elaborata da software specifici.

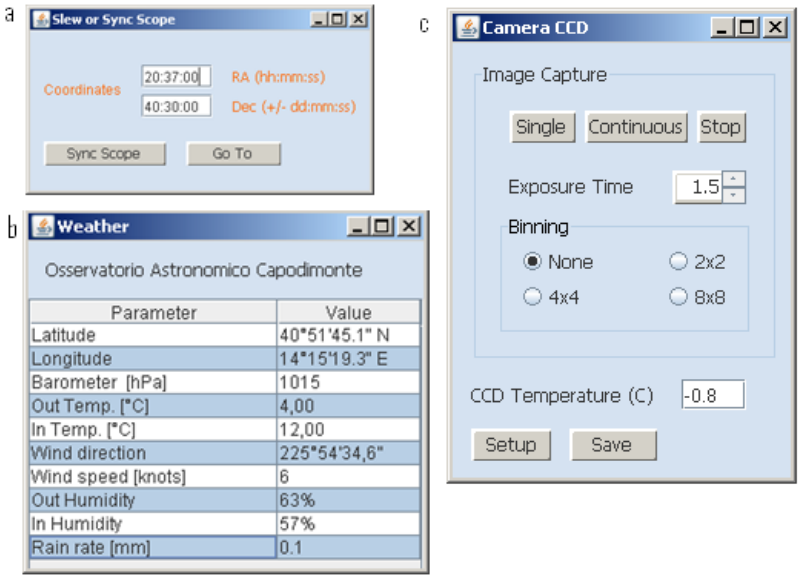


Figura 5.9: a) Frame per l'inserimento delle coordinate del target del telescopio. b) Visualizza dati acquisiti dalla centralina meteo c) Frame per l'acquisizione dati dalla Camera

In figura 5.10 è riportata un immagine acquisita utilizzando Astronomical Virtual Laboratory. Si tratta dell' Ammasso Globulare di Ercole, conosciuto anche come M13, della costellazione di Ercole.

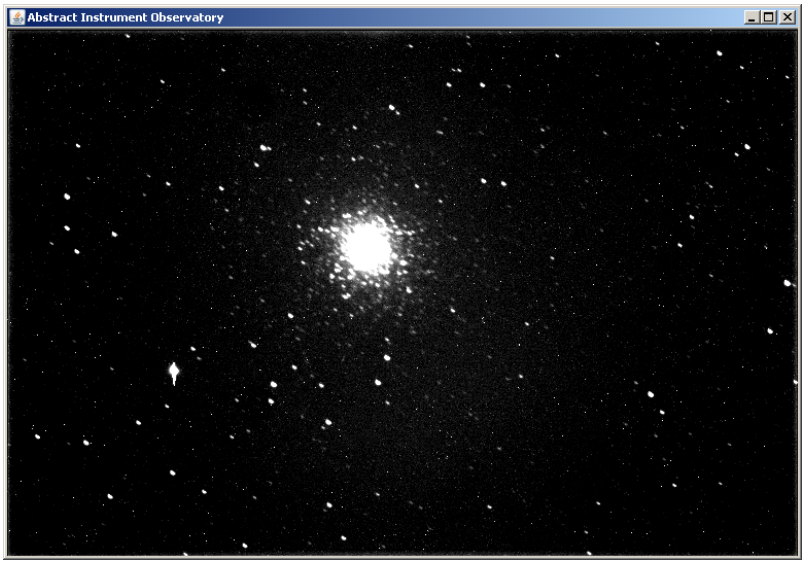


Figura 5.10: M13, costellazione di Ercole

# Capitolo 6

# Conclusione e Sviluppi Futuri

In questo lavoro di Tesi è stata proposta una soluzione alternativa e altamente innovativa per la risoluzione di problematiche comuni nel campo della ricerca scientifica integrando strumenti scientifici di acquisizione al fine di trasformarli in Grid Citizens al pari di risorse computazionali e di storage.

Le componenti Grid realizzate durante lo svolgimento del lavoro di laurea forniscono un supporto concreto al Grid Developer nella fase di virtualizzazione delle risorse hardware e nella fase di condivisione degli strumenti su Grid.

Al fine di provare il raggiungimento degli obiettivi preposti è stato sviluppato un Laboratorio Virtuale Astronomico che consente di controllare ed acquisire dati dagli strumenti scientifici virtualizzati utilizzando la Grid come infrastruttura. Questo approccio apre nuovi scenari nelle collaborazioni tra enti diversi, che potranno condividere le risorse a disposizione in un ambiente sicuro grazie alla Grid per effettuare esperimenti altrimenti irrealizzabili.

Le componenti Grid sviluppate si adattano ad ogni tipo di strumento scientifico di acquisizione e, pertanto in futuro con tali strategie possono

essere integrati altri tipi di strumenti per la realizzazione di laboratori virtuali di qualsiasi specie e ambito di sviluppo quali ambientali, chimici, fisici, medici e altro ancora. I processi di virtualizzazione e condivisione delle risorse hardware e software potranno essere ancora più immediati ed *user-friendly* attraverso lo sviluppo di un supporto grafico efficace ed efficiente che guidi l'utente nello sviluppo delle operazioni descritte e generi in modalità semi-automatica il codice sorgente necessario.

# Appendice A

# Abstract Instrument Description Language

```
<?xml version="1.0"?>
<root  xmlns:aidl="ns://instruments.dsa.uniparthenope.it">
<aidl:instruments name="myInstrumentsSet">
       <aidl:instrument name="Telescope" class="it.uniparthenope.dsa.telescope.Telescope">
         <aidl:type>Telescope</aidl:type>
         <aidl:sleepTime>1000</aidl:sleepTime>
         <aidl:position>14.22,40.85,12</aidl:position>
         <aidl:params>
            <aidl:param name="idDriver" type="string">LXP.Telescope</aidl:param>
         </aidl:params>
         <aidl:sensors>
            <aidl:sensor
                 name="DataSensor"
                 class="it.uniparthenope.dsa.telescope.DataSensor">
            </aidl:sensor>
         </aidl:sensors>
         <aidl:handlers>
            <aidl:handler
                 name="PointScopeHandler"
                     class="it.uniparthenope.dsa.telescope.PointScopeHandler">
            </aidl:handler>
```

```
   </aidl:handlers>
</aidl:instrument>
<aidl:instrument name="CameraFLI"
         class="it.uniparthenope.dsa.telescope.camerafli.CameraFLI">
   <aidl:type>CCDCamera</aidl:type>
   <aidl:sleepTime></aidl:sleepTime>
   <aidl:position>14.22,40.85,12</aidl:position>
   <aidl:params>
      <aidl:param name="idDriver" type="string">CCDFLI.Camera</aidl:param>
   </aidl:params>
   <aidl:sensors>
      <aidl:sensor
         name="CCDSensor"
              class="it.uniparthenope.dsa.telescope.camerafli.CCDSensor">
      </aidl:sensor>
   </aidl:sensors>
</aidl:instrument>
<aidl:instrument name="VantagePro2"
    class="it.uniparthenope.dsa.weatherstations.davis.VantagePro2Instrument">
<aidl:type>Weather station</aidl:type>
<aidl:sleepTime>15000</aidl:sleepTime>
<aidl:position>14.22,40.85,12</aidl:position>
<aidl:params>
   <aidl:param name="portName" type="string">COM3</aidl:param>
   <aidl:param name="baudRate" type="int">19200</aidl:param>
   <aidl:param name="bit" type="int">8</aidl:param>
   <aidl:param name="parity" type="string">none</aidl:param>
   <aidl:param name="stopBit" type="int">1</aidl:param>
</aidl:params>
<aidl:sensors>
   <aidl:sensor name="Barometer"
          class="it.uniparthenope.dsa.weatherstations.davis.BarometerSensor">
          <aidl:sleepTime>60000</aidl:sleepTime>
   </aidl:sensor>
   <aidl:sensor name="Humidity"
         class="it.uniparthenope.dsa.weatherstations.davis.HumiditySensor">
      <aidl:sleepTime>60000</aidl:sleepTime>
   </aidl:sensor>
   <aidl:sensor name="Rain"
```

```
                class="it.uniparthenope.dsa.weatherstations.davis.RainSensor">
            <aidl:sleepTime>60000</aidl:sleepTime>
         </aidl:sensor>
         <aidl:sensor name="Temperature"
                 class="it.uniparthenope.dsa.weatherstations.davis.TemperatureSensor">
            <aidl:sleepTime>60000</aidl:sleepTime>
         </aidl:sensor>
         <aidl:sensor name="Wind"
                 class="it.uniparthenope.dsa.weatherstations.davis.WindSensor">
            <aidl:sleepTime>60000</aidl:sleepTime>
         </aidl:sensor>
      </aidl:sensors>
      <aidl:handlers>
         <aidl:handler
               name="Test"
                  class="it.uniparthenope.dsa.weatherstations.davis.TestHandler">
            <aidl:params>
               <aidl:param name="timeOutThreshold" type="int">5</aidl:param>
               <aidl:param name="sleepTime" type="int">2000</aidl:param>
               <aidl:param name="commandTimeOutThreshold" type="int">50</aidl:param>
            </aidl:params>
         </aidl:handler>
      </aidl:handlers>
   </aidl:instrument>
   <aidl:instrument name="BoatA"
          class="it.uniparthenope.dsa.weatherstations.simulator.SimulatorInstrument">
      <aidl:type>Board Instrument</aidl:type>
      <aidl:sleepTime>15000</aidl:sleepTime>
      <aidl:params>
         <aidl:param name="dataFile" type="string">.//etc//20060429.csv</aidl:param>
      </aidl:params>
   </aidl:instrument>
   <aidl:instrument name="BoatB" type="Board Instrument"
         class="it.uniparthenope.dsa.weatherstations.simulator.SimulatorInstrument">
      <aidl:type>Board Instrument</aidl:type>
      <aidl:sleepTime>15000</aidl:sleepTime>
      <aidl:params>
         <aidl:param name="dataFile" type="string">.//etc//20060430.csv</aidl:param>
      </aidl:params>
```

```
   </aidl:instrument>
</aidl:instruments>
</root>
```

# Appendice B

# Classi principali di Abstract Instrument

## B.1 Instrument.java

```
package it.uniparthenope.dsa;


import java.net.InetAddress;
import java.net.ServerSocket;
import java.io.IOException;
import java.util.ArrayList;
import java.util.GregorianCalendar;
import java.util.Vector;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import ucar.nc2.Dimension;
import ucar.nc2.NetcdfFileWriteable;
import it.uniparthenope.dsa.JUnitConverter;
import com.sun.xml.internal.messaging.saaj.util.ByteOutputStream;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.XMLStreamConstants;
import org.apache.log4j.Logger;
```

```
import java.util.zip.Deflater;
import java.util.zip.DeflaterOutputStream;
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;

public abstract class Instrument implements Runnable {

static final Log logger = LogFactory.getLog(Instrument.class);
static final Logger logger=Logger.getLogger(Instrument.class.getName());

   private Vector<StreamServer> streamServers=new Vector<StreamServer>();
   private Params params=null;
   public Params getParams() { return params; }
   public void setParams(Params params) { this.params = params; }

   private JUnitConverter unitConverter=null;
   public JUnitConverter getUnitConverter() { return unitConverter; }

   private String name="";
   public String getName() { return name; }
   public void setName(String name) { this.name = name; }

   private String type="";
   public String getType() { return type; }
   public void setType(String type) { this.type = type; }

   private String description="";
   public String getDescription() { return description; }
   public void setDescription(String description) { this.description = description; }

   private int status=0;
   public int getStatus() { return status; }
   public void setStatus(int status) { this.status = status; }

   private double x,y,z;
   public double getX() { return x; }
   public double getY() { return y; }
   public double getZ() { return z; }
   public void setX(double x) { this.x = x; }
```

```
public void setY(double y) { this.y = y; }
public void setZ(double z) { this.z = z; }



private Object object;
public Object getObject() { return object; }
public void setObject(Object object) { this.object = object; }



private int sleepTime=5000;
public int getSleepTime() { return sleepTime; }
public void setSleepTime(int sleepTime) { this.sleepTime = sleepTime; }

private Thread thread=null;
public Thread getThread() { return thread; }

private Sensors sensors=new Sensors();
public Sensors getSensors() { return sensors; }
public void setSensors(Sensors sensors) { this.sensors=sensors; }

private Handlers handlers= new Handlers();
public Handlers getHandlers() { return handlers; }
public void setHandlers(Handlers handlers) { this.handlers=handlers; }

private String tmpFileName;



public abstract void onInitialize() throws Exception;
public abstract void onDestroy() throws Exception;



public void read(XMLStreamReader xmlr) throws XMLStreamException,
       IllegalAccessException,InstantiationException, ClassNotFoundException {
   boolean bDone = false;

     // Create logger
   logger = new Logger(this);
```

```
String name="unnamed";
String type="unknown";
Params params=null;
Sensors sensors=null;
Handlers handlers=null;

int sleepTime=-1;
double x = Double.NaN;
double y = Double.NaN;
double z = Double.NaN;

// Retrieve the namespace
String ns = xmlr.getAttributeNamespace(0);

// While there are more elements in the job element...
while(xmlr.hasNext() & bDone==false) {
       // Select the case of the rised Stax event:
       switch (xmlr.getEventType()) {
       // In the case of the start element...
          case XMLStreamConstants.START_ELEMENT:
                if (xmlr.getLocalName().equals("name")) {
                   xmlr.next();
                   if (xmlr.hasText()==true){
                      setName(xmlr.getText());
                   }
                }
                  // Parse the parameters element
                else if (xmlr.getLocalName().equals("params"))
                          params=new Params(xmlr);
                // Parse the sensors element
                else if (xmlr.getLocalName().equals("sensors"))
                      sensors=new Sensors(xmlr,this);
                // Parse the sensors element
                else if (xmlr.getLocalName().equals("handlers")){
                          handlers=new Handlers(xmlr,this);
                }
                  // Parse type
                  else if (xmlr.getLocalName().equals("type")) {
                          xmlr.next();
```

```
                        if (xmlr.hasText()==true)
                           type=xmlr.getText();
                }
                // Parse sleepTime
                else if (xmlr.getLocalName().equals("sleepTime")) {
                        xmlr.next();
                        if (xmlr.hasText()==true)
                           sleepTime=Integer.parseInt(xmlr.getText());
                }
                // Parse location
                else if (xmlr.getLocalName().equals("position")) {
                        xmlr.next();
                        if (xmlr.hasText()==true) {
                           String location=xmlr.getText();
                           String[] parts = location.split(",");
                           x = Double.parseDouble(parts[0]);
                           y = Double.parseDouble(parts[1]);
                           z = Double.parseDouble(parts[2]);
                        }
                }
                break;
        // In the case of the end element...
        case XMLStreamConstants.END_ELEMENT:
                // If is the end of the job element exit from this method
                if (xmlr.getLocalName().equals("instrument"))
                        bDone=true;
                break;
        }
        // Read the next element
        xmlr.next();
}
// Set the instrument name
this.name=name;
// Set the instrument type
this.type = type;
// Set the instrument sleepTime
this.sleepTime=sleepTime;
// Set the instrument parameters
this.params = params;
```

```
        // Set sensors
        this.sensors = sensors;
        // Set handlers
        this.handlers = handlers;
        // Set Location
        this.x = x;
        this.y = y;
        this.z = z;
}


 protected void finalize() throws Throwable {
     try {
         onDestroy();        // close open files
     } finally {
         super.finalize();
     }
 }

 public void start() throws Exception {
    thread = new Thread(this);
    try {
       unitConverter = new JUnitConverter();
    } catch (IOException ioException) {
       logger.error("Troubles with the unit converter: "+ioException.getMessage());
       unitConverter=null;
    }
    if (sensors != null){
       for(Sensor sensor:sensors) {
          sensor.start();
       }
    }
    if (thread!=null){
       thread.start();
       setStatus(1);
    }
 }

 public void stop() throws Exception {
```

```
   if (thread!=null) thread.stop();
   for(Sensor sensor:sensors) {
      sensor.stop();
   }
   setStatus(0);
   onDestroy();
}

public void run() {
   for(;;) {
      onTimer();
      try {
         Thread.sleep(sleepTime);
      } catch (InterruptedException interruptedException) {};
   }
}

public String getData() throws Exception {
   logger.info("getData");
   GregorianCalendar gc = new GregorianCalendar();

   String ncepDate = Utils.cal2ncep(gc);

   tmpFileName=name+"." + ncepDate +".nc";

   ucar.nc2.NetcdfFileWriteable ncFile =
                  new ucar.nc2.NetcdfFileWriteable(tmpFileName,false);

   ncFile.addGlobalAttribute("Generator",this.getClass().getName());
   ncFile.addGlobalAttribute("DateStamp",ncepDate);
   ncFile.addGlobalAttribute("InstrumentName",name);
   ncFile.addGlobalAttribute("InstrumentType",type);
   ncFile.addGlobalAttribute("InstrumentDescription",description);
   ncFile.addGlobalAttribute("InstrumentPositionX",x);
   ncFile.addGlobalAttribute("InstrumentPositionY",y);
   ncFile.addGlobalAttribute("InstrumentPositionZ",z);
```

```
ArrayList<Dimension> dims= onCreateNC(ncFile);
if (sensors!=null){
   for (Sensor sensor:sensors) {
      sensor.onCreateNC(ncFile,dims);
   }
}
try {
   ncFile.create();
} catch (IOException ioException) {
   throw new IOException("Troubles creating the netcdf file");
}
if (sensors!=null){
   for (Sensor sensor:sensors) {
      try {
         sensor.onGetData(ncFile);
      } catch (Exception e) {
         throw new IOException("Exception while reading " +
                     "data from sensor:"+sensor.getName()+ " "+e);
      }
   }
}
onGetData(ncFile);
try {
   ncFile.flush();
} catch (IOException ioException) {
   throw new IOException("Troubles closing the netcdf file");
}

ByteOutputStream bos = new ByteOutputStream();
ucar.nc2.NCdump ncDump = new NCdump();

try {
   ncDump.print(ncFile, bos, true, true, false, true, null, null);
   ncFile.close();
} catch (IOException ioException) {
   throw new IOException("Troubles while nc dumping");
}
bos.close();
String result = new String(bos.toByteArray());
```

```
   new File(tmpFileName).delete();

   return result;
}
public NetcdfFileWriteable getDataSet() throws Exception {
   logger.info("getData");
   GregorianCalendar gc = new GregorianCalendar();

   String ncepDate = Utils.cal2ncep(gc);

   tmpFileName=name+"." + ncepDate +".nc";

   ucar.nc2.NetcdfFileWriteable ncFile =
                  new ucar.nc2.NetcdfFileWriteable(tmpFileName,false);

   ncFile.addGlobalAttribute("Generator",this.getClass().getName());
   ncFile.addGlobalAttribute("DateStamp",ncepDate);
   ncFile.addGlobalAttribute("InstrumentName",name);
   ncFile.addGlobalAttribute("InstrumentType",type);
   ncFile.addGlobalAttribute("InstrumentDescription",description);
   ncFile.addGlobalAttribute("InstrumentPositionX",x);
   ncFile.addGlobalAttribute("InstrumentPositionY",y);
   ncFile.addGlobalAttribute("InstrumentPositionZ",z);

   ArrayList<Dimension> dims= onCreateNC(ncFile);
   if (sensors!=null){
      for (Sensor sensor:sensors) {
         sensor.onCreateNC(ncFile,dims);
      }
   }
   try {
      ncFile.create();
   } catch (IOException ioException) {
      throw new IOException("Troubles creating the netcdf file");
   }

   if (sensors!=null){
      for (Sensor sensor:sensors) {
         try {
```

```
            sensor.onGetData(ncFile);
         } catch (Exception e) {
            throw new IOException("Exception while reading data" +
                                  "from sensor:"+sensor.getName()+ " "+e);
         }
      }
   }

   onGetData(ncFile);

   return ncFile;
}


public String getDemoData() {
   String result =
      "Data:netcdf VantagePro2.1011059 {\n"+
      " dimensions:\n"+
      "   lat = 1;\n"+
      "   lon = 1;\n"+
      " variables:\n"+
      "   float outTemp(lat, lon);\n"+
      "     outTemp:units = \"F\";\n"+
      " data:\n"+
      "outTemp =\n"+
      "  {\n"+
      "    {64.2}\n"+
      "  }\n"+
      "}\n";
   return result;
}

public abstract void onValueChanged();
public abstract void onTimer();
public abstract void onGetData(NetcdfFileWriteable ncFile);
public abstract ArrayList<Dimension> onCreateNC(NetcdfFileWriteable ncFile);

public Sensor getSensor(String name) {
   return sensors.get(name);
```

```
}
public Handler getHandler(String name) {
   return handlers.get(name);
}
//start streaming
public String startStreaming() throws Exception {
   String result="";
   // Find localhost name
   InetAddress localHost=InetAddress.getLocalHost();

   int portBase=2048;
   int portEnd=3072;

   boolean done=false;
   int port=portBase;
   do {
      try {
         ServerSocket serverSocket=new ServerSocket(port);
         serverSocket.close();
         done=true;
      } catch (IOException ioException) {
         port++;
      }
   } while (!done && port<portEnd);
   if (port==portEnd)
      throw new Exception("No TCP ports available!");

   result="aisp://"+localHost.getHostName()+":"+port;

   System.out.println(result);

   StreamServer streamServer=new StreamServer(port,this);
   Thread thread=new Thread(streamServer);
   thread.start();
   return result;
}
public byte[] getCompressedData() throws Exception{
   String data=getData();
```

```
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DeflaterOutputStream dos = new DeflaterOutputStream(baos);
        ObjectOutputStream oos = new ObjectOutputStream(dos);
        oos.writeObject(data);

       oos.flush();
       oos.close();
       baos.close();
       byte[] result=baos.toByteArray();

       return result;
   }
   public abstract void onSetupStreaming(StreamServer streamServer);
   public abstract void onPlayStreaming(StreamServer streamServer);
   public abstract void onPauseStreaming(StreamServer streamServer);
   public abstract void onTeardownStreaming(StreamServer streamServer);
}
```

## B.2 Sensor.java

```
package it.uniparthenope.dsa;

import java.util.ArrayList;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import org.apache.log4j.Logger;
import ucar.nc2.Dimension;
import ucar.nc2.NetcdfFileWriteable;


public abstract class Sensor implements Runnable {

static final Logger logger=Logger.getLogger(Sensor.class.getName());

   private Params params=null;
   public Params getParams() { return params; }
```

```
public void setParams(Params params) { this.params = params; }

private String name;
public String getName() { return name; }
public void setName(String name) { this.name = name; }

private String type="";
public String getType() { return type; }
public void setType(String type) { this.type = type; }

private Instrument instrument=null;
public void setInstrument(Instrument instrument) { this.instrument=instrument; }
public Instrument getInstrument() { return instrument; }

private Thread thread=null;
public Thread getThread() { return thread; }

protected Object value=null;
public Object getValue() { return value; }

private int sleepTime=60000;
public int getSleepTime() { return sleepTime; }
public void setSleepTime(int sleepTime) { this.sleepTime = sleepTime; }

public Sensor(){

}

public Sensor(Instrument instrument, String name) {
   this.instrument=instrument;
   this.name = name;
   thread = new Thread(this);
   thread.setName(name);
   value=null;
}

public void read(XMLStreamReader xmlr,String name,Instrument instrument)
                                                    throws XMLStreamException {
   boolean bDone = false;
```

```
  // Create logger
  logger = new Logger(this);

String name="unnamed";
String type="unknown";
Params params=null;
int sleepTime=-1;
  // Retrieve the namespace
  String ns = xmlr.getAttributeNamespace(0);

  // While there are more elements in the job element...
  while(xmlr.hasNext() & bDone==false) {
          // Select the case of the rised Stax event:
          switch (xmlr.getEventType()) {
          // In the case of the start element...
          case XMLStreamConstants.START_ELEMENT:
                if (xmlr.getLocalName().equals("name")) {
                   xmlr.next();
                   if (xmlr.hasText()==true)
                      name=xmlr.getText();
                }
                  // Parse the properties element
                else if (xmlr.getLocalName().equals("params"))
                          params=new Params(xmlr);
                  // Parse type
                  else if (xmlr.getLocalName().equals("type")) {
                          xmlr.next();
                          if (xmlr.hasText()==true)
                             type=xmlr.getText();
                  }
                  // Parse sleepTime
                  else if (xmlr.getLocalName().equals("sleepTime")) {
                          xmlr.next();
                          if (xmlr.hasText()==true)
                             sleepTime=Integer.parseInt(xmlr.getText());
                          logger.debug("   sleepTime : "+sleepTime);
                  }
                  break;
```

```
                // In the case of the end element...
                case XMLStreamConstants.END_ELEMENT:
                        // If is the end of the job element exit from this method
                        if (xmlr.getLocalName().equals("sensor"))
                                bDone=true;

                        break;
                }
                // Read the next element
                xmlr.next();
        }

        // Set the instrument name
        this.name=name;

        // Set the instrument type
        this.type = type;

        // Set the instrument sleepTime
        this.sleepTime=sleepTime;

        // Set the instrument parameters
        this.params = params;

        this.instrument=instrument;

        thread = new Thread(this);
        thread.setName(name);
        value=null;

}

public void start() throws Exception {
   onInitialize();
   if (thread!=null){
      thread.start();}
}

public void stop() throws Exception {
```

```
        if (thread!=null) thread.stop();
        onDestroy();
    }



    public void update(Object newValue) {
        if ((value==null && newValue!=null) ||
                    (value!=null && newValue!=null && !value.equals(newValue))) {
            value=newValue;
            onValueChanged();
        }
    }

    public void run() {
        for(;;) {
            onTimer();
            try {
                Thread.sleep(sleepTime);
            } catch (InterruptedException interruptedException) {};
        }
    }

    public abstract void onCreateNC(NetcdfFileWriteable ncFile,
                            ArrayList<Dimension> dims) throws Exception;
    public abstract void onGetData(NetcdfFileWriteable ncFile) throws Exception;
    public abstract void onValueChanged();
    public abstract void onTimer();
    public abstract void onInitialize() throws Exception;
    public abstract void onDestroy() throws Exception;
}
```

## B.3 Handler.java

```
package it.uniparthenope.dsa;

import javax.xml.stream.XMLStreamConstants;
```

```
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;

import ucar.nc2.NetcdfFile;

public abstract class Handler  {

private Instrument instrument=null;
public void setInstrument(Instrument instrument) { this.instrument=instrument; }
public Instrument getInstrument() { return instrument; }

private String name;
public String getName() { return name; }
public void setName(String name) { this.name = name; }

private Params params=null;
public Params getParams() { return params; }
public void setParams(Params params) { this.params = params; }

private String type="";
public String getType() { return type; }
public void setType(String type) { this.type = type; }



public Params doCommand(Params args) throws Exception {
Params results=null;
results=onDoCommand(args);
return results;
}

public abstract Params onDoCommand(Params params) throws Exception;

public Handler(){
}

public Handler(Instrument instrument, String name) {
this.instrument=instrument;
this.name = name;
this.type="";
```

```
}


   public void read(XMLStreamReader xmlr,String name,Instrument instrument)
                                                    throws XMLStreamException {
      boolean bDone = false;

        // Create logger
        logger = new Logger(this);



      String type="unknown";
      Params params=null;

        // Retrieve the namespace
        String ns = xmlr.getAttributeNamespace(0);



        // While there are more elements in the job element...
        while(xmlr.hasNext() & bDone==false) {

                // Select the case of the rised Stax event:
                switch (xmlr.getEventType()) {

                // In the case of the start element...
                case XMLStreamConstants.START_ELEMENT:

                      if (xmlr.getLocalName().equals("name")) {
                         xmlr.next();
                         if (xmlr.hasText()==true){
                            name=xmlr.getText();
                         }
                      }
                      // Parse the properties element
                      if (xmlr.getLocalName().equals("params")){
                         params=new Params(xmlr);
                      }
                       // Parse type
                        else if (xmlr.getLocalName().equals("type")) {
```

```
                        xmlr.next();
                        if (xmlr.hasText()==true)
                           type=xmlr.getText();
                    }

                    break;
                // In the case of the end element...
                case XMLStreamConstants.END_ELEMENT:

                    // If is the end of the job element exit from this method
                    if (xmlr.getLocalName().equals("handler"))
                        bDone=true;

                    break;
                }

                // Read the next element
                xmlr.next();
            }

            // Set the handler name
            this.name=name;

            // Set the handler type
            this.type = type;


            // Set the handler parameters
            this.params = params;

            //Set the instrument
            this.instrument=instrument;
        }

    }
```

# Appendice C

# Plug-in Telescopio

## C.1 Telescope.java

```
package it.uniparthenope.dsa.telescope;

import java.util.ArrayList;
import ucar.nc2.Dimension;
import ucar.nc2.NetcdfFileWriteable;
import it.uniparthenope.dsa.Instrument;
import it.uniparthenope.dsa.Instruments;
import it.uniparthenope.dsa.Param;
import it.uniparthenope.dsa.Params;
import it.uniparthenope.dsa.StreamServer;

public class Telescope extends Instrument
{

   private com.jeamy.jascom.Telescope telescope=null;

   public com.jeamy.jascom.Telescope getTelescope() {
      return telescope;
   }

   public void setTelescope(com.jeamy.jascom.Telescope telescope) {
```

```
   this.telescope = telescope;
}

public Telescope(){

}

public void onInitialize() throws Exception
{
   String idDriver=getParams().get("idDriver").getValue();
   com.jeamy.jascom.Telescope telescope = new com.jeamy.jascom.Telescope(idDriver);
   setTelescope(telescope);
   telescope.setConnected(true);
   ((PointScopeHandler)getHandler("PointScopeHandler")).setScope(telescope);


}
public void onDestroy() throws Exception{
   getTelescope().setConnected(false);
   getTelescope().close();
}

public static void main(String[] args) throws Exception
{

}
public void onTimer() {
   String data="";
   try {
      data = getData();
   } catch (Exception e) {
      e.printStackTrace();
   }
   System.out.println("Data:\n"+data);
}

public void onValueChanged(){}
public void onGetData(NetcdfFileWriteable ncFile){}
public ArrayList<Dimension> onCreateNC(NetcdfFileWriteable ncFile){return null;}
```

```
   public void onSetupStreaming(StreamServer streamServer){}
   public void onPlayStreaming(StreamServer streamServer){}
   public void onPauseStreaming(StreamServer streamServer){}
   public void onTeardownStreaming(StreamServer streamServer){}
}
```

## C.2 DataSensor.java

```
package it.uniparthenope.dsa.telescope;

import java.io.IOException;
import java.util.ArrayList;

import com.jacob.com.ComFailException;
import com.jeamy.jascom.helper.Util;
import ucar.ma2.ArrayChar;
import ucar.ma2.DataType;
import ucar.ma2.Index;
import ucar.ma2.InvalidRangeException;
import ucar.nc2.Dimension;
import ucar.nc2.NetcdfFileWriteable;
import it.uniparthenope.dsa.Instrument;
import it.uniparthenope.dsa.Sensor;

public class DataSensor extends Sensor {
   private Telescope lX200=null;
   public DataSensor() {
      super();
   }
   public DataSensor(Instrument instrument) {
      super(instrument,"DataSensor");
      lX200=(Telescope)instrument;
   }
   public String getSideralTime() {
      try {
         Util util=new Util();
         String sd= util.hoursToHMS(lX200.getTelescope().siderealTime());
```

```
      util.close();
      return sd;
   } catch (ComFailException e) {
      e.printStackTrace();
   } catch (Exception e) {
      e.printStackTrace();
   }
   return null;
}
public String getRightAscension() {
   try {
      Util util=new Util();
      String ra= util.hoursToHMS(lX200.getTelescope().rightAscension());
      util.close();
      return ra;
   } catch (ComFailException e) {
      e.printStackTrace();
   } catch (Exception e) {
      e.printStackTrace();
   }
   return null;
}
public String getDeclination() {
   try {
      Util util=new Util();
      String dec= util.degreesToDMS(lX200.getTelescope().declination());
      util.close();
      return dec;
   } catch (ComFailException e) {
      e.printStackTrace();
   } catch (Exception e) {
      e.printStackTrace();
   }
   return null;
}
public String getAzimuth() {
   try {
      Util util=new Util();
      String az= util.degreesToDMS(lX200.getTelescope().azimuth());
```

```
      util.close();
      return az;
   } catch (ComFailException e) {
      e.printStackTrace();
   } catch (Exception e) {
      e.printStackTrace();
   }
   return null;
}
public String getAltitude(){
   try {
      Util util=new Util();
      String alt= util.degreesToDMS(lX200.getTelescope().altitude());
      util.close();
      return alt;
   } catch (ComFailException e) {
      e.printStackTrace();
   } catch (Exception e) {
      e.printStackTrace();
   }
   return null;
}
public String  getUTCDate() throws Exception{
    return lX200.getTelescope().UTCDate().toString();
}

public void onInitialize() throws Exception {}
public void onDestroy() throws Exception {}
public void onCreateNC(NetcdfFileWriteable ncFile, ArrayList<Dimension> dims) {

   Dimension dimString=ncFile.addUnlimitedDimension("String");
   ncFile.addVariable("UTC",DataType.CHAR,new Dimension[]{dimString});
   ncFile.addVariableAttribute("UTC", "units","date");
   ncFile.addVariable("SideralTime",DataType.CHAR,new Dimension[]{dimString});
   ncFile.addVariableAttribute("SideralTime", "units","hms");
   ncFile.addVariable("RightAscension",DataType.CHAR,new Dimension[]{dimString});
   ncFile.addVariableAttribute("RightAscension", "units","hms");
   ncFile.addVariable("Declination",DataType.CHAR,new Dimension[]{dimString});
   ncFile.addVariableAttribute("Declination", "units","dms");
```

```
   ncFile.addVariable("Azimuth",DataType.CHAR,new Dimension[]{dimString});
   ncFile.addVariableAttribute("Azimuth", "units","dms");
   ncFile.addVariable("Altitude",DataType.CHAR,new Dimension[]{dimString});
   ncFile.addVariableAttribute("Altitude", "units","dms");
}
public void onGetData(NetcdfFileWriteable ncFile) throws Exception{

   String utcString=getUTCDate();
   ArrayChar.D1 utc=new ArrayChar.D1(utcString.length());
   Index iUtc = utc.getIndex();
   for (int i=0;i<utcString.length();i++)
      utc.setChar(iUtc.set(i),utcString.charAt(i));

   String sidTime=getSideralTime();
   ArrayChar.D1 sid=new ArrayChar.D1(sidTime.length());
   Index iSidTime = sid.getIndex();
   for (int i=0;i<sidTime.length();i++)
      sid.setChar(iSidTime.set(i),sidTime.charAt(i));
   String raString=getRightAscension();
   ArrayChar.D1 ra=new ArrayChar.D1(raString.length());
   Index iRa = ra.getIndex();
   for (int i=0;i<raString.length();i++)
      ra.setChar(iRa.set(i),raString.charAt(i));

   String decString=getDeclination();
   ArrayChar.D1 dec=new ArrayChar.D1(decString.length());
   Index iDec = dec.getIndex();
   for (int i=0;i<decString.length();i++)
      dec.setChar(iDec.set(i),decString.charAt(i));

   String azString=getAzimuth();
   ArrayChar.D1 az=new ArrayChar.D1(azString.length());
   Index iAz = az.getIndex();
   for (int i=0;i<azString.length();i++)
      az.setChar(iAz.set(i),azString.charAt(i));

   String altString=getAltitude();
   ArrayChar.D1 alt=new ArrayChar.D1(altString.length());
   Index iAlt = alt.getIndex();
```

```
      for (int i=0;i<azString.length();i++)
         alt.setChar(iAlt.set(i),altString.charAt(i));

      try {
         System.out.println("onGetData: "+getUTCDate());
         ncFile.write("UTC", utc);
         System.out.println("onGetData: "+getSideralTime());
         ncFile.write("SideralTime",sid);
         System.out.println("onGetData: "+getRightAscension());
         ncFile.write("RightAscension",ra);
         System.out.println("onGetData: "+getDeclination());
         ncFile.write("Declination",dec);
         System.out.println("onGetData: "+getAzimuth());
         ncFile.write("Azimuth",az);
         System.out.println("onGetData: "+getAltitude());
         ncFile.write("Altitude",alt);
      } catch (IOException ioException) {
         System.out.println("IOException: "+ ioException.getMessage());
      } catch (InvalidRangeException invalidRangeException) {
         System.out.println("InvalidRangeException: "+invalidRangeException.getMessage());
      }

   }
   public void onTimer() {}
   public void onValueChanged() {}
}
```

## C.3 PointScopeHandler.java

```
package it.uniparthenope.dsa.telescope;

import it.uniparthenope.dsa.Handler;
import it.uniparthenope.dsa.Instrument;
import it.uniparthenope.dsa.Params;
import com.jeamy.jascom.helper.Util;
import com.jeamy.jascom.wrap.TelescopeAxes;
```

```
public class PointScopeHandler extends Handler implements Runnable {
   com.jeamy.jascom.Telescope scope=null;
   public com.jeamy.jascom.Telescope getScope() {
      return scope;
   }
   public void setScope(com.jeamy.jascom.Telescope scope) {
      this.scope = scope;
   }
   public PointScopeHandler(){
      super();
   }
   public PointScopeHandler(Instrument instrument){
      super(instrument,"MotionHandler");
   }
   public void slewToAltAz(double az,double alt) throws Exception{
      getScope().slewToAltAz(az, alt);
   }

   public void slewToTarget(double ra,double dec) throws Exception {

      getScope().setTargetDeclination(dec);
      getScope().setTargetRightAscension(ra);
      getScope().slewToTarget();
   }
   public void abortSlew()throws Exception {
      getScope().abortSlew();
      System.out.println("Abort sleewing");
   }

   public void moveAxis(TelescopeAxes axis,double rate) throws Exception {
      getScope().moveAxis(axis,rate);
   }
   public void park() throws Exception{
      getScope().park();
   }
   public void unPark() throws Exception{
      getScope().unpark();
   }
```

```
   public Params onDoCommand(Params params) throws Exception{
      if (params.get("command").getValue().compareTo("SlewToALtAz")==0){
         double alt=Double.valueOf(params.get("altitude").getValue());
         double az=Double.valueOf(params.get("azimuth").getValue());
         slewToAltAz(az, alt);
      }else if(params.get("command").getValue().compareTo("AbortSlew")==0){
         abortSlew();
      }else if(params.get("command").getValue().compareTo("SlewToTarget")==0){
         Util util=new Util();
         double ra=util.DMSToDegrees(params.get("ra").getValue());
         double dec=util.DMSToDegrees(params.get("dec").getValue());
         System.out.println(ra + ","+dec);
         util.close();
         slewToTarget(ra, dec);
      }else if(params.get("command").getValue().compareTo("MoveAxis")==0){
         TelescopeAxes axis;
         if (params.get("axis").getValue().compareTo("axisPrimary")==0){
            axis=TelescopeAxes.axisPrimary;
         }else
            axis=TelescopeAxes.axisPrimary;
         double rate=Double.valueOf(params.get("rate").getValue());
         moveAxis(axis, rate);
      }else if(params.get("command").getValue().compareTo("Park")==0){
         park();
      }else if(params.get("command").getValue().compareTo("UnPark")==0){
         unPark();
      }else if(params.get("command").getValue().compareTo("Tracking")==0){
         getScope().setTracking(true);
      }else if(params.get("command").getValue().compareTo("UnTracking")==0){
         getScope().setTracking(false);
      }
      return null;
   }
}
```

# Appendice D

# Classi principali di Instrument Service

## D.1 InstrumentServiceImpl.java

```
package it.uniparthenope.InstrumentService.service;

import it.uniparthenope.InstrumentService.context.stubs.types.
                              InstrumentServiceContextReference;
import it.uniparthenope.InstrumentService.context.service.globus.resource.
                                       InstrumentServiceContextResourceHome;
import it.uniparthenope.InstrumentService.service.globus.resource.
                                     InstrumentServiceResourceHome;
import it.uniparthenope.InstrumentService.context.service.globus.resource.
                                         InstrumentServiceContextResource;
import it.uniparthenope.InstrumentService.service.globus.resource.
                                        InstrumentServiceResource;
import gov.nih.nci.cagrid.introduce.servicetools.security.SecurityUtils.
                                createCreatorOnlyResourceSecurityDescriptor;
import it.uniparthenope.InstrumentService.context.stubs.types.
                                   InstrumentServiceContextReference;
import org.apache.axis.message.addressing.EndpointReferenceType;
import org.globus.wsrf.utils.AddressingUtils;
```

```
import org.apache.axis.MessageContext;
import org.globus.wsrf.ResourceKey;
import org.globus.wsrf.Constants.JNDI_SERVICES_BASE_NAME;
import javax.naming.Context;
import javax.naming.InitialContext;
import java.rmi.RemoteException;
import it.uniparthenope.dsa.Instrument;
import it.uniparthenope.dsa.Instruments;

public class InstrumentServiceImpl extends InstrumentServiceImplBase {


   public InstrumentServiceImpl() throws RemoteException {
      super();
   }

  public InstrumentServiceContextReference create(String instrumentName)
                                                  throws RemoteException {
     EndpointReferenceType epr = new EndpointReferenceType();
      InstrumentServiceContextResourceHome home = null;
      ResourceKey resourceKey = null;
      MessageContext ctx = .MessageContext.getCurrentContext();
      String servicePath = ctx.getTargetService();
      String homeName = JNDI_SERVICES_BASE_NAME + servicePath + "/" +
                                         "instrumentServiceContextHome";

      try {
         Context initialContext = new InitialContext();
         home = (InstrumentServiceContextResourceHome) initialContext.lookup(homeName);
         resourceKey = home.createResource();

         //  Grab the newly created resource
         InstrumentServiceContextResource
            thisResource = (InstrumentServiceContextResource)home.find(resourceKey);

         thisResource.setSecurityDescriptor(createCreatorOnlyResourceSecurityDescriptor());

         try {
            InstrumentServiceResourceHome resourceHome =
```

```
            (InstrumentServiceResourceHome)getResourceHome();
         InstrumentServiceResource
            singletonResource = (InstrumentServiceResource)resourceHome.find(null);

         Instrument  instrument = singletonResource.getInstruments().get(instrumentName);
         if (instrument==null)
            throw new RemoteException("Instrument not found");

         thisResource.setInstrument(instrument);

      } catch (Exception e) {
         throw new RemoteException("Troubles with resource home:"+e);
      }

      String transportURL = (String) ctx.getProperty(MessageContext.TRANS_URL);
      transportURL = transportURL.substring(0,transportURL.lastIndexOf('/') +1 );
      transportURL += "InstrumentServiceContext";
      epr = AddressingUtils.createEndpointReference(transportURL,resourceKey);
   } catch (Exception e) {
      throw new RemoteException("Error looking up InstrumentServiceContext home:" +
                                                                  + e.getMessage(), e);
   }

   //return the typed EPR
   InstrumentServiceContextReference ref = new InstrumentServiceContextReference();
   ref.setEndpointReference(epr);

   return ref;

}

public String[] getList() throws RemoteException {
      try {
      InstrumentServiceResourceHome resourceHome =
             (InstrumentServiceResourceHome)getResourceHome();
      InstrumentServiceResource
        singletonResource = (InstrumentServiceResource)resourceHome.find(null);
      Instruments  instruments = singletonResource.getInstruments();
      int nInstruments = instruments.size();
```

```
            String[] result = new String[nInstruments];
            for(int i=0;i<nInstruments;i++) {
               result[i]=instruments.get(i).getName();
               System.out.println(result[i]);
            }
            return result;
         } catch (Exception e) {
            throw new RemoteException("Troubles with resource home:"+e);
         }

   }

   public String getData(String instrumentName) throws RemoteException {
          InstrumentServiceResourceHome resourceHome = null;
    try {
       resourceHome = (InstrumentServiceResourceHome)getResourceHome();
    } catch (Exception e) {
       throw new RemoteException("Troubles while accessin resource home");
    }

    InstrumentServiceResource
    singletonResource = (InstrumentServiceResource)resourceHome.find(null);
    Instruments  instruments = singletonResource.getInstruments();
    Instrument instrument = instruments.get(instrumentName);
    if (instrument==null) {
       throw new RemoteException("Instrument "+instrumentName+" not found");
    }

    try {
       String result = instrument.getData();
       return result;
    } catch (Exception e) {
       throw new RemoteException("Troubles while getting data"+e);
    }

   }
}
```

## D.2 InstrumentServiceContextImpl.java

```
package it.uniparthenope.InstrumentService.context.service;

import it.uniparthenope.InstrumentService.context.service.globus.resource.
                                        InstrumentServiceContextResourceHome;
import it.uniparthenope.InstrumentService.service.globus.resource.
                                        InstrumentServiceResourceConfiguration;
import it.uniparthenope.InstrumentService.service.globus.resource.
                                        InstrumentServiceResourceHome;
import it.uniparthenope.InstrumentService.context.service.globus.resource.
                                        InstrumentServiceContextResource
import it.uniparthenope.dsa.Params;
import it.uniparthenope.dsa.Instrument;
import java.io.IOException;
import java.rmi.RemoteException;
import ucar.nc2.NetcdfFileWriteable;
import org.globus.wsrf.ResourceContext;
import org.globus.wsrf.ResourceContextException;

public class InstrumentServiceContextImpl extends InstrumentServiceContextImplBase {


public InstrumentServiceContextImpl() throws RemoteException {
super();
}

  public java.lang.String getData() throws RemoteException {

      ResourceContext resourceContext = ResourceContext.getResourceContext();
   InstrumentServiceContextResource thisResource =
                 (InstrumentServiceContextResource)resourceContext.getResource();
   if (thisResource==null)
   throw new RemoteException("I cant't find the specified resource");

   Instrument instrument=thisResource.getInstrument();
   if (instrument==null)
   throw new RemoteException("Instrument not found");
```

```
   try {
   String data=(String)instrument.getData();
   return data;

   } catch (Exception exception) {
throw new RemoteException("Troubles with gettting data: "+exception.getMessage());
}

  }
public NetcdfFileWriteable getDataSet() throws RemoteException {

      ResourceContext resourceContext = ResourceContext.getResourceContext();
   InstrumentServiceContextResource thisResource =
                 (InstrumentServiceContextResource)resourceContext.getResource();
   if (thisResource==null)
   throw new RemoteException("I cant't find the specified resource");

   Instrument instrument=thisResource.getInstrument();
   if (instrument==null)
   throw new RemoteException("Instrument not found");

   try {
   NetcdfFileWriteable data=(NetcdfFileWriteable )instrument.getDataSet();
   return data;

   } catch (Exception exception) {
throw new RemoteException("Troubles with gettting data: "+exception.getMessage());
}


  public java.lang.String getDemoData() throws RemoteException {
      try {
   ResourceContext resourceContext = ResourceContext.getResourceContext();
   InstrumentServiceContextResource thisResource =
      (InstrumentServiceContextResource)resourceContext.getResource();
   Instrument instrument=thisResource.getInstrument();
   String data=(String)instrument.getDemoData();
   return data;
   } catch (Exception e) {
```

```
throw new RemoteException("Troubles with context resource home: "+e.getMessage());
}
  }

  public String[] getSensorsList() throws RemoteException {

   try {
   ResourceContext resourceContext = ResourceContext.getResourceContext();
   InstrumentServiceContextResource thisResource =
                     (InstrumentServiceContextResource)resourceContext.getResource();
   it.uniparthenope.dsa.Instrument instrument=thisResource.getInstrument();
   int nSensors = instrument.getSensors().size();
   String[] result=new String[nSensors];
   for(int i=0;i<nSensors;i++)
   result[i]=instrument.getSensors().get(i).getName();
   return result;
} catch (Exception e) {
throw new RemoteException("Troubles with context resource home: "+e.getMessage());
}

  }

  public String[] getHandlersList() throws RemoteException {
       try {
   ResourceContext resourceContext = ResourceContext.getResourceContext();
   InstrumentServiceContextResource thisResource =
          (InstrumentServiceContextResource)resourceContext.getResource();
   Instrument instrument=thisResource.getInstrument();
   int nHandlers = instrument.getHandlers().size();
   String[] result=new String[nHandlers];
   for(int i=0;i<nHandlers;i++)
   result[i]=instrument.getHandlers().get(i).getName();
   return result;
} catch (Exception e) {
throw new RemoteException("Troubles with context resource home: "+
                                                    + e.getMessage());
}

  }
```

```
  public java.lang.String startStreming() throws RemoteException {
   ResourceContext resourceContext = ResourceContext.getResourceContext();
   InstrumentServiceContextResource thisResource =
               (InstrumentServiceContextResource)resourceContext.getResource();
   Instrument instrument=thisResource.getInstrument();
   try {
return instrument.startStreaming();
} catch (Exception e) {
throw new RemoteException("Troubles with start streaming: "+e.getMessage());
}
  }

  public byte[] getCompressedData() throws RemoteException {
   ResourceContext resourceContext = ResourceContext.getResourceContext();
   InstrumentServiceContextResource thisResource =
               (InstrumentServiceContextResource)resourceContext.getResource();
   Instrument instrument=thisResource.getInstrument();
   try {
return instrument.getCompressedData();
} catch (Exception e) {
throw new RemoteException("Troubles with get compressed data: "+e.getMessage());
}
  }


  public void onInitialize() throws RemoteException {
   ResourceContext resourceContext = ResourceContext.getResourceContext();
   InstrumentServiceContextResource thisResource =
            (InstrumentServiceContextResource)resourceContext.getResource();
   Instrument instrument=thisResource.getInstrument();

   try {
 instrument.onInitialize();
} catch (Exception e) {
throw new RemoteException("Troubles with initialize telescope: "+e.getMessage());
}
  }
```

```
  public void onDestroy() throws RemoteException {
   ResourceContext resourceContext = ResourceContext.getResourceContext();
   InstrumentServiceContextResource thisResource =
             (InstrumentServiceContextResource)resourceContext.getResource();
   Instrument instrument=(LX200) thisResource.getInstrument();
   try {
 instrument.onDestroy();
} catch (Exception e) {
throw new RemoteException("Troubles with destroy telescope: "+e.getMessage());
}
  }
}
```

# Bibliografia


[1] R. Montella, G. Agrillo, D. Mastrangelo, and M. Menna. *A Globus Toolkit 4 Based Instrument Service For Environmental Data Acquisition And Distribution*. Boston, Massachusetts, USA. (2008)

[2] R. Montella. *Development of a GT4-based Resource Broker Service: an application to on-demand weather and marine forecasting*. volume LNCS 4459 of LNCS. Springer, May 2007

[3] R. Montella, G. Giunta, and G. Laccetti. *A grid based service oriented environmental modeling laboratory for research and production applications*. volume LNCS. Springer, September 2008.Springer, September 2008.

[4] R. Montella, G. Giunta, and A. Riccio. *Using grid computing based components in on demand environmental data delivery*. June 2007.

[5] R. Montella, G. Giunta, and A. Riccio. *An integrated ClassAd-Latent Semantic Indexing matchmaking algorithm for Globus Toolkit based computing grids*. volume LNCS. Springer, September 2008.

[6] I. Ascione, G. Giunta, R. Montella, P. Mariani, and A. Riccio. *A Grid Computing Based Virtual Laboratory for Environmental Simulations*. volume LNCS 4128 of LNCS. Springer, August/September 2006.

[7] G. Giunta, P. Mariani, R. Montella, and A. Riccio. *ppom: A nested, scalable, parallel and fortran 90 implementation of the princeton ocean model. Envirnonmental Modelling and Software*, 22:117-122, 2007.

[8] G. Giunta, P. Mariani, R. Montella, and A. Riccio. *Modeling and computational issues for air/water quality problems. a grid computing approach.* Il Nuovo Cimento, 28C, March/April 2007.

[9] G. Giunta, G. Laccetti, R. Montella. *Five dimension environmental data resource brokering on computational grids and scientific clouds.*

[10] Borja Sotomayor, *The Cloud Ecosystem.* UChicago DSL Seminar. University of Chicago, January 2009

[11] Borja Sotomayor, R. Montero, I. Llorente, I. Foster. *Capacity Leasing in Cloud Systems using the OpenNebula Engine.* University of Chicago.

[12] B. Sotomayor and L. Childers. *Globus Toolkit 4: Programming Java Services.* Morgan Kaufman, 2005.

[13] Borja Sotomayor, Kate Keahey, Ian Foster, *Combining Batch Execution and Leasing Using Virtual Machines.* ACM/IEEE International Symposium on High Performance Distributed Computing 2008 (HPDC 2008), Boston, MA. June 2008

[14] I.Foster, C Kesselman , and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International Journal of High Performance Computing Application, 15(3). 200-222.2001

[15] I. Foster, *The Grid: A New Infrastructure for 21st Century Science.* Physics Today, 55 (2). 42-47. 2002

[16] Greg Boss, Padma Malladi, Dennis Quan, Linda Legregni, and Harold Hall, *Cloud Computing*. High Performance On Demand Solutions (HiPODS).

[17] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, and A. Warfield, *Xen and the Art of Virtualization*. University of Cambridge Computer Laboratory. 2003

[18] N. Santos, B. Koblitz, *Metadata Services on the Grid*, CERN, Geneva, Switzerland

[19] I. M. Atkinson, D. du Boulay, C. Chee, K. Chiu, K. L. Huffman, D. F. McMullen, R. Quilici, P. Turner, and M. Wyatta. *Common instrument middleware architecture*. Extensions for the australian e-research environment. 2005.

[20] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. *The globus striped gridftp framework and server.* November 2005.

[21] E. Frizziero, M. Gulmini, F. Lelli ,G. Maron, A. Oh, S. Orlando, A. Petrucci, S. Squizzato, and S. Traldi, *Instrument Element: a new Grid component that enables the control of remote instrumentation.*

[22] A.V. , *Introduction to Grid Computing with Globus*, IBM 2003 http://ibm.com/redbook

[23] The Globus Toolkit, The Globus Alliance, http://www.globus.org

[24] . B. Eckel, *Thinking in Java, 3rd Edition Revision 4.0*, Apogeo, 2002.

[25] C. S. Horstmann, G Cornell, *Core Java 2*, Pearson Prentice Hall, 2003

[26] J. Arlow, I. Neustadt, *UML 2 e Unifield Process*, second edition, McGraw-Hill, 2005

[27] G. K. Thiruvathukal, *XML and computational science*, Computing in Science & Engineering, Volume 6 numero 1, gennaio/febbraio 2004;

[28] AA. VV., *Java Web Service Tutto & Oltre*, Apogeo, 2003

[29] AA.VV. *Simple Object Access Protocol (SOAP)* 1.1. W3C, Note 8, 2000;

[30] *Introduce Toolkit Documentation*, http://www.cagrid.org/wiki/Introduce:Documentation.

[31] *OGSA-DAI*, University of Edinburgh, http://www.ogsadai.org.uk/.