Università degli studi di Napoli

Federico II

Facoltà di Scienze MM. FF. NN.

Corso di Laurea in Informatica

A.A. 2009-2010

Tesi di Laurea Triennale Sperimentale

DAME-KappA: integrazione del framework KNIME nella Suite DAME

Tutor

Dott.ssa Anna Corazza

Dott. Massimo Brescia

Candidato

Esposito Pamela

566/372

SOMMARIO

INDICE DELLE FIGURE

CAPITOLO 1

INTRODUZIONE

Il lavoro svolto durante l'attività di tirocinio presso il Osservatorio Astronomico di Capodimonte (INAF-OACN) e documentato in questo elaborato di tesi, consiste nell'integrazione di due piattaforme software di calcolo, rispettivamente KNIME per l'analisi dati e DAME per il data mining su Massive Data Sets (MDS) in Astrofisica, in modo da renderle interoperabili e fruibili in modo omogeneo da parte dell'utente.

In particolare, l'obiettivo finale consiste nell'arricchire i modelli di data mining messi a disposizione nella Suite DAME con la gran quantità di funzionalità offerte da KNIME, e d'altro canto fornendo a quest'ultima la possibilità di offrire servizi di analisi dati alla comunità astronomica.

L'elaborato di tesi descrive le caratteristiche della Suite (Capitolo 2) che rappresenta il contesto in cui verrà integrata la piattaforma KNIME. Descrive, inoltre, KNIME in tutte le sue caratteristiche (Capitolo 3) per fornire una visione d'insieme della sua architettura e degli strumenti, per poter garantire un'integrazione ottimale e per valutare le funzionalità messe a disposizione. Infine, fornisce una descrizione dettagliata della soluzione implementativa in termini di progettazione e sviluppo (Capitolo 4) con relativo testing (Capitolo 5).

Il mio lavoro, in qualità di membro del gruppo del progetto DAME è stato svolto in partnership con:

- Università degli studi Federico II, Dipartimento di Fisica, sezione di Astrofisica;
- Istituto Nazionale di Astrofisica (INAF), sede regionale Osservatorio Astronomico di Capodimonte[1];
- California Institute of Technology, Pasadena – USA[2];

e in collaborazione con:

- Virtual Observatory Technological Infrastructures (VOTECH);
- S.Co.P.E[3];
- Ministero dell'Istruzione dell'Università e della Ricerca (MIUR);
- European Virtual Observatory (EURO-VO).

1 www.oacn.inaf.it
2 www.caltech.edu
3 www.scope.unina.it

CAPITOLO 2

IL PROGETTO DAME

La tecnologia ha ormai coinvolto tutti i settori della nostra società. Ciò è particolarmente vero in astronomia e astrofisica, dove l'ottimizzazione di telescopi di Survey (Fig. 1), usati per l'esplorazione e la mappatura delle regioni del cielo, aumenta enormemente il potenziale di scoperte scientifiche. Si tratta di telescopi a grande campo caratterizzati da un rivelatore panoramico localizzato nel piano focale, che permette l'osservazione e la mappatura di ampie regioni di cielo. Le regioni sono fotografate ripetutamente applicando filtri con caratteristiche spettrali diverse, producendo enormi archivi di immagini e cataloghi di oggetti dell'ordine di alcuni TB/notte.

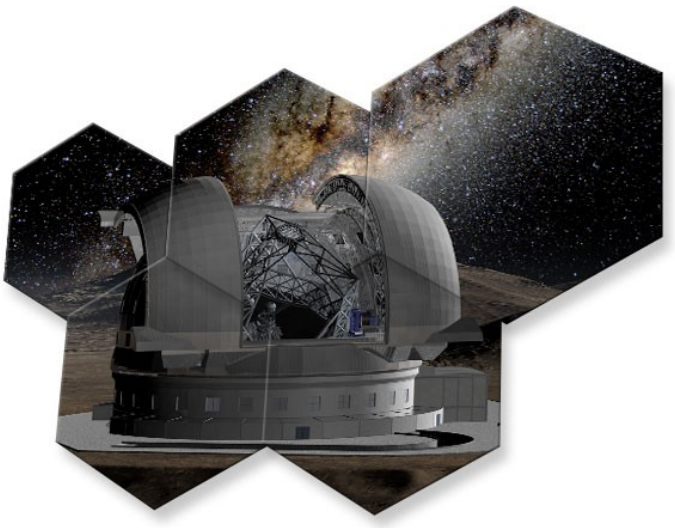

Fig. 1: Il telescopio E-ELT di prossima generazione
(ESO Courtesy)

L'esigenza di conservazione, standardizzazione ed interoperabilità di enormi set di dati distribuiti, unita alla necessità di poter accedere, recuperare, analizzare, estrarre e di integrare i dati in modo semplice, rende necessario l'utilizzo di tecniche computazionali per l'estrazione di conoscenza circa i dati, dove per conoscenza intendiamo modelli, correlazioni, classificazioni, etc [REF.1]. Tutto questo significa fare Data Exploration. Se a questo uniamo la possibilità di compiere scoperte non previste o prevedibili a priori, significa compiere anche operazioni di Data Mining (DM).

Il progetto Data Mining & Exploration (DAME) nasce proprio con lo scopo di mettere a disposizione della comunità astronomica un ambiente per effettuare data mining su dati astronomici. Il progetto consiste di una Suite di servizi per il trattamento di enormi quantità di dati (Massive Data Sets, MDS).

Nello studio delle caratteristiche della Suite abbiamo considerato l'architettura della piattaforma (Par.1) e le tecnologie utilizzate (Par. 2) per la realizzazione. Successivamente abbiamo analizzato le funzionalità attualmente integrate nella Suite (Par. 3), i formati trattati (Par. 4) e la capacità di estensione delle funzionalità e degli algoritmi (Par. 5).

. L' architettura della Suite

In primo luogo ho analizzato l'architettura di DAME per avere un'idea generale delle sue caratteristiche progettuali.

L'architettura modulare rende l'infrastruttura della Suite facilmente aggiornabile e intercambiabile, senza modificare l'intera struttura (vedi Fig. 2). Questo significa che ogni componente implementa uno specifico insieme di funzioni e le relazioni tra i componenti sono ben definite, permettendo la sostituzione o l'aggiornamento di una o più componenti

senza apportare modifiche strutturali all'architettura. La Suite è costituita da cinque componenti che comunicano tra di loro in modo asincrono e standard (XML-based), e svolgono le seguenti funzionalità:

1. Il Driver (DR) ha il compito di gestire al livello hardware l'ambiente di elaborazione, in termini di:
    - file di I/O degli esperimenti con operazioni di download, upload, copia, cancellazione e recupero dei file interessati;
    - esecuzione di esperimenti, memorizzazione e manipolazione di insiemi di dati (dataset) in ingresso;
    - interfaccia con la macchina con operazioni a seconda del tipo di piattaforma utilizzata.
2. Il Data Mining Model (DMM) contiene le librerie degli algoritmi di DM organizzati sotto forma di API (Application Program Interface) ed interfacciate con il sistema centrale attraverso un modulo di wrapping standard in Java, responsabile della configurazione dei parametri interni e della gestione a runtime dell'esecuzione dell'algoritmo.
3. Il Registry & Database (REDB) ha il compito di gestire tutte le informazioni relative agli utenti, alle sessioni di lavoro, agli esperimenti ed ai dataset utilizzati per gli esperimenti.
4. Il Framework (FW), di tipo stateless e restful, ha il compito di gestire e coordinare tutti gli altri componenti per:
    - esecuzione e monitoraggio degli esperimenti richiesti dagli utenti attraverso il DR;
    - gestione dei file attraverso i componenti DR e REDB;

- gestione degli account utente (registrazione e autenticazione) attraverso il REDB;
- gestione delle funzionalità attraverso il DMM;
- gestione degli errori.

5. Il Front End (FE), di tipo client-server e organizzato secondo il pattern MVC, ha il compito di gestire le interazioni con l'utente:
   - fornire un'interfaccia utente semplice e intuitiva, mediante la quale l'utente possa gestire le sue sessioni di lavoro, creare e lanciare esperimenti, fare upload e download di file;
   - gestire la comunicazione con il FW e interpretare i risultati.

La Suite DAME è indipendente dalla piattaforma grazie alla sua astrazione attraverso il componente DR, che favorisce la separazione logica delle caratteristiche funzionali dalla loro implementazione. In tal modo si ha un appropriato accesso alle risorse da parte di tutte le funzionalità specifiche della Suite. La realizzazione avviene attraverso una classe denominata Driver (DR) Management System, che gestisce per l'appunto l'ambiente di elaborazione implementando una interfaccia a basso livello con l'ambiente di calcolo al fine di consentire all'applicazione FW di accedere alle risorse tramite il driver specifico su piattaforme diverse (Stand-Alone, CLOUD o GRID) (Fig. 3). In particolare è utilizzato per la gestione di I/O (caricamento e memorizzazione), formattazione dei dati dell'utente, scheduling dei job, gestione della memoria e re-direzione dei processi.

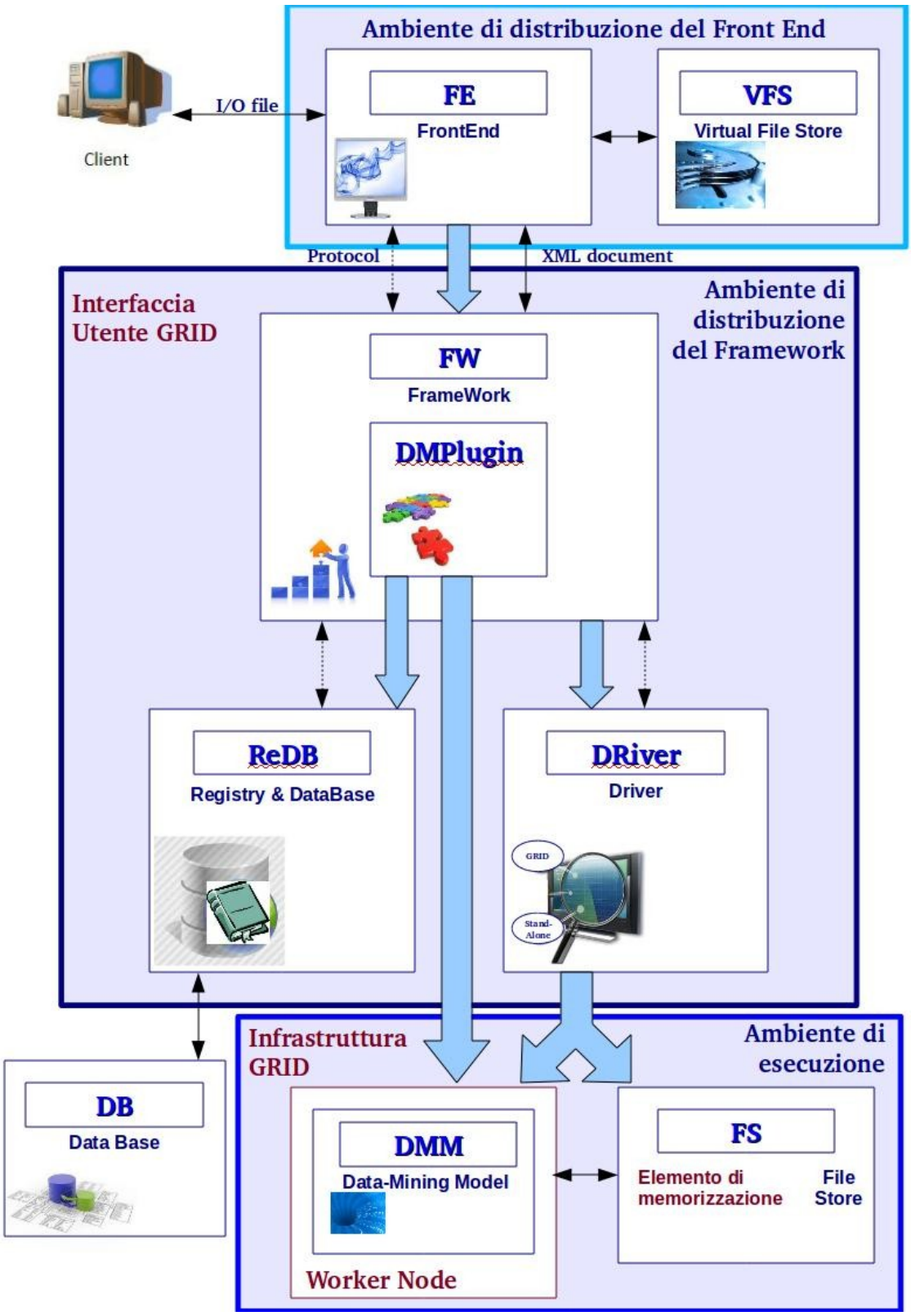

Fig. 2: L'architettura della Suite DAME

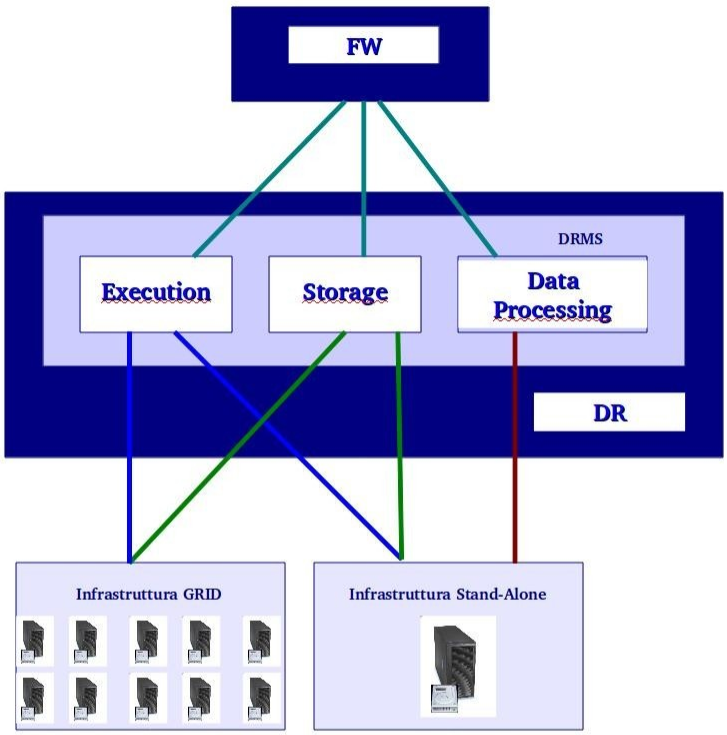

Fig. 3: Il componente Driver (DR)

L'espandibilità è una delle caratteristiche più importanti dell'architettura della Suite ed uno degli obiettivi principali che si sono posti i progettisti del team DAME. La sua realizzazione avviene attraverso un sub-componente del FW, denominato Data Mining Plugin (DMPlugin). Il principio secondo cui è stato sviluppato è quello di strutturare le funzionalità e gli algoritmi sotto forma di plug-in, permettendo l'estensione della Suite senza doverne modificare l'architettura interna. Ciò risulta particolarmente utile nel settore astrofisico, dove l'astronomo classico ha esigenze di continuare ad usare le proprie routine di riduzione dati sui moderni archivi basati su MDS dislocati in vari data center sparsi nel mondo. Tutto questo sfruttando potenza di calcolo e capacità di storage pressoché illimitate, gestite in modo trasparente dalla piattaforma DAME, semplicemente integrandola con il plug-in che rende eseguibile la routine dell'astronomo in modo standard. La configurazione dei plug-in nella Suite avviene attraverso un'applicazione Java, dotata di interfaccia grafica e di una procedura amministratore per la sua installazione, attivazione o disattivazione nell'infrastruttura. Questo

meccanismo di integrazione di funzionalità e algoritmi è particolarmente rilevante per lo svolgimento del mio lavoro nella Suite, per questo motivo lo vedremo più nel dettaglio nel prossimo paragrafo (Par. 5).

L'interfaccia verso l'utente è stata progettata seguendo il pattern architetturale Model-View-Control (MVC) (Fig. 4), che prevede la suddivisione modulare di compiti tra tre ruoli identificati in tre oggetti indipendenti:

- Il FE, nel ruolo di Model, rappresenta la parte logica dell'applicazione e si occupa dell'elaborazione e dell'accesso ai dati.
- La GUI, nel ruolo di View, ha il compito prendere i dati dal Model ed organizzarli per mostrarli all'utente.
- Il FW, nel ruolo di Controller, gestisce le richieste, le elabora e le indirizza verso l'oggetto destinatario. E' il "vero responsabile" delle azioni degli utenti.

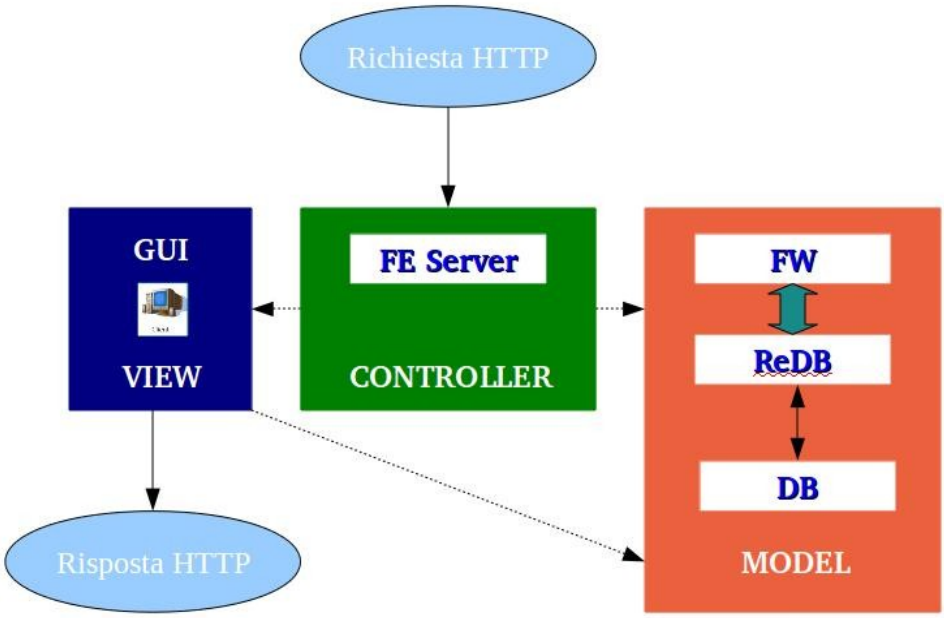

Fig. 4: Il modello MVC nella Suite

Per quanto riguarda la memorizzazione delle informazioni, utilizza un DataBase (DB) locale per la gestione degli utenti e la loro area di lavoro (workspace), oltre ad un DB distribuito per i dati. Il tutto è implementato attraverso il modello Entità-Relazione (ER).

Riassumendo le caratteristiche architetturali alla Suite sono:

1. standardizzata, in termini di comunicazione interna (XML-based), di rappresentazione dei dati (VO compatibile);
2. indipendente dalla piattaforma (Stand-Alone, CLOUD, GRID);
3. espandibile in termini di funzionalità ed algoritmi, dotandosi di un wrapper Java che permette di utilizzare in modo omogeneo librerie realizzate in qualsiasi linguaggio di programmazione;
4. orientata ai servizi;
5. basata su tecnologie WEB;
6. non richiede installazione o memorizzazione di moduli o dati lato utente (basta un semplice browser), permettendone il pieno utilizzo mediante qualunque dispositivo con accesso a Internet (dal laptop allo smartphone);
7. modulare.

. Le tecnologie utilizzate

Lo studio effettuato ha compreso una analisi delle tecnologie utilizzare per la progettazione, di cui discuteremo in questo paragrafo. In tal modo cercheremo di fornire la

conoscenza di base utile per comprendere le scelte progettuali del team DAME per lo sviluppo della Suite.

Il World Wide Web (WWW o Web) è stato uno degli avventi tecnologici che ha cambiato in modo radicale la nostra società, la sua diffusione a livello globale ha portato ad un notevole incremento di ogni genere di servizi basati sulla rete internet mondiale, messi a disposizione della comunità attraverso le applicazioni web (webapp). Le *webapp* sono applicazioni accessibili attraverso una rete internet o intranet. Il Web è un esempio di webapp client/server distribuita operante sulla rete Internet e reti intranet Transmission Control Protocol/Internet Protocol (TCP/IP), attraverso il protocollo HyperText Transfer Protocol (HTTP). Per intenderci, tutti i computer collegati a Internet sono chiamati detti *host* (o *terminali* o *end system*), una webapp client-server è un'applicazione in cui un programma *client* è un programma eseguito su un terminale che usa un servizio da un programma *server* eseguito su un altro terminale (o sullo stesso). Mentre i protocolli definiscono il formato e ordine dei messaggi scambiati tra due o più entità comunicanti e le azioni a seguito della trasmissione e/o ricezione di un messaggio o di altri eventi. In particolare:

- TCP è un protocollo a livello di trasporto, utilizzato per fornire un servizio di comunicazione tra i terminali per la trasmissione dei pacchetti;
- IP è un protocollo di rete, usato per specificare il formato dei pacchetti che sono scambiati tra router e tra terminali;
- HTTP è un protocollo a livello di applicazione, usato come principale sistema per la trasmissione di informazioni sul Web.

L'evoluzione del Web è stata rapida, caratterizzata inizialmente da uno scambio di

informazioni mono-direzionale in cui il client richiedeva al server una risorsa ricevendo in cambio un documento in formato elettronico. Ben presto la comunicazione divenne bidirezionale, permettendo al client di passare al server alcune informazioni utilizzate per parametrizzare la richiesta oppure ad acquisire dati (ad esempio in un database), introducendo così la programmazione lato server. Si è passati così dal web statico al web dinamico, ma l'evoluzione non si è fermata qui. Ad oggi sono stati introdotti nuovi strumenti che permettono di aggiungere dinamicità al browser, come standard per la grafica in grado di gestire al meglio animazioni e video attraverso la programmazione lato client. La programmazione lato client è una soluzione efficiente ai problemi di manutenibilità e tempi di risposta (dipendenti dalla dimensione dei dati trasferiti, carico del server e carico della rete) legati all'architettura server-browser, in cui tutto il carico di lavoro è affidato al server. Alla base della progettazione della Suite vi sono proprio queste caratteristiche di interattività.

La Suite offre un servizio web attraverso una Rich Internet Application (RIA) implementata nella componente FE[4]. Una RIA è webapp con le stesse caratteristiche di interattività e efficienza delle applicazioni desktop, ed è realizzata attraverso la programmazione lato client. L'applicazione non richiede installazione o memorizzazione di moduli o dati lato utente (basta un semplice browser), permettendone il pieno utilizzo mediante qualunque dispositivo con accesso a Internet (dal laptop allo smartphone). Le operazioni svolte client-side attraverso il browser sono relative a:

- chiamate ad un metodo RPC;
- de-serializzazione dell'oggetto ricevuto dal server;
- inserimento dell'oggetto de-serializzato nei widget della GUI;
- validazione dei form per garantire la correttezza dei dati immessi dall'utente.

4 Vedi RIF. 4

Le operazioni svolte server-side sono:

- controllo della correttezza dei dati ricevuti;
- invio di una richiesta HTTP al FW;
- attesa di una risposta dal FW in un documento XML;
- parsing del documento ricevuto e wrapping del contenuto dell'XML in oggetti Java;
- invio della risposta al client.

Lo sviluppo dell'applicazione fa uso della tecnica Asynchronous JavaScript and XML (AJAX), una delle tecniche di sviluppo più note per webapp RIA. Esattamente AJAX è un nuovo modo di utilizzare le tecnologie già esistenti, quali:

- HyperText Markup Language (HTML) è un linguaggio usato per controllare la struttura dei documenti ipertestuali;
- Extensible Hypertext Markup Language (XHTML);
- Cascading Style Sheets (CSS);
- JavaScript è un linguaggio di scripting basato su oggetti che permette di effettuare computazioni e manipolare oggetti computazionali all'interno di un ambiente ospite;
- Document Object Model (DOM), è una forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti;
- Extensible Markup Language (XML) è un linguaggio di mark-up progettato per

descrivere i dati;

- Extensible Stylesheet Language Transformations (XSLT) è un linguaggio per implementare trasformazioni su documenti XML;
- l'oggetto XMLHttpRequest è un set di API che possono essere usate per trasferire documento XML o altri dati.

AJAX consente una comunicazione client-server asincrona attraverso XMLHttpRequest. Il principale vantaggio è la possibilità di aggiornare dinamicamente una pagina web senza ricaricare l'intera pagina, in tal modo sarà possibile velocizzare l'applicazione. L'idea di base è quella di diminuire il numero di scambi HTTP (Fig. 5), questo è possibile grazie a client più potenti. Quindi il client scaricherà più informazioni del necessario in modo da fare molta di questa navigazione fuori linea usufruendo delle informazioni ottenute, comunicando con il server solo quando è necessario. E' chiaro che non viene scaricata tutta l'applicazione spesso in funzione della potenza della macchina.

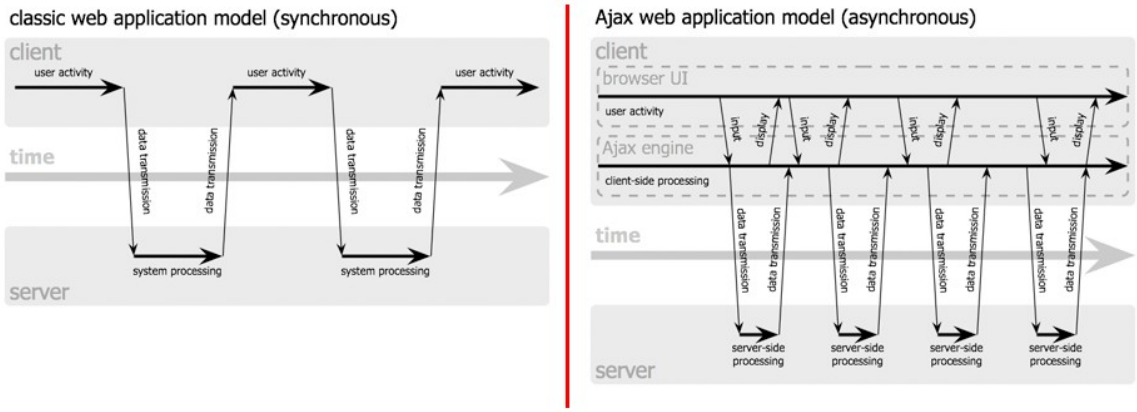

Fig. 5: L'interazione sincrona vs interazione asincrona AJAX

Nell'immagine seguente è mostrata in che modo avviene la comunicazione per una richiesta AJAX (Fig. 6).

In particolare, utilizza JavaScript per interagire con la DOM, per costruire dinamicamente la pagina a seconda della richiesta dell'utente. XSLT trasforma lo stile di presentazione dell'informazione visualizzata. L'oggetto XMLHttpRequest è fondamentale poiché è quello che mi permette di mandare la richiesta HTTP, scartare ed elaborare la richiesta HTTP, inviare la risposta. Uno dei principali svantaggi dell'utilizzo di AJAX è che bisogna considerare nello sviluppo le caratteristiche di diversi tipi di browser.

L'applicazione è stata sviluppata attraverso l'innovativo tool open source Google Web Toolkit (GWT)[5], che permette di creare e manutenere complesse applicazioni front end Javascript scritte in Java. Tra i vantaggi offerti dall'utilizzo di GWT citiamo:

- non è necessario gestire le incompatibilità e le peculiarità del browser;
- gestisce le operazioni che dovrebbero essere in Javascript o attraverso le DOM mediante Java ;
- sfrutta vari strumenti del linguaggio di programmazione Java per la scrittura / debug / test.

5 Vedi RIF. 8

GWT permette di scrivere applicazioni tramite l'uso del linguaggio Object Oriented Programming (OOP) Java successivamente convertite in Javascript, compatibile con tutti i browser web.

La Suite è stata progettata attraverso il paradigma OOP e scritta in linguaggio Java. La scelta del linguaggio di programmazione è stata fatta in base alle caratteristiche di sicurezza e portabilità. La portabilità è un requisito fondamentale della nostra applicazione essendo un ambiente di elaborazione distribuita, questo ci permette di renderla indipendente dalla piattaforma grazie alla JVM che si occupa di interpretare il bytecode in linguaggio macchina, qualunque sia la macchina su cui avviene l'elaborazione. Ogni macchina avrà installata la JVM appropriata alle sue caratteristiche. La sicurezza è altrettanto essenziale, in quanto grazie ad una serie di meccanismi il linguaggio non permette istruzioni per alcuni comportamenti insicuri, ad esempio l'uso dei puntatori. Inoltre offre un supporto per l'esecuzione del codice da sorgenti remote, il codice (scaricato da un server) viene eseguito in un'area tecnicamente detta sandbox, che non permette al codice di eseguire operazioni all'esterno della stessa. In particolare, per quanto riguarda la GUI la scelta combinata di Java e GWT, rispetto all'alternativa di sviluppo attraverso un linguaggio di scripting, ha portato a diversi vantaggi , tra cui:

- la possibilità di effettuare il controllo statico in linguaggio Java aumenta la produttività e riduce gli errori.
- la possibilità di catturare in compilazioni gli errori comuni, ad esempio errori di battitura, piuttosto che da parte degli utenti a runtime.
- la disponibilità online di guide e codici.

La Suite implementa un servizio web in cui i servizi sono mostrati come risorse ed

identificate univocamente da un Uniform Resource Identifier (URI) e ogni richiesta è considerata una transazione indipendente dalle precedenti richieste. Fornisce quindi un servizio web RESTful e stateless.

Infine la Suite grazie al componente DR agisce su piattaforme Stand-Alone e di calcolo distribuito resource-based (GRID) o service-based (CLOUD), secondo gli standard internazionali dell'International Virtual Observatory Alliance[6] (IVOA).

. Le funzionalità

La prima versione della Suite mette a disposizione funzionalità di classificazione e regressione, ben presto sarà arricchita con funzionalità per il clustering.

Ogni funzionalità può supportare le seguenti modalità di esecuzione (casi d'uso):

- Il train, che prevede un addestramento, ossia un processo che dato in input un insieme di dati noti, relativi ad osservazioni di un determinato fenomeno, ne apprenda le dipendenze.
- Il run, che permette l'esecuzione del modello implementato, prendendo in input una combinazione di parametri.
- Il test, che permette di valutare l'accuratezza di un addestramento, prendendo in input un insieme di dati di cui conosciamo i risultati.
- Il full, che comprende tutte le funzionalità precedenti, prende in input un insieme di dati per l'addestramento, un insieme di dati per valutare l'accuratezza e un

6 www.ivoa.net

insieme di dati per l'esecuzione del modello.

La classificazione opera il raggruppamento di singoli elementi in base a informazioni su una o più caratteristiche interne e attraverso una procedura supervised (training con dati noti).

La regressione è utilizzata per effettuare previsioni (ad esempio per prevedere dati futuri di una serie temporale), inferenza statistica, per testare ipotesi o per modellare delle relazioni di dipendenza.

Il clustering è utilizzato per la suddivisione in clusters di un insieme di elementi rappresentati in forma varia all'interno di uno spazio di parametri, cioè identificare dei sottogruppi accomunati da determinate caratteristiche (parametri).

I modelli attualmente disponibili nella Suite sono:

- Support Vector Machine (SVM): un algoritmo di classificazione o regressione in grado di lavorare in uno spazio fortemente multidimensionale, detto "feature space".
- Multi-Layer Perceptron (MLP): è un modello di classificazione e/o regressione di dati non linearmente separabili, che fa uso dell'algoritmo Back Propagation di apprendimento.
- Multi-Layer Perceptron with Genetic Algorithm (MLPGA): è un modello di classificazione e regressione basato su un sistema ibrido fondato sul paradigma dell'evoluzione statistica darwiniana.
- Multi Layer Clustering & SOM (Self-Organizing Maps): modello unsupervised, basato su architetture gerarchiche, i cui strati possono contenere diverse configurazioni di modelli unsupervised, in particolare il modello neurale SOM.

. I formati dei dati

Uno dei punti essenziali della progettazione del software da me sviluppato è la compatibilità dei formati dei dati. L'importanza dei formati dei dati è dovuta al fatto che i dati astronomici prevedono l'utilizzo di formati particolari, i quali oltre a contenere i dati effettivi contengono anche dei parametri descrittivi dei dati stessi. I formati trattati dalla Suite sono i formati definiti secondo lo standard internazionale dall'IVOA, quali:

- *Flexible Image Transport Specification* (FITS). Il formato FITS è il formato standard memorizzare, trasmettere e manipolare dati scientifici approvato dalla NASA e dall'International Astronomical Union (IAU). Utilizzato anche per immagini scientifiche ad alta risoluzione. Progettato principalmente per archiviare dati scientifici contiene quindi un header con molte informazioni relative alla calibrazione fotometrica e spaziale, insieme a vari metadati che riportano dettagli astrometrici, condizioni di seeing e atmosferiche, strumento usato etc [RIF. 6].

- VOTABLE. Il formato VOTable è un documento XML standard per lo scambio di dati tabulari. E' caratterizzato da un header e da un elemento VOTABLE, in cui sono definiti i metadati relativi ai parametri generici del documento e le risorse (elemento RESOURCE) in cui sono rappresentate le tabelle. Ogni tabella definisce parametri descrittivi attraverso attribuiti degli elementi FIELD e PARAM della risorsa e i dati effettivi [RIF. 5].

- *American Standard Code for Information Interchange* (ASCII). L'ASCII è un sistema di codifica dei caratteri comunemente utilizzato.

- *Comma-Separated Values* (CSV). Il formato CSV basato su file di testo è spesso

utilizzato per lo scambio di dati tabulari. La caratteristica di base di questo formato è la rappresentazione delle sue righe attraverso linee di testo, l'identificazione dalle colonne avviene attraverso un convenzionale carattere separatore (definito dall'utente).

1. Manipolazione dei dati

Per motivi di incompatibilità di alcuni algoritmi e funzionalità, che sono vincolati solo all'utilizzo di alcuni formati, la Suite per permettere il loro utilizzo si avvale di un potente tool per la gestione formati. La gestione dei set di dati è realizzata in modo semplice attraverso le Starlink Tables Infrastructure Library (STIL) [RIF. 7], un insieme di librerie di classi Java per la gestione di dati e meta-dati. In particolare ci permette la conversione di un insieme di dati da un formato all'altro, ci permette di accedere ai dati di ogni formato attraverso una struttura standardizzata. Questo software sviluppato per l'utilizzo con tavole astronomiche, trova applicazione in ogni campo in cui si trattano dati tabulari di ogni genere.

Attraverso metodi STIL è possibile determinare:

- le dimensioni della tabella contando il numero di righe e il numero di colonne;
- le informazioni sulle colonne della tabella, come:
  - nome attribuito alla colonna;
  - unità di misure dei valori contenuti nella colonna;
  - la classe Java dei valori contenuti nella colonna.

Inoltre contiene metodi per l'accesso sequenziale alla tabella dai dati attraverso metodi per la lettura nella struttura creata e metodi per la conversione dei formati.

. Estensione di DAME

Un aspetto fondamentale per la realizzazione del progetto da me svolto è la capacità della Suite di espandere la repository delle proprie funzionalità e algoritmi. I componenti che si occupano dell'implementazione di questo requisito sono DMPlugin[7] e DMM[8]. Come anticipato in precedenza (Par 1), la Suite realizza l'integrazione strutturando funzionalità e algoritmi sotto forma di plug-in.

Nell'analisi del meccanismo di integrazione intervengono due oggetti:

- i DMPlugin, che rappresentano le funzionalità messe a disposizione della Suite;
- i DMM rappresentano algoritmi di data mining disponibili.

La relazione tra i due è la seguente: una funzionalità DMPlugin per effettuare un esperimento utilizza uno o più modelli di data mining.

1. Il DMPlugin

Ogni funzionalità DMPlugin implementa un'interfaccia attraverso la quale i componenti esterni possono configurare il plug-in, eseguire il relativo esperimento e gestire il flusso dei dati. Essa contiene informazioni riguardanti il plug-in, il proprietario e le modalità di esecuzione (train, test, run,...). Per ogni modalità di esecuzione contiene informazioni su parametri di input, file di input e di output. Nel DMPlugin vengono richiamati i metodi

7 Vedi RIF. 14 e RIF. 15
8 Vedi RIF. 16 e RIF. 17

disponibili nel DMM.

2. Il DMM

Il DMM è una libreria di classi. Ogni volta che viene introdotto un nuovo modello deve essere introdotta una nuova classe che implementa l'interfaccia comune DMMInterface (Fig. 7), la quale garantisce l'utilizzo in modo omogeneo di librerie realizzate in qualsiasi linguaggio.

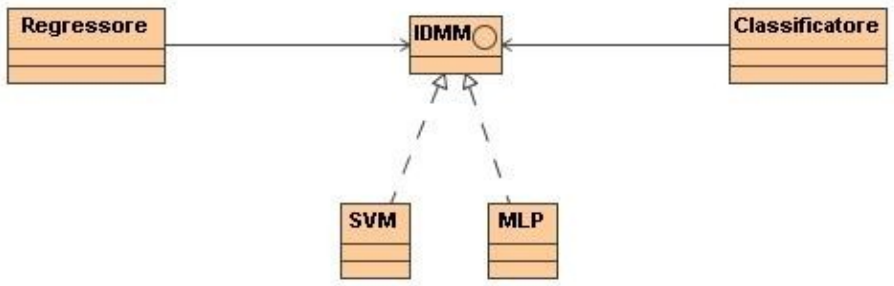

Fig. 7: Il DMM Class diagram

3. Il meccanismo di funzionamento

Per comprendere meglio come avviene l'esecuzione di una generica funzionalità, esamineremo passo dopo passo le azioni che la Suite compie per eseguire l'esperimento [RIF. 3].

Il primo passo per l'integrazione è la configurazione del plug-in attraverso un'interfaccia dedicata (Tool Java DMPlugin Wizard, Par. 5.4), introducendo le direttive per l'esecuzione di

un nuovo modello e/o di una nuova funzionalità, generandone in automatico il codice di interfaccia per il nuovo DMPlugin.

Il DMPlugin viene attivato nel momento in cui il FE richiede al FW di creare un nuovo esperimento. In primo luogo viene effettuata la configurazione del DMPlugin, creando anche i DMM necessari all'esecuzione. Il controllo è poi passato al DR per iniziare l'esecuzione dell'esperimento. A questo punto il DMPlugin fa partire l'esperimento invocando il metodo run del DMPlugin (Vedi Fig. 8).

Questo meccanismo dà la possibilità all'utente di distribuire una propria applicazione di DM e in contemporanea estendere la Suite con tools e algoritmi.

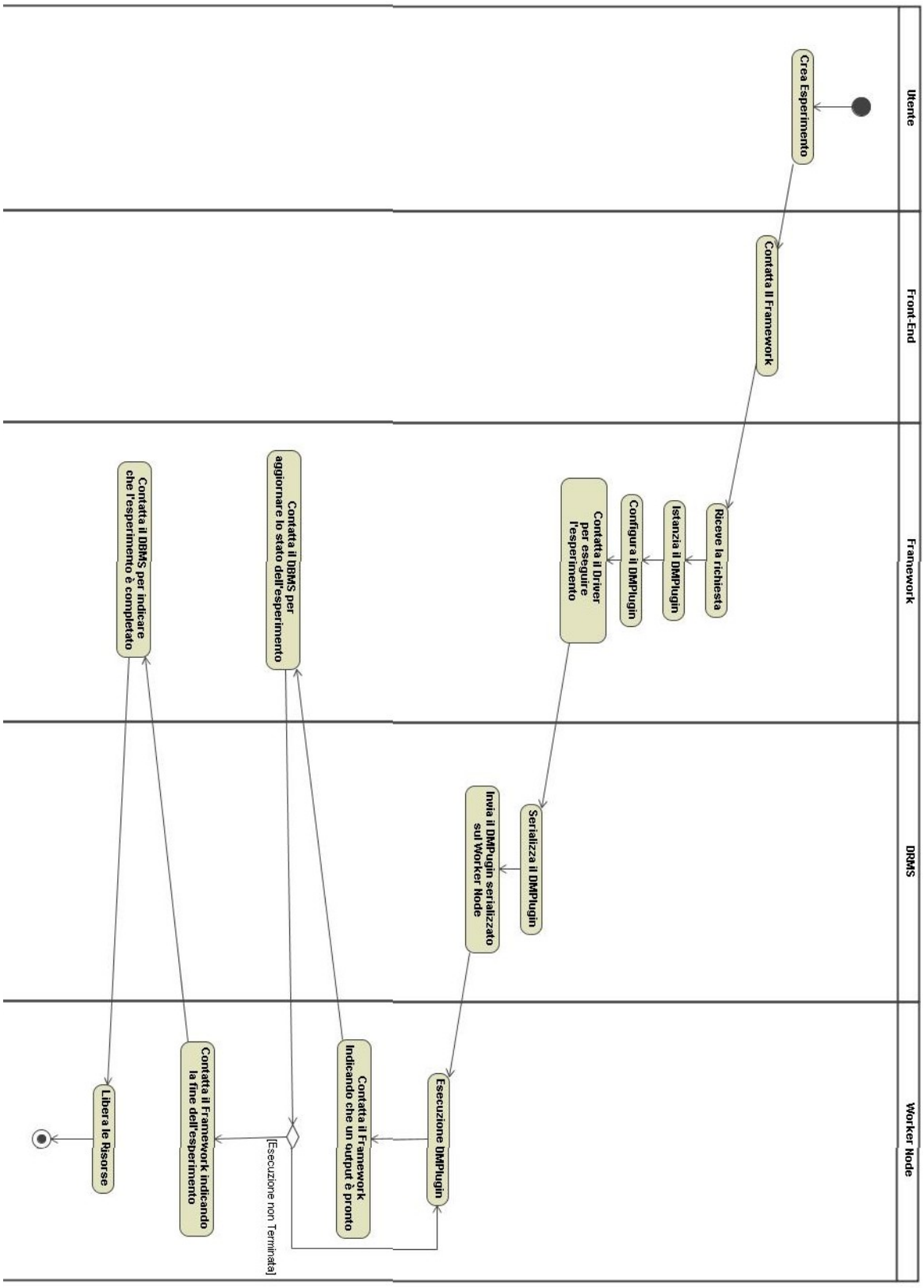

Fig. 8: Il sequence Diagram DMPlugin su GRID

4. Il DMPlugin Wizard

Il DMPlugin Wizard[9] permette l'estensione delle funzionalità e degli algoritmi attraverso una procedura automatica, la quale genera una classe che implementa l'interfaccia AbstractDMPlugin. La procedura prevede la configurazione (Fig. 9) del plug-in varie informazioni, fra cui fornitore e casi d'uso, come segue:

- il nome del plug-in, il riferimento alla documentazione, la versione del software e il dominio, nella sezione “Plugin information”;
- il nome e la mail del proprietario, nella sezione “Owner information”;
- le modalità di esecuzione previste dal nostro plugin, nella sezione “Running Modes Information”.

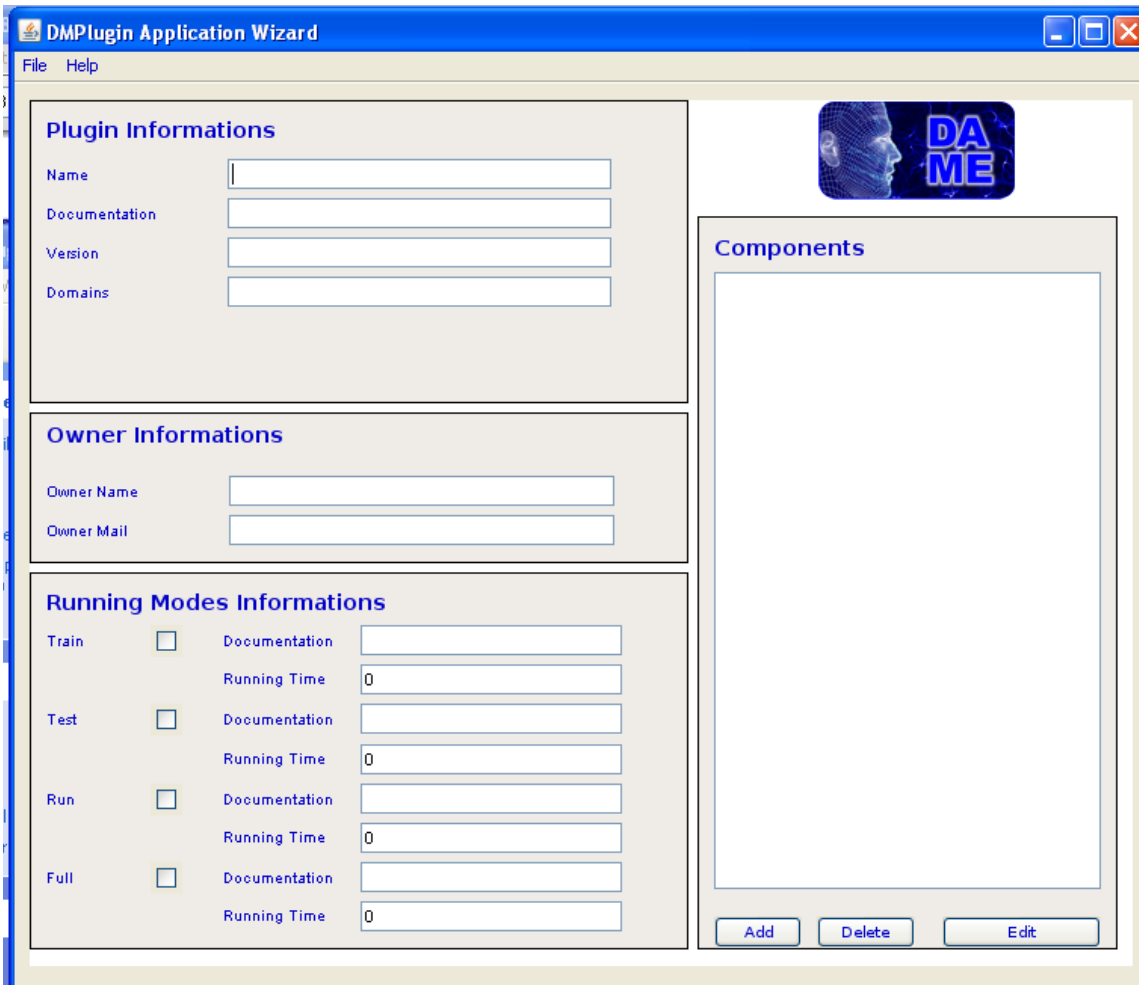

Fig. 9: Form di configurazione del plugin

9 REF. 1

Per ogni caso d'uso selezionato è possibile configurare le componenti “Field”, “Input File” e “Output File” (Fig. 10). I “Field” sono i parametri impostabili per orientare l'esecuzione dell'esperimento, gli “Input File” sono i datates di ingresso all'esperimento e gli “Output File” sono i file prodotto al termine dell'esperimento.

Per la configurazione dei “Field” sono necessarie informazioni come il nome della variabile del campo, il tipo del valore, la descrizione e altro (Fig. 11).

Fig. 11: Form di configurazione del caso d'uso

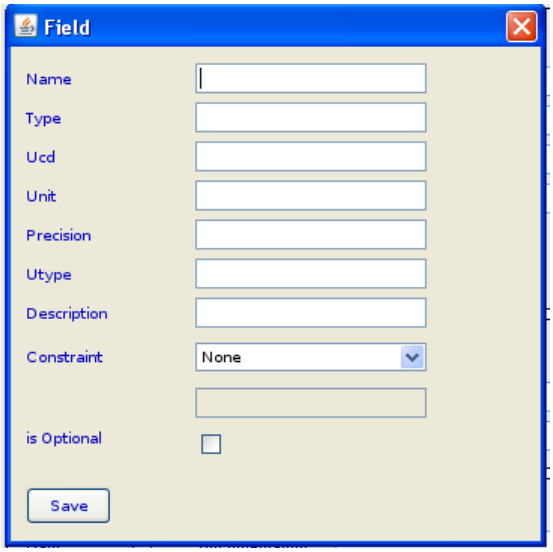

Fig. 10: Form di configurazione del Field

Per la configurazione dei “Input File” sono necessarie informazioni come il nome della variabile del file, la descrizione e l'etichetta del file (Fig. 12).

Per la configurazione dei “Output File” sono necessarie informazioni come il nome della variabile del file, la descrizione ed il formato del file (Fig. 13).

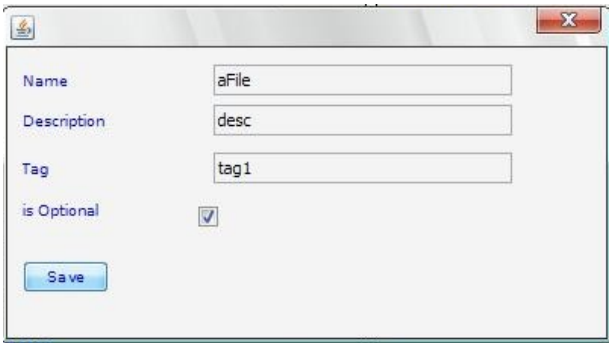

Fig. 12: Form di configurazione dei File di input

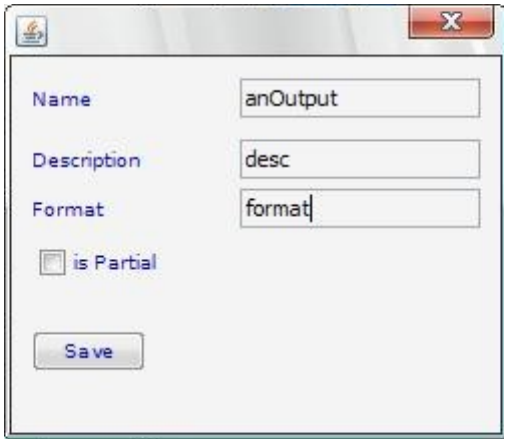

Fig. 13: Form di configurazione dei File di output

Completata la configurazione del plugin è possibile salvarla mediante la voce “Save” del menù “File” (Fig. 14). Questa operazione genera un file che può essere caricato mediante la voce “Load”. Il codice relativo può essere generato automaticamente tramite la voce “Generate Code” (Fig. 14), la quale crea un file sorgente java in cui può essere integrato il codice per le modalità d'esecuzione previste.

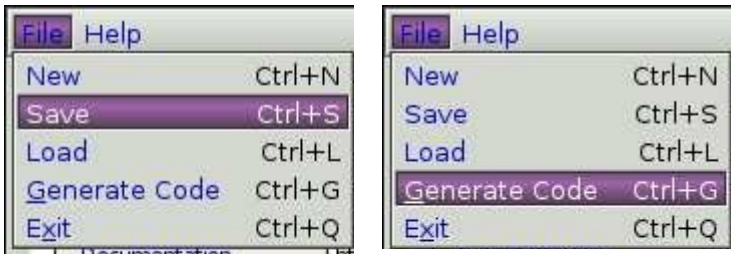

Fig. 14: Menù wizard - Save e Generate code

5. Esportare e distribuire il nodo come un plugin

Per l'integrazione della funzionalità nella Suite è necessario che l'amministratore del sistema provveda all'installazione del plug-in. Nel caso il plugin richieda l'introduzione di un nuovo modello, questo deve essere introdotto opportunamente nel componente DMM.

CAPITOLO 3

KNIME

Un secondo passo dello studio di fattibilità è stato quello di effettuare il reverse engineering del KNIME kernel allo scopo di valutare la compatibilità con la Suite e garantire l'interoperabilità tra le due piattaforme.

KNIME[10] è una piattaforma modulare per la creazione di pipeline di data analysis per l'integrazione, l'elaborazione, l'analisi e l'esplorazione dei dati. Una delle caratteristiche principali è la capacità di eseguire selettivamente tutte o alcune fasi dell'analisi, inoltre è possibile visualizzare interattivamente dati e modelli.

Nello studio delle caratteristiche di KNIME sono state valutate in primo luogo le caratteristiche generali (Par. 1), l'architettura e le tecnologie utilizzate (Par.2). Successivamente abbiamo studiato in che modo sono rappresentate internamente le componenti KNIME (Par. 3) e come avviene la comunicazione con il kernel (Par. 4). Abbiamo inoltre studiato le funzionalità messe a disposizione (Par. 5) e la capacità di estensione della repository (Par. 6). Infine, abbiamo approfondito i formati trattati (Par. 7) e definito il modello di progetto KNIME (Par. 8).

10 Maggiori informazioni su http://www.knime.org

. Descrizione generale

I componenti base di KNIME sono:

- il progetto, che rappresenta un esperimento condotto su un insieme di dati osservati relativi ad un determinato fenomeno;
- il workflow, che rappresenta il procedimento da eseguire sui dati per effettuare l'esperimento;
- il nodo, che rappresenta una singola operazione effettuata sui dati.

Il progetto è caratterizzato da uno o più workflow, il quale consiste di una pipeline di nodi connessi da linee per il trasporto di dati o modelli (Fig. 15).

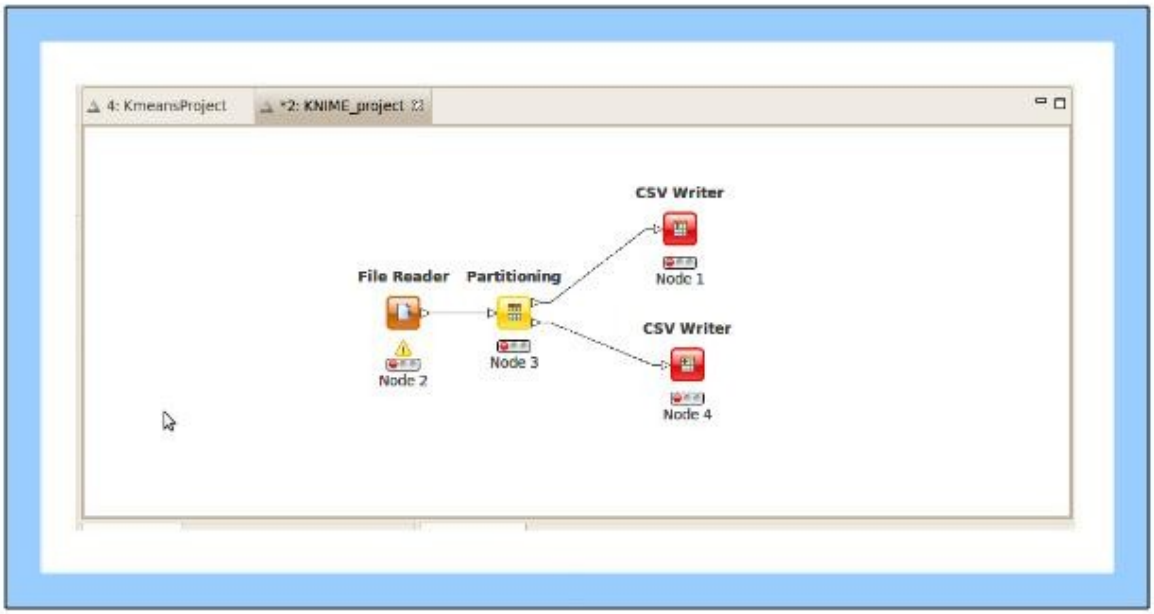

Fig. 15: Il progetto

Il nodo è caratterizzato da (Fig. 16):

- porte di input, attraverso le quali può ricevere dati provenienti dalla porta di output di nodo predecessore;
- porte di output, attraverso le quali può fornire i risultati dell'elaborazione al nodo successore;
- parametri di configurazione, attraverso i quali si può orientare l'esecuzione di una particolare funzionalità (es: nel partizionamento di un insieme di dati si può scegliere di farlo secondo un partizionamento relativo oppure assoluto).

Le porte di input/output sono di due tipologie:

- per i dati, contenenti osservazioni dei fenomeni di cui si vuole effettuare l'esperimento;
- per i modelli, contenenti solo metadati di un particolare insieme di dati modellizzato.

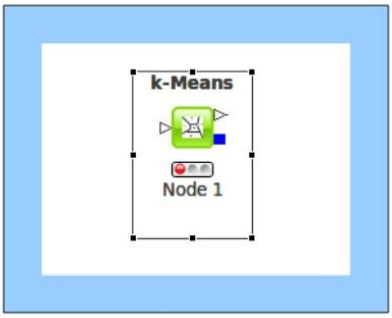

Fig. 16: Il nodo

Due nodi possono essere connessi in una pipeline se e soltanto se la porta di output del nodo di partenza è della stessa tipologia della porta di input al nodo di destinazione.

Un altro componente è il meta-nodo, che può essere visto come un nodo speciale contenente un sub-workflow incapsulato. Uno dei principali vantaggi nell'utilizzo di meta

nodi sta nella possibilità di progettare workflow complessi in modo semplice e modulare. E' rappresentato nella pipeline come un semplice nodo (Fig. 17).

Fig. 17: Il meta-nodo

. L'architettura e le tecnologie

KNIME è scritto in Java e la GUI è implementata come plug-in Eclipse [RIF. 2]. E' stato progettato e realizzato sulla base di tre requisiti:

- creare un ambiente visuale per interfacciarsi con l'utente in modo semplice e interattivo;
- progettare un software che favorisca l'esecuzione su un sistema distribuito;
- creare un ambiente che permetta l'espansione delle funzionalità nel modo più semplice possibile.

Per l'accesso ai MDS viene utilizzata una potente strategia di caching, che permette di accedere alla tabella dati direttamente sul disco rigido nel momento in cui diventa troppo grande per tenere tutte le righe in memoria.

1. Le strutture dati

Un requisito progettuale di KNIME è l'indipendenza tra i nodi e le strutture dati, consentendo una facile distribuzione dell'elaborazione. I dati transitano da un nodo all'altro incapsulati in un particolare formato interno ottenuto mediante la classe chiamata DataTable, mentre l'accesso avviene mediante iterazione su istanze della classe DataRow. Il formato ottenuto contiene i meta-dati relativi alla natura delle sue colonne e i dati effettivi. Ogni riga è caratterizzata da un identificatore univoco (o chiave primaria) e uno specifico numero di oggetti DataCell, contenenti i dati effettivi (Fig. 18).

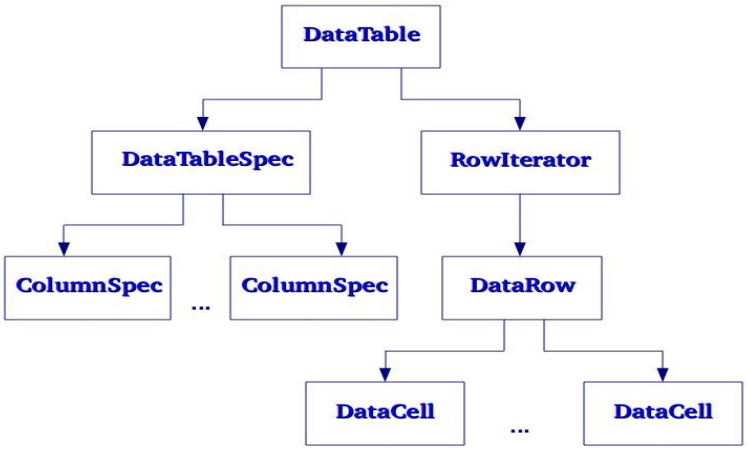

Fig. 18: diagramma UML della struttura dati e le principali classi

2. Il nodo

Il funzionamento del nodo è gestito attraverso la classe denominata Node, per lo svolgimento del suo compito utilizza:

- un'istanza della classe NodeModel, che si occupa dello svolgimento della funzionalità;

- un'istanza della classe NodeDialog, che si occupa dell'impostazione dei parametri per l'orientamento dell'esecuzione;
- una o più istanza della classe NodeView, che si occupa della visualizzazione dei dati o modelli.

Questo schema segue il ben noto modello di progettazione MVC (Fig. 19).

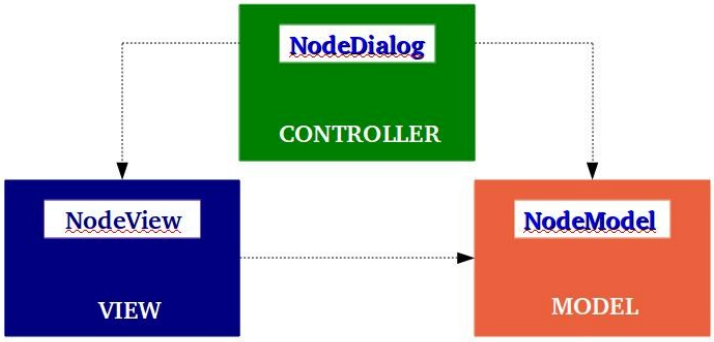

Fig. 19: MVC KNIME

In aggiunta, per le connessioni input e output, ogni nodo ha un numero di istanza di Inport e Outport, che possono trasportare dati o modelli (Fig. 20).

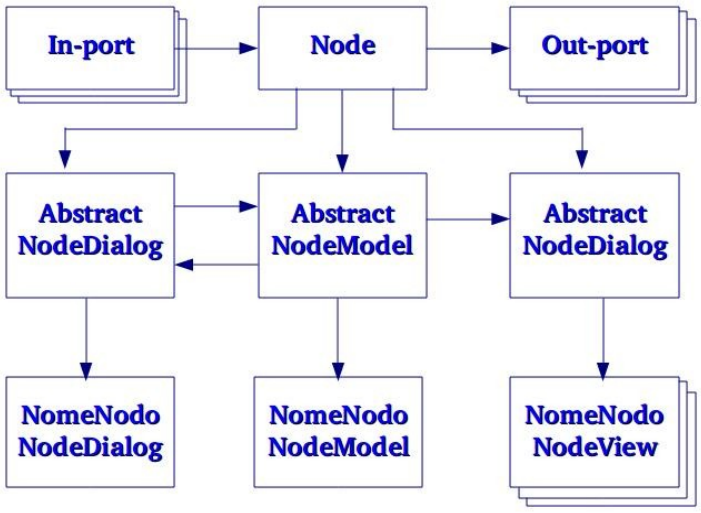

Fig. 20: Diagramma UML del Node e le principali classi ad essi relazionati

3. Workflow

Il Workflow, dal punto di vista tecnico, è un grafo connesso di nodi, o più formalmente Direct Acyclic Graph (DAG) (Fig. 15). Il WorkflowManager gestisce l'inserimento di nuovi nodi, le connessioni orientate tra due nodi e tiene traccia dello stato dei nodi (non configurato, configurato, eseguito) ottenendo l'insieme dei nodi eseguibili. In tal modo è possibile elaborare in modo distribuito il lavoro, grazie alla struttura interna dei dati (Fig. 18) il WorkflowManager può determinare i nodi che devono essere eseguiti lungo la pipeline per eseguire il nodo richiesto dall'utente.

. La rappresentazione interna

Un aspetto importante di KNIME è la rappresentazione interna dei componenti quali il progetto, il workflow, il nodo ed il meta-nodo, per capire come il motore interno di KNIME vede queste componenti. Dallo studio[11] è emerso che il kernel rappresenta le sue componenti attraverso documenti XML, che da componente a componente variano nel numero di documenti e nella loro struttura.

1. Il progetto

Il progetto è una delle componenti fondamentali di KNIME. Tutti i progetti sono localizzati in una cartella workspace (Fig. 21), definita dall'utente all'avvio del programma. Un progetto è caratterizzato da un nome ed un workflow. Nella rappresentazione interna l'unità logica di

11 Vedi RIF. 12 e RIF. 13

un progetto è una cartella, il cui nome è assegnato dall'utente al momento della creazione del progetto. Il progetto è descritto mediante l'utilizzo di file e cartelle localizzate nella cartella del progetto (Fig. 22).

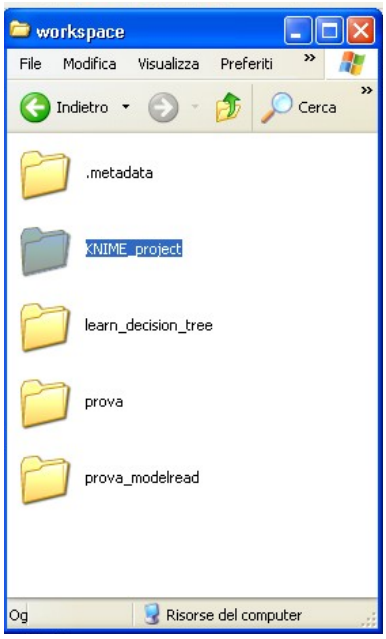

Fig. 21: la cartella workspace

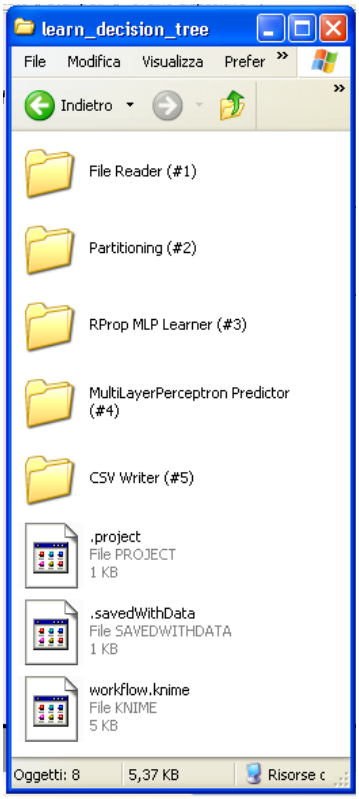

Fig. 22: il contenuto della cartella di progetto

I file di progetto sono:

1. "workflow.knime", che è un documento XML contenente informazioni relative allo stato del progetto, ai nodi del workflow e le loro connessioni;
2. ".project", che è un documento XML contenente informazioni relative al tipo di elemento di KNIME ed alla sua rappresentazione sull'area di lavoro;
3. ".savedWithData", che è un file testuale contenente informazioni relative ai salvataggi del progetto.

Le cartelle di progetto sono rappresentative di un nodo del workflow, esiste quindi una cartella per ogni nodo presente nel workflow (Fig. 23). Ogni cartella è denominata secondo la particolare sintassi: `<nome_nodo> (#<numero>)`

dove *nome_nodo* è il nome della funzionalità e *numero* è un progressivo associato i nodi di un workflow.

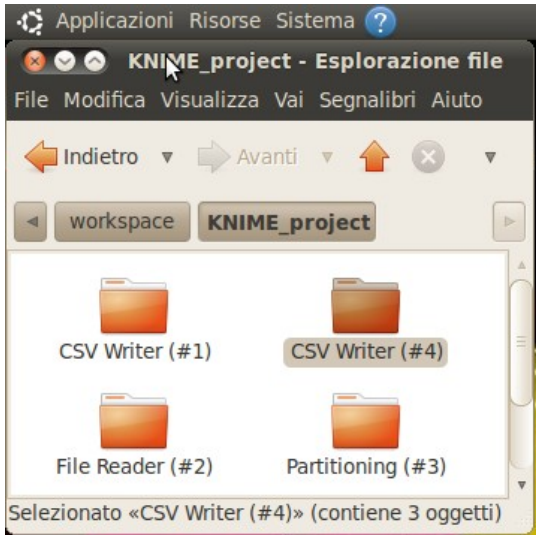

Fig. 23: La rappresentazione dei nodi in un progetto

2. Il workflow

Il workflow è descritto dal file di progetto "workflow.knime". Il documento è sintatticamente strutturato in tre parti (Fig. 24):

1. la prima parte contiene informazioni di progetto;
2. la seconda parte contiene informazioni sulla configurazione dei nodi;
3. la terza parte contiene informazioni sulla configurazione delle connessioni.

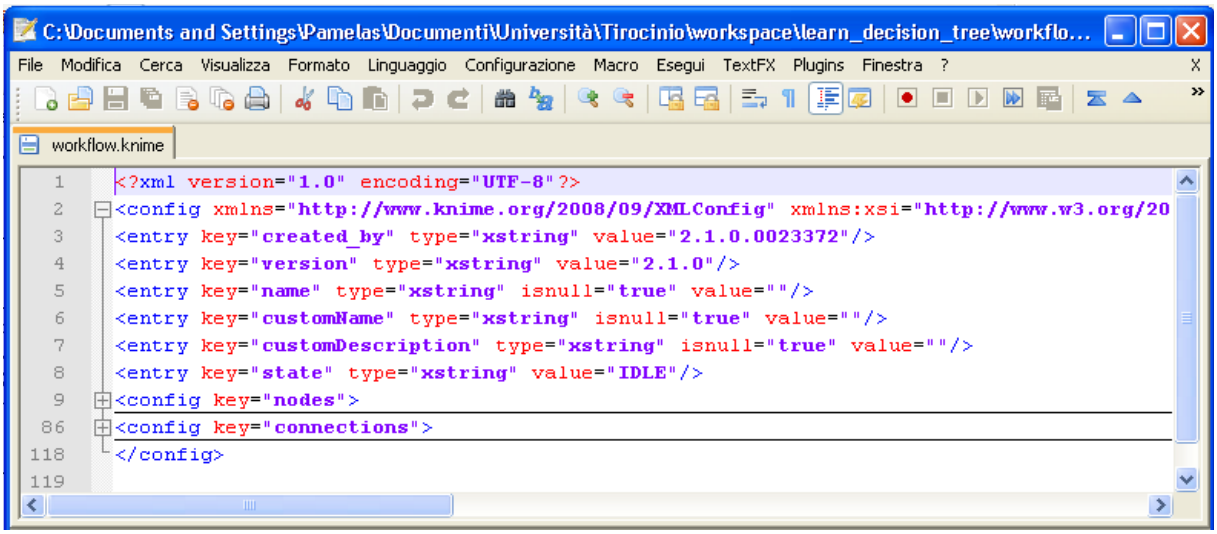

Fig. 24: La struttura del “workflow.knime”

Tra le informazioni contenute nella prima parte del documento la più significativa è lo stato (Fig. 24), che può assumere tre possibili valori:

- “IDLE”, nel caso in cui il progetto non sia configurato;
- “CONFIGURED”, nel caso in cui il progetto sia correttamente configurato;
- “EXECUTED”, nel caso in cui il progetto sia correttamente eseguito.

La seconda parte è rappresenta dall'elemento “nodes”, in cui sono contenute, per ogni nodo, tutte le informazioni necessarie, come l'identificativo e la posizione nell'area di lavoro (Fig. 25).

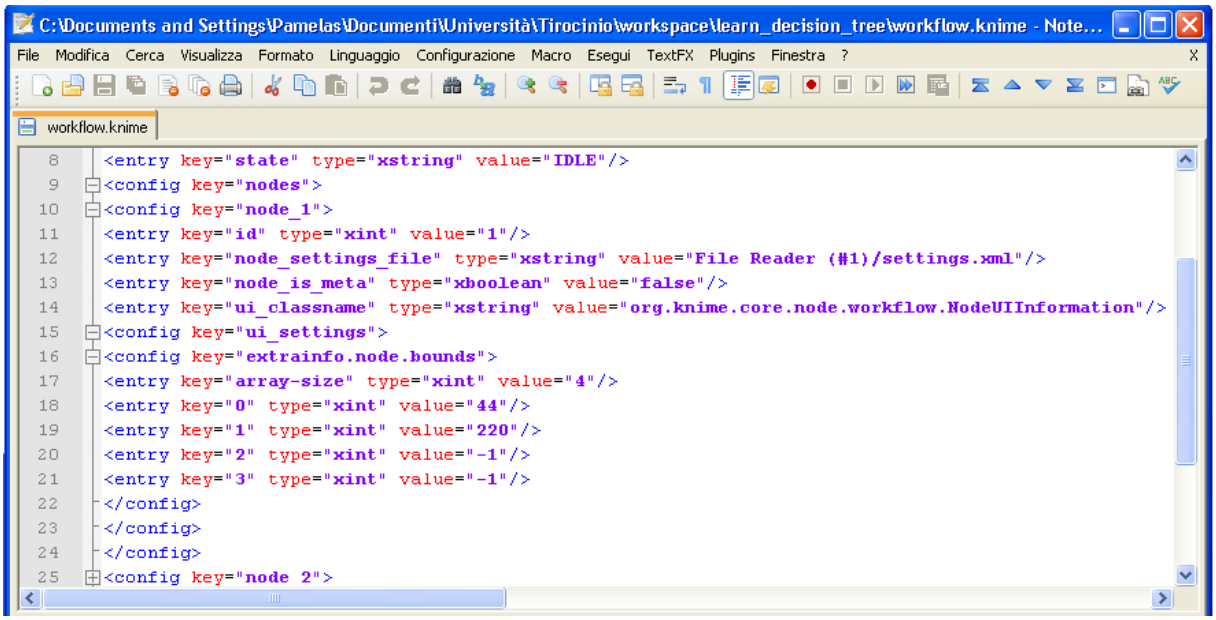

Fig. 25: configurazione dei nodi

La terza parte è rappresentata dall'elemento “connections”, in cui sono contenute le informazioni riguardanti le connessioni tra i nodi, come l'identificativo della connessione, identificativo e numero di porta del nodo di origine e identificativo e numero di porta del nodo destinatario (Fig. 26).

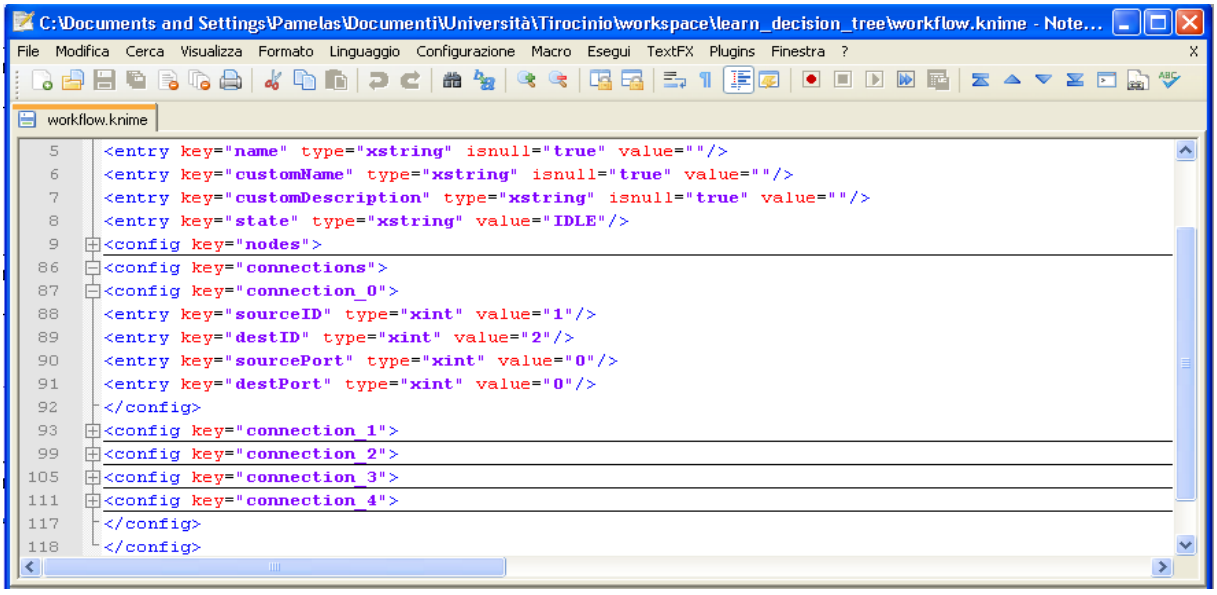

Fig. 26: configurazione delle connessioni

3. Il meta-nodo

Il meta-nodo è rappresentato internamente da una cartella, denominata secondo la sintassi dei nodi di progetto. La sua struttura interna è quella caratteristica di un progetto:

- contiene i file “workflow.knime”, “.project” e “.savedWithData”
- contiene una cartella per ogni suo nodo.

KNIME offre all'utente la possibilità di utilizzare una serie di metanodi predefiniti, inoltre dà la possibilità all'utente di crearne di suoi, mediante un tool fornito con il software di base e facilmente accessibile all'utente mediante un'icona sulla barra degli strumenti.

4. Il nodo

La struttura interna del nodo è caratterizzata da file di nodo e cartelle di nodo contenuti nella cartella del nodo (Fig. 27).

Fig. 27: file e cartelle di un nodo

I file utilizzati sono:

- “node”, un documento XML contenente le informazioni relative al nodo;
- “settings”, un documento XML contenente le informazioni relative allo stato del nodo, alla gestione della memoria ed altre informazioni del nodo.

Il documento “node” è logicamente diviso in due parti (Fig. 28): una parte contenente la configurazione dei parametri e una parte contenente le configurazioni delle uscite.

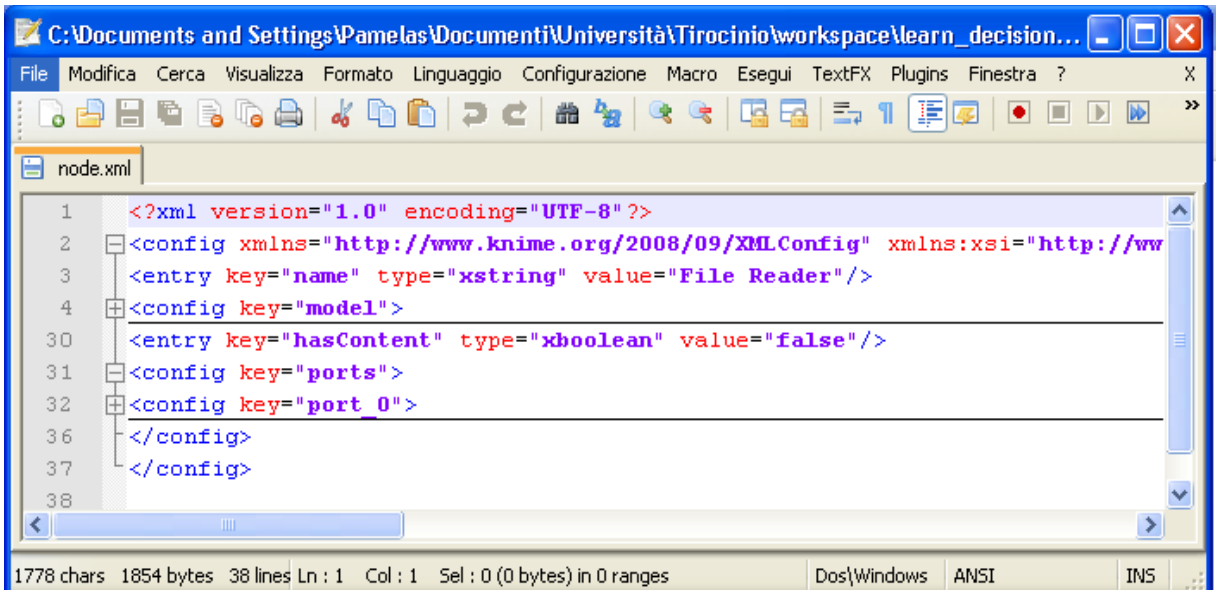

Fig. 28: struttura interna del documento "node"

La configurazione dei parametri è gestita nell'elemento “model” contenente tutti i parametri settabili del nodo (Fig. 29). Il contenuto di questo elemento cambia a seconda della tipologia di nodo. I parametri non impostati esplicitamente in configurazione assumono valori di default, tranne per parametri per cui sia specificato obbligatorio il settaggio da parte dell'utente.

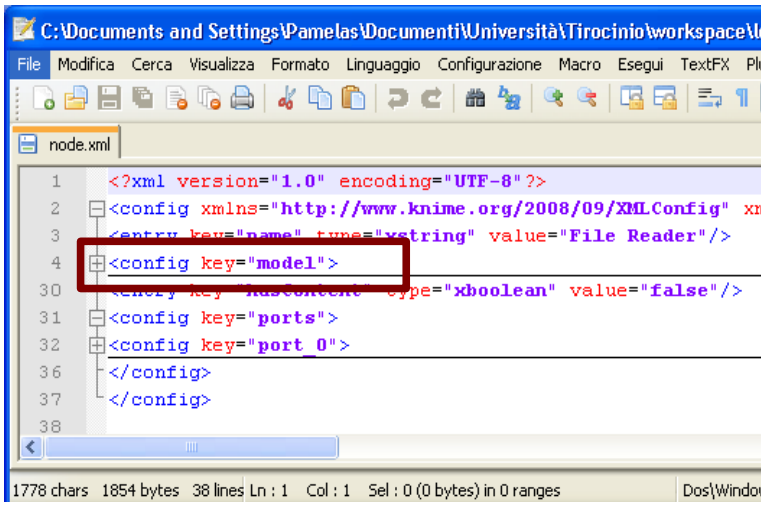

Fig. 29: La configurazione dei parametri

La configurazione delle uscite è caratterizzata da una flag, che indica la presenza di dati in uscita, e dall'elemento “ports” (Fig. 30). Nell'elemento ports sono indicate le informazioni e le locazioni dei dati in uscita sulle eventuali porte, impostate dopo l'esecuzione del nodo.

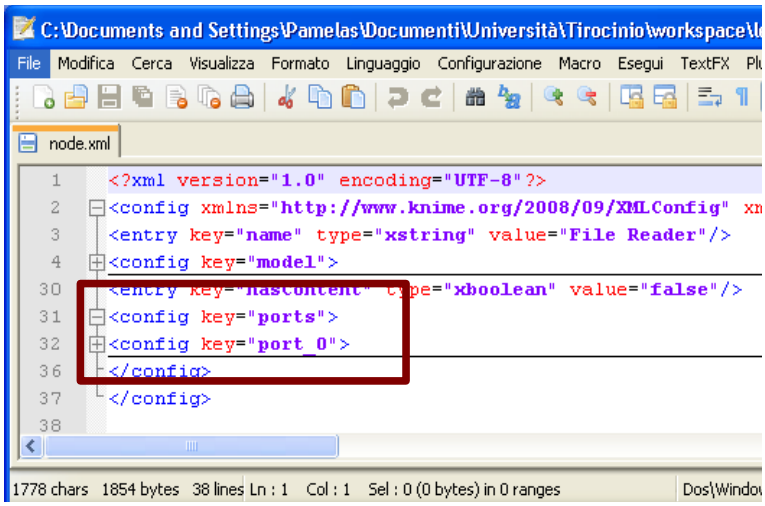

Fig. 30: configurazione delle porte di uscita

Le cartelle di un nodo sono prodotte dopo la sua esecuzione e sono:

- “internal”, contenente dati prodotti dal nodo per uso interno;

- “port_<numero_di_porta>”, prodotte una per ogni porta di output, contengono i file di output del nodo rappresentate nel formato interno.

. La comunicazione con il kernel KNIME

Per poterci interfacciare con il kernel di KNIME in DAME è necessario analizzare come questa comunicazione possa avvenire. La comunicazione con il kernel di KNIME è gestita mediante l'interfaccia grafica oppure mediante commandline.

1. La GUI

La GUI KNIME semplice da usare e intuitiva è costituita da:

1. il banco di lavoro su cui comporre la pipeline;
2. la repository delle funzionalità disponibili;
3. l'area dei progetti in lavorazione;
4. l'area di descrizione delle funzionalità, in cui interattivamente sono mostrate le caratteristiche della funzionalità selezionata.

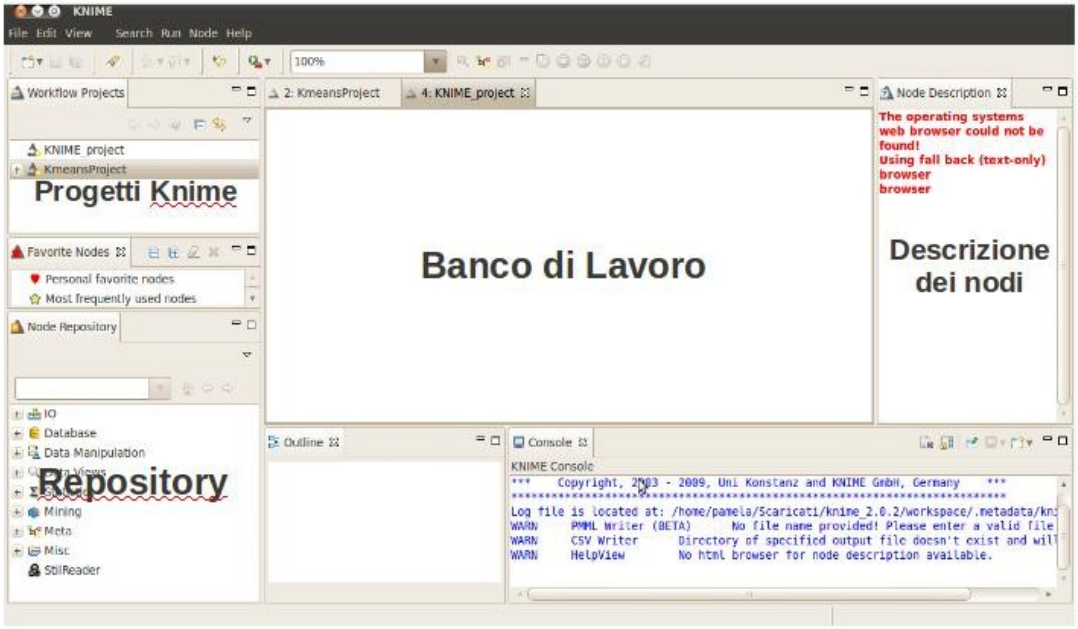

Fig. 31: interfaccia visuale di Knime

L'interfaccia interattiva drag and drop permette di comporre il flusso di analisi cliccando sull'icona della funzionalità e trascinandola sul banco di lavoro, allo stesso modo si possono connettere le funzionalità. Le operazioni di configurazione ed esecuzione della funzionalità sono possibili in modo altrettanto semplice: facendo click sull'icona del nodo con il tasto destro del mouse è mostrato un menù a tendina con tutte le operazioni possibili (Fig. 32).

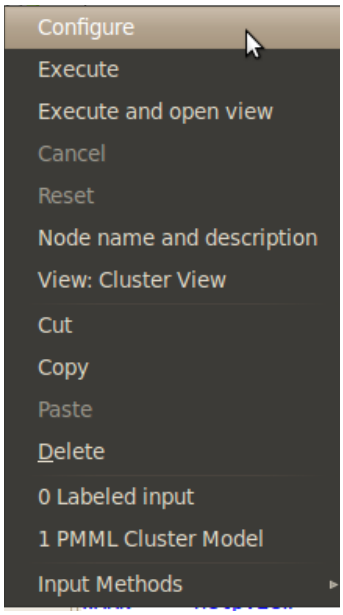

Fig. 32: menù delle operazioni sul nodo

2. La Commandline

La comunicazione da commandline permette di creare, configurare ed eseguire un progetto KNIME attraverso la composizione di un'unica linea di comando. La creazione del progetto è possibile solo attraverso un modello di progetto KNIME (Par. 8). Per eseguire la linea di comando bisogna posizionarsi nella cartella dove è localizzato l'eseguibile di KNIME.

Per la creazione ed esecuzione di un progetto KNIME già configurato basta eseguire la linea di comando seguente:

```
knime -nosplash -application org.knime.product.KNIME_BATCH_APPLICATION
-workflowFile = "<path_progetto>" -destFile = "<directory>"
```

dove *path_progetto* è il percorso del modello di progetto KNIME e *directory* è la directory in cui verrà creato il nuovo progetto.

Per eseguire un progetto già configurato basta eseguire la linea di comando seguente:

```
knime -nosplash -application org.knime.product.KNIME_BATCH_APPLICATION
-workflowDir = "<path_progetto>"
```

dove *path_progetto* è il percorso del progetto.

In entrambe i casi, esecuzione o creazione di un progetto, è possibile modificare i parametri di configurazione aggiungendo al termine della linea di comando l'opzione seguente:

```
-option="numero_nodo","parametro","valore","tipo"
```

dove *numero_nodo* è il numero del nodo nel workflow di cui modificare il parametro, *parametro* è il nome del parametro da modificare nel nodo, *valore* è il valore da assegnare al parametro e *tipo* è il tipo del parametro. Il tipo può essere uno qualsiasi dei tipi primitivi Java: String, ofStringCell, DoubleCell o IntCell. Nel caso in cui si voglia modificare un parametro nidificato, il nome va specificato scrivendo l'intero percorso separando gli elementi con il carattere "/". Nel caso in cui il nodo è parte di un meta-nodo bisogna fornire anche l'ID di nodo del nodo padre (s),ad esempio 90/56.

La linea di comando può essere utilizzata anche con altre opzioni, come:

- nosave non salvare il flusso di lavoro dopo che ha terminato l'esecuzione;
- reset ripristinare il flusso di lavoro prima dell'esecuzione;
- masterkey per richiedere la password master (utilizzati per esempio i nodi di database);

Le istruzioni sopra elencate sono per il sistema operativo Linux, per Windows bisogna aggiungere le opzioni ”`-noexit`“ e ”`-consoleLog`”. Ad esempio per l'esecuzione di un progetto già configurato si esegue:

```
knime -nosplash -application -noexit -consoleLog
org.knime.product.KNIME_BATCH_APPLICATION -workflowDir = “<path_progetto>”
```

. Le funzionalità

KNIME offre una grande varietà di nodi, attraverso i quali fornisce una grande quantità di operazioni sui dati tra cui operazioni di I/O, manipolazione, trasformazione, data mining e learning machine e una serie di componenti di visualizzazione. La maggior parte di essi è stata progettata per una stretta integrazione con framework, altri nodi sono dei wrapper che integrano funzionalità da librerie di terze parti.

1. La modalità avanzata

KNIME prevede una modalità avanzata per un'utenza specializzata, in cui sono previste ulteriori funzionalità.

Per passare alla modalità avanzata bisogna inserire la stringa “-Dknime.expert.mode=true” nel file di configurazione dell'applicazione, in particolare:

- per la versione Desktop il file è “knime.ini”;
- per la versione SDK il file è “eclipse.ini”.

2. I/O

Le operazioni di I/O si occupano della lettura e scrittura dei dataset scelti per l'elaborazione.

2.1. I nodi di lettura

I nodi di lettura, detti anche nodi di Read, si occupano della lettura dei dataset nei diversi formati trattati e sono (Fig. 33):

1. il nodo “File Reader” per la lettura di un file ASCII;
2. il nodo “ARFF Reader” per la lettura di un file in formato Attribute-Relation File Format (ARFF);
3. il nodo “Table Reader” per la lettura delle tabelle nel formato Table definito da KNIME (Par. 2.1);
4. il nodo “PMML Reader” per la lettura di un modello in formato Predictive Model Markup Language (PMML);
5. il nodo “Model Reader” per la lettura di un modello in un formato interno a KNIME.

Fig. 33: rappresentazione dei nodi di Read

Il nodo “Model Reader”, come il “Table Reader” per i dati, rappresenta un modello in un formato interno che può essere scritto solo attraverso il nodo di scrittura “Model Writer” (vedi Par. 5.2.2).

Il PMML utilizza l'XML per rappresentare modelli di data mining, la struttura dei modelli è

descritta da un XML Schema. Uno o più modelli di data mining possono essere contenuti in un PMML. KNIME tratta il formato PMML nella versione 3.1.

Essendo di interesse per la mia applicazione verranno forniti maggiori dettagli nel paragrafo 7.

2.2. I nodi di scrittura

I nodi di scrittura, detti anche nodi di Write, sono tutti i nodi per la scrittura dei risultati nei diversi formati. I nodi di scrittura sono (Fig. 34):

1. il nodo “CSV Writer” per la scrittura di un file in formato CSV;
2. il nodo “ARFF Writer” per la scrittura di un file in formato ARFF;
3. il nodo “Table Writer” per la scrittura delle tabelle;
4. il nodo “PMML Writer” per la scrittura di un modello in formato PMML;
5. il nodo “Model Writer” per la scrittura di un modello in un formato interno.

Fig. 34: rappresentazione dei nodi di Write

Anche per i nodi di scrittura verranno forniti maggiori dettagli nel paragrafo 7.

3. Manipolazione dei dati

Per la manipolazione dei dati KNIME fornisce nodi per il filtraggio di righe e colonne, partizionamento, campionamento, ordinamento, mescolamento casuale, unione e fusione dei dati. In particolare vedremo il nodo di partizionamento “Partitioning”.

3.1. Partitioning

Il nodo “Partitioning” manipola le righe di un dataset dividendolo in due dataset disgiunti. Ha una porta di input su cui riceve il dataset da partizionare e due porte di output (Fig.35), sulle quali sono riportate le due partizioni.

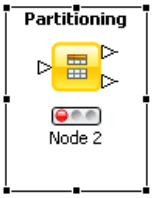

Fig. 35: rappresentazione del nodo Partitioning

I parametri di configurazione richiesti sono i seguenti (Fig. 36):

- la scelta tra “Absolute” / “Relative” per il tipo di partizionamento. Scegliendo “Absolute” viene effettuato il partizionamento assoluto, per cui bisogna specificare il numero di righe del dataset della prima partizione. Scegliendo “Relative” viene effettuato il partizionamento relativo, per cui bisogna specificare la percentuale di

righe del dataset per la prima partizione;

- la scelta tra “Take from top” / “Linear sampling” / “Draw randomly” / “Stratified sampling” per stabilire il criterio di selezione delle righe della prima partizione. Scegliendo “Take from top” le righe sono selezionate dal dataset in input a partire dalla prima riga. Scegliendo “Linear sampling” le righe sono selezionate dal dataset in input scegliendo in modo alternato la prima e l'ultima riga. Scegliendo “Draw randomly” le righe sono selezionate dal dataset in input scegliendole in modo casuale. Scegliendo “Stratified sampling” le righe sono selezionate dal dataset in input scegliendole in modo stratificato, rispettando per ogni classe la percentuale di righe del dataset di input (es: dataset di origine 60% uomini e 40% donne, la prima partizione 60% uomini e 40% donne).

- Scegliendo “Draw randomly” o “Stratified sampling” è possibile scegliere un seme per il partizionamento.

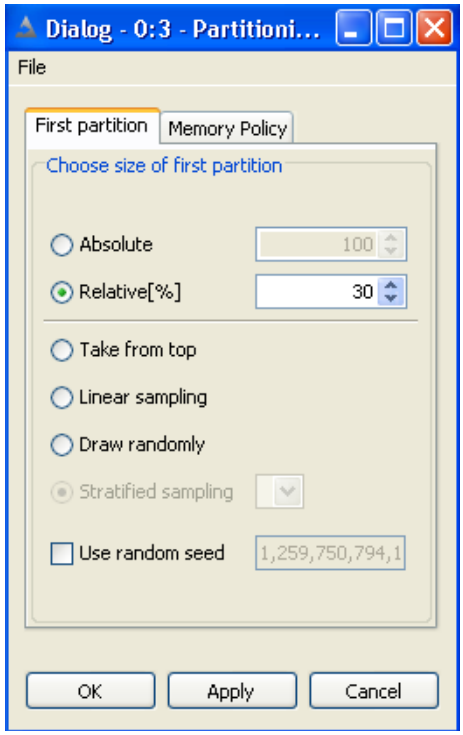

Fig. 36: finestra di dialogo per il nodo partitioning

4. Mining

Per il data mining sui dati, KNIME fornisce algoritmi di clustering, decision tree, regressione, reti neurali (probabilistici e reti neurali multi-strato) ed altro ancora. Abbiamo studiato alcuni nodi interessanti per il clustering e le reti neurali. Per le reti neurali abbiamo approfondito i nodi “MLP Learner” e “MLP Predictor” come base di confronto con l'omologo modello già implementato nella Suite DAME, mentre per il clustering il nodo “Kmeans”.

4.1. MLP Learner

Il nodo “MLP Learner” si occupa dell'apprendimento su un campione di dati, per definirne la classe di appartenenza. Implementa l'algoritmo RProp per le reti feedforward multistrato, il quale esegue un adattamento locale del peso in base al comportamento della funzione di errore. Il nodo fornisce anche una visione dell'andamento dell'errore. Ha una porta in input per ricevere il dataset su cui operare l'addestramento. Ha una porta in output su cui riportare i risultati ottenuti (Fig. 37).

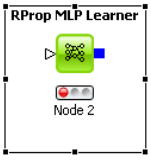

Fig. 37: rappresentazione del
nodo MLP Learner

I parametri richiesti sono (Fig. 38):

- il numero di iterazioni di apprendimento;
- il numero di strati nascosti nell'architettura della rete neurale;
- il numero di neuroni contenuti in ogni strato nascosto;
- la colonna del dataset contenente la variabile di classificazione. Essa può essere testuale o numerica;
- indica se vuoi utilizzare o meno le righe con valori omessi.

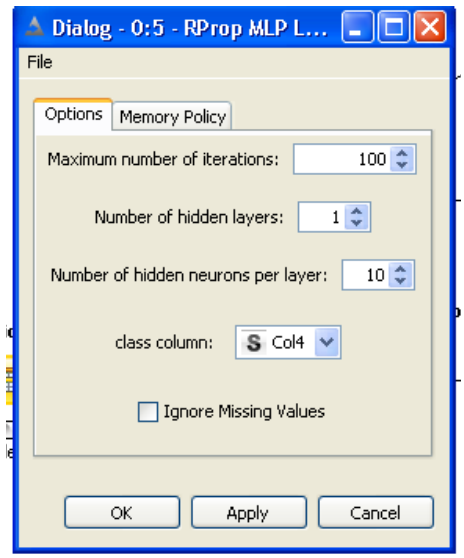

Fig. 38: finestra di dialogo per il nodo MLP Learner

4.2. MLP Predictor

Il nodo “MLP Predictor” effettua previsioni della classe di appartenenza su un insieme di dati non classificati. La previsione è effettuata in base ad un modello MultiLayerPerceptron ottenuto dalla porta di input per i modelli, in base al quale vengono fatte le previsioni sul dataset ottenuto dall'altra porta di ingresso. Il nodo non richiede impostazione di parametri ed

ha due porte di input: una porta dati per ricevere i dati di cui fare la previsione e una porta modello per ricevere il modello secondo cui fare la previsione; e una porta di output su cui vengono riportati i risultati (Fig. 39).

Fig. 39: rappresentazione del nodo MLP Predictor

4.3. Kmeans

Il nodo “Kmeans” fornisce i centroidi (centri di cluster) per un numero predefinito di cluster. L'algoritmo utilizzato assegna un vettore di dati ad un cluster e suddivide il vettore in uno specificato numero di sottoinsiemi denominati centroidi e definiti attraverso il calcolo della distanza tra gli elementi. L'algoritmo termina quando le assegnazioni di cluster non cambiano più. Ha una porta di input per il dataset su cui effettuare l'operazione e due porte di output: una per il dataset di output con le etichette del cluster di appartenenza ed una per il modello di cluster PMML (Fig. 40).

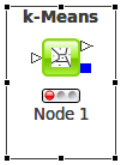

Fig. 40: Il nodo K-means

I parametri di configurazione del nodo sono:

- il numero di cluster che devono essere creati;
- il numero massimo di iterazioni prima che l'algoritmo termini;

5. Loop

I nodi di loop non sono immediatamente disponibili, poiché sono ritenuti come appartenenti ad una modalità avanzata. Per poterli utilizzare è necessario un particolare procedimento descritto in precedenza (Par. 5.1).

I nodi di loop sono nodi per svolgere operazioni cicliche e si suddividono in nodi di inizio e fine loop. I nodi di inizio sono posti prima del corpo del loop e sono i seguenti:

1. il nodo “Counting Loop Start” esegue il corpo del loop un numero predefinito di volte, necessita di un nodo “LoopEnd” per la raccolta dei risultati;
2. il nodo “Generic Loop Start” esegue il corpo del loop fino al verificarsi di una certa condizione, necessita di un nodo “Variable Condition Loop (End)” per delimitare i nodi appartenenti al corpo del ciclo;
3. il nodo “TableRow To Variable Loop Start” riceve in ingresso una tabella contenente l'insieme dei valori da utilizzare per l'iterazione, il nome della variabile è definito dal nome dato alla colonna nella tabella;
4. il nodo “Interval Loop Start” esegue il loop per un intervallo predefinito di valori, necessita di un nodo “LoopEnd” per la raccolta dei risultati.

I nodi di fine loop delimitano la fine del corpo del loop e sono:

1. il nodo “Loop End” raccoglie i risultati di un loop eseguito un numero predefinito di volte, necessita di un nodo di inizio loop in cui definire il criterio di terminazione;
2. il nodo “Variable Condition Loop End” raccoglie i risultati di un loop eseguito fino al verificarsi di una determinata condizione. Il nodo consente di impostare la condizione di fine loop, necessita di un nodo “Variable Condition Loop (Start)” per delimitare i nodi appartenenti al corpo del ciclo.

6. Meta-nodi

I meta-nodi predefiniti messi a disposizione da KNIME sono:

1. “Loop x times”;
2. “Iterate list of files”;
3. “X-Validation”.

6.1. Loop x times

Il meta-nodo “Loop x times” esegue un determinato flusso di lavoro più volte. “Loop x times” contiene i seguenti nodi:

- “Counting Loop Start” come nodo di inizio ciclo;
- “Loop End” come nodo di fine ciclo;
- altri nodi inseriti dall'utente.

Richiede la configurazione del numero di iterazioni da effettuare (Counting Loop Start).

6.2. Iterate list of files

Il meta-nodo “Iterate list of files” esegue iterativamente il contenuto di un sub-workflow per una lista di file. La lista di files ha bisogno di essere definita da una tabella di ingresso in cui ogni riga rappresenta un URL di un file. I nodi contenuti sono:

- “Variable Based File Reader” per la lettura di un file ASCII da una locazione URL;
- “Row To Variable Loop Start” per lo scorrimento di una tabella dove ogni riga contiene i valori delle impostazioni per una iterazione;
- “Loop End” per delimitare la fine di un ciclo.

6.3. X-Validation

Il metanodo “X-Validation” fornisce uno scheletro per il cross-validation, contiene i nodi:

- X-Aggregator per aggregare il risultato per la cross-validation;
- X-Partitioner per partizionare i dati per l'utilizzo in un flusso di cross-validation.

. Estensione di KNIME

L'estensione di KNIME attraverso nuovi nodi avviene come per tutti i nodi già predefiniti in KNIME, attraverso l'implementazione di un plugin Eclipse.

1. La struttura di un plugin Eclipse

In Eclipse tutto è un plug-in! Alla base c'è solo un piccolo motore di runtime per l'esecuzione dei plug-in e per determinarne le dipendenze. La struttura alla base di un plug-in comprende:

- un file “plugin.xml” contenente le dipendenze da altri plug-in ed i punti di estensione a cui si vuole connettere;
- un “Bundle Activator” che si occupa di registrare la condizione di “MANIFEST.MF”, file della directory META-INF in cui sono contenute informazioni sul percorso della classe, i plug-in necessari, il fornitore ed altro;
- un file “build.properties” in cui è definito il nome del file JAR in cui memorizzare le classi, i file da includere nella distribuzione ed altro.

2. La struttura di un plugin KNIME Node-Extension

Un plug-in per l'estensione delle funzionalità KNIME è un plug-in Eclipse. Il plug-in prevede inoltre l'implementazione delle seguenti classi:

- una classe che estende la classe astratta NodeModel responsabile dei passaggi per lo svolgimento della funzionalità. Essa richiede di sovrascrivere tre metodi principali:

    - il metodo configure() che prende le meta informazioni delle tabelle di input e definisce le specifiche di output;
    - il metodo execute() che crea effettivamente i dati di output o modelli;
    - il metodo reset() che ripristina tutti i risultati intermedi;
- una classe che estende la classe astratta NodeDialog, utilizzata per specificare la finestra di dialogo in cui sono impostati i parametri del nodo che influenzano la sua l'esecuzione;
- una classe che estende la classe astratta NodeView, utilizzata per creare viste virtuali sui dati, che può essere estesa più volte per consentire più viste sul modello sottostante.
- una classe NodeFactory per creare nuove istanze delle classi viste sopra.

Nella versione SDK di KNIME è disponibile un wizard (KNIME node-extension), che consente di generare tutte le classi richieste per la creazione di un nuovo nodo (Par. 6.3).

3. Il wizard KNIME Node-Extension

La struttura del plugin viene creata automaticamente tramite wizard, nel modo seguente:

1. Creazione del plugin per il nodo(Fig. 41 e Fig. 42);

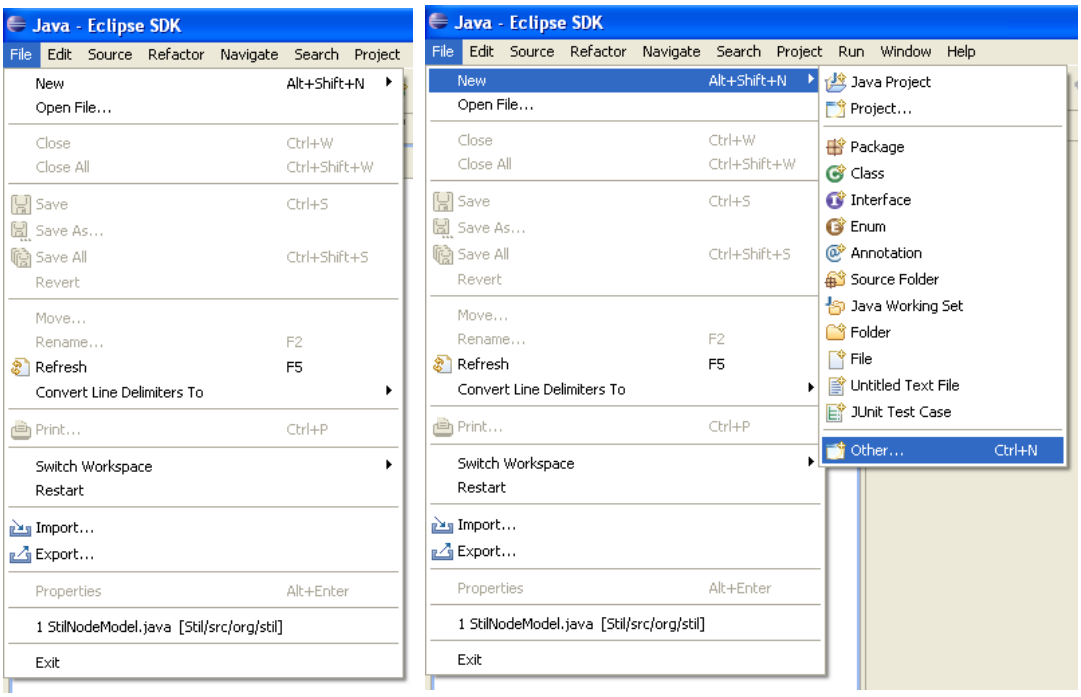

Fig. 41: creare un nuovo nodo plugin

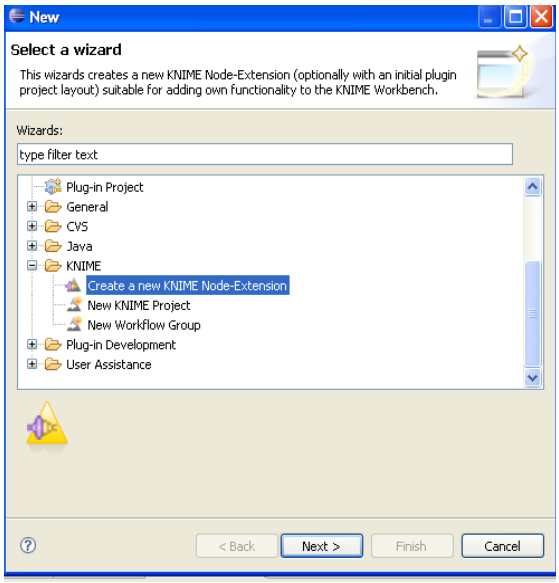

Fig. 42: selezione del wizard

2. Configurazione del plugin con le informazioni relative al nodo: nome del nodo, il nome della classe, nome del package e descrizione della funzionalità (Fig. 43).

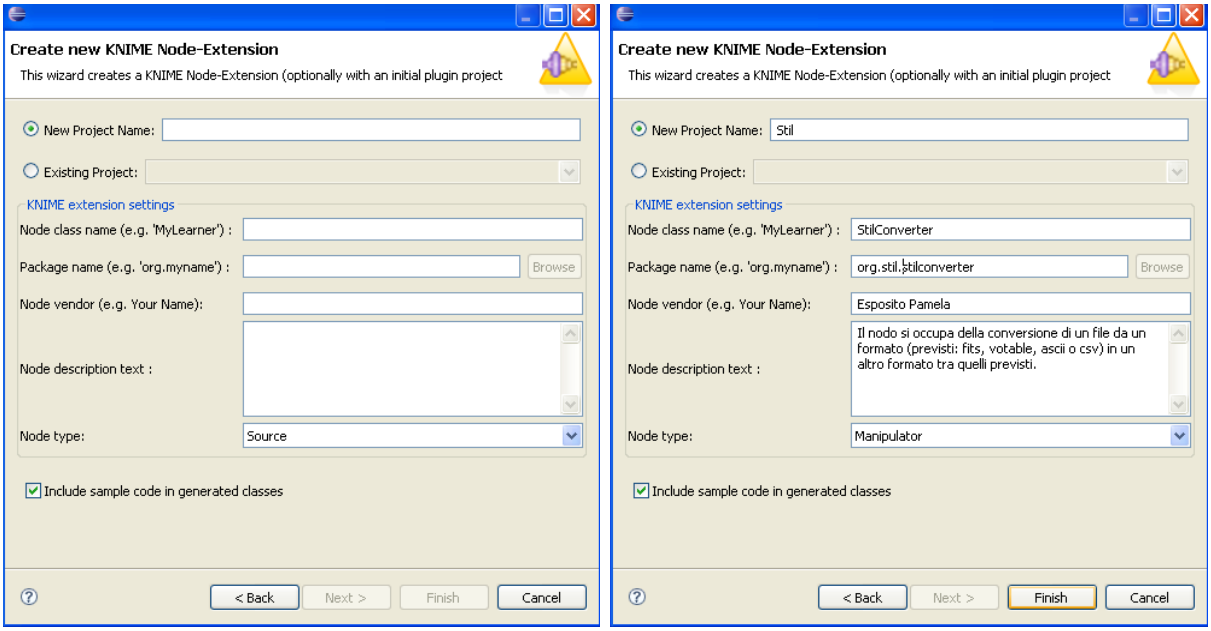

Fig. 43: Configurazione del plugin

4. Integrazione nella repository

L'esportazione e la distribuzione del nodo come plug-in può essere realizzata attraverso i seguenti passaggi:

- fai clic destro e scegli “Export”;
- nella finestra che appare selezionare l'opzione “Deployable plug-ins and fragments”;
- dopo aver cliccato il pulsante "Avanti" la procedura guidata visualizza un elenco con tutti i plug-in disponibili nella vostra area di lavoro;
- selezionare il plug-in da esportare;
- selezionare una directory in cui distribuire il vostro plug-in.

La directory conterrà il plug-in come archivio JAR. Il plug-in può essere installato in qualsiasi distribuzione di KNIME o di Eclipse, copiando il JAR nella cartella che contiene i plug-in KNIME base.

. I formati dei dati

Per poter valutare la compatibilità dei formati dei dati di input/output delle due piattaforme analizzeremo i formati trattati da KNIME. Per l'analisi è bastato approfondire la conoscenza dei nodi di Read e i nodi di Write (descritti nel paragrafo 5.2).

KNIME può leggere dataset nei formati ASCII, ARFF e Table Reader. Il nodo “File Reader” per la lettura di ASCII, legge correttamente il formato CSV, mentre i formati FITS e VOTABLE non sono letti correttamente.

Per la scrittura in output dei dati, KNIME può scrivere dataset nei formati CSV, ARFF e TABLE.

Il nodo “Table Reader”, come anticipato, legge solo dataset nel formato interno table, la scrittura di dataset in questo particolare formato può avvenire solo attraverso il nodo di scrittura “Table Writer”.

Il formato ARFF è un formato generalmente utilizzato per la memorizzazione di dati in un database.

. Il modello di progetto KNIME

Un modello di progetto KNIME è un progetto KNIME creato e configurato attraverso la GUI e compresso in un archivio ZIP.

CAPITOLO 4

IL PROGETTO DAME-KAPPA

Il lavoro svolto da me nell'ambito del progetto DAME consiste in un'applicazione middleware denominata KNIME applications Assembly in DAME (DAME-KappA). L'applicazione integra la piattaforma KNIME nella Suite, senza apportare modifiche strutturali né all'architettura della Suite né all'architettura KNIME.

. Descrizione del problema

KNIME in versione freeware nasce come applicazione desktop, ossia una piattaforma di analisi dati basata. Quest'ultima è la differenza sostanziale rispetto a DAME, che viceversa nasce come piattaforma di calcolo distribuito per data mining. Da questo punto di vista dunque i due sistemi risultano perfettamente complementari. La possibilità di creare un middleware, in grado di renderli interoperabili, avrebbe proprio lo scopo di rafforzarne reciprocamente le possibilità di utilizzo da parte degli utenti. KNIME per esempio, non fornisce strumenti per il trattamento dei dati nei formati più utilizzati in campo astrofisico (FITS, VOTABLE). Mentre in DAME implementare le funzionalità di KNIME comporterebbe tempi di sviluppo e risorse umane elevate essendo alcuni algoritmi anche

molto complessi. Per questo motivo, la soluzione consiste nell'ereditare in DAME le funzionalità di KNIME compatibilmente alle caratteristiche progettuali della Suite, risolvendo l'incompatibilità dei formati di I/O nelle due piattaforme.

. Valutazione delle possibili soluzioni

Le soluzioni esaminate avevano come scopo primario trovare una soluzione efficiente e in accordo con le scelte politiche della Suite[12]. E' importante precisare che l'intenzione non è duplicare KNIME in DAME, bensì analizzare la possibilità di ottenere un sistema ibrido potendo quindi utilizzare in modo ambivalente tutti i modelli di entrambe le applicazioni e ottenendo quindi il miglior grado di interoperabilità fra le due piattaforme, pur mantenendo integre le peculiarità di entrambe.

La prima soluzione valuta la possibilità di affiancare all'attuale componente Framework della Suite un nuovo componente per la gestione dell'esecuzione di esperimenti attraverso il kernel KNIME. La soluzione implica una variazione nell'architettura della Suite, oltre all'integrazione degli attuali componenti con funzionalità per la comunicazione con il nuovo componente. Esiste tuttavia un'alternativa all'adattamento del architettura della Suite per l'uso integrato di KNIME. Infatti, la seconda soluzione presa in considerazione utilizza la possibilità in DAME di integrare strumenti e algoritmi esterni, mediante il DMPlugin. L'idea di fondo di questa seconda soluzione proposta consiste nel creare off-line dei workflow completi con KNIME, per poi ereditarne il package di configurazione generato, per associarlo e renderlo disponibile in DAME, attraverso la procedura classica che fa uso del sub-componente DMPlugin. Questa soluzione è stata valutata come la scelta migliore, in quanto

12 Vedi RIF. 9

non apporta modifiche all'architettura della Suite e non comporta modifiche ai componenti già esistenti.

. Descrizione della soluzione

La soluzione scelta per l'implementazione realizza l'integrazione attraverso una procedura che si può sintetizzare nei seguenti passi:

1. La creazione di un modello di progetto KNIME per lo svolgimento della funzionalità;
2. La creazione di un plug-in DAME per ogni funzionalità;
3. La creazione di un modello DMM per l'esecuzione del plug-in.

Una volta individuati i nodi considerati interessanti per l'utenza di DAME, vengono composti i workflow e creati i relativi modelli di progetto KNIME (vedi Cap. 3 Par. 8).

L'integrazione in DAME della funzionalità avviene attraverso la creazione di un DMPlugin. Inoltre, poiché la funzionalità viene eseguita direttamente dal kernel KNIME, la comunicazione tra le due piattaforme deve essere implementata proprio nel plug-in. Infatti i DMPlugin per l'esecuzione sotto KNIME utilizzeranno un particolare modello che si occupa dell'esecuzione, denominato KnimeModel e integrato nel componente DMM della Suite. Il modello usufruisce della caratteristica di KNIME che permette di comunicare attraverso commandline.

Per la risoluzione dell'incompatibilità dei formati di I/O tra le piattaforme, per permettere l'utilizzo della piattaforma KNIME con DAME per i formati astronomici, creeremo un nodo per la lettura ed un nodo per la scrittura dei formati FITS tabellari, VOTABLE, CSV e ASCII.

I nodi saranno creati sottoforma di plugin Eclipse, per la corretta integrazione in KNIME.

. L'architettura

La scelta dell'architettura di DAME-KappA è vincolata dalle esigenze della Suite DAME e dal formalismo imposto da KNIME. Per l'integrazione di una generica funzionalità DMPlugin l'architettura deve considerare i seguenti fattori[13]:

- infrastruttura della Suite;
- incompatibilità dei formati di I/O tra la Suite e KNIME;
- formalismo di KNIME per la definizione dei suoi componenti.

I principi su cui è stata progettata l'architettura del middleware sono:

- la possibilità in DAME di integrare strumenti e algoritmi esterni mediante il DMPlugin (Cap. 2, Par. 5);
- la possibilità di comunicare con il KNIME kernel mediante linea di comando (Cap. 3, Par. 4.2);
- la possibilità in KNIME di integrare la repository dei nodi mediante un plugin Eclipse (Cap. 3, Par. 6).

1. I componenti

Il software consiste di quattro prodotti:

13 Vedi RIF. 10

- un nodo KNIME per la lettura di formati astronomici, denominato StilReader;
- un nodo KNIME per la scrittura di formati astronomici, denominato StilWriter;
- un modello DMM per l'esecuzione di workflow KNIME in DAME, denominato KnimeModel;
- i DMPlugin per eseguire funzionalità Knime.

In particolare svilupperemo un DMPlugin per fare clustering attraverso l'algoritmo K-means, denominato K-means.

Naturalmente il DMPlugin K-means è stato implementato a titolo d'esempio dello sviluppo di plug-in generici, in grado di inglobare workflow KNIME.

I nodi StilReader e StilWriter risolvono il problema dell'incompatibilità dei formati di I/O. Sono utilizzati in KNIME per comporre workflow di esperimenti su dati astronomici, inglobati in DAME attraverso modelli di progetto KNIME (vedi Cap. 3 Par. 8).

La compatibilità dei nodi con i nodi originari della piattaforma è garantita dai requisiti ereditati da KNIME. In particolare, i nodi sono implementati in Java sotto forma di plug-in Eclipse, caratteristica ereditata attraverso il tool messo a disposizione da KNIME SDK (vedi Cap. 3, Par. 6.3), il quale genera in modo automatico la struttura del plug-in. I dati transitanti tra i nodi del workflow sono incapsulati in una particolare struttura definita da KNIME (descritta nel Cap. 3 Par. 2.1), caratteristica ereditata grazie alla creazione della suddetta struttura durante l'implementazione dei nodi.

L'algoritmo di StilReader utilizza la libreria STIL[14] per la conversione del dataset in input nel formato CSV, salvando i risultati in un file temporaneo utilizzato per riportare i dati nel formato interno.

14 Vedi RIF. 7

L'algoritmo di StilWriter utilizza la libreria STIL per la conversione del dataset ricevuto dalla porta in ingresso e salvato in un file temporaneo nel formato CSV, nel formato richiesto dall'utente. In tal modo, per poter effettuare un esperimento su dati astronomici, è sufficiente comporre i workflow inserendo i nostri nodi StilReader e StilWriter.

Il modello KnimeModel permette la comunicazione tra le due piattaforme per l'esecuzione dell'esperimento. E' utilizzato dai DMPlugin per eseguire il workflow KNIME per lo svolgimento della funzionalità. Il modello è scritto in linguaggio Java ed implementa l'interfaccia DMMInterface (Fig.44) per garantire la corretta integrazione nel DMM e il corretto utilizzo da parte dei DMPlugin. L'algoritmo è basato sulla peculiarità di KNIME che permette di eseguire un progetto KNIME da commandline. Esso compone la linea di comando in base ai parametri ricevuti e la esegue.

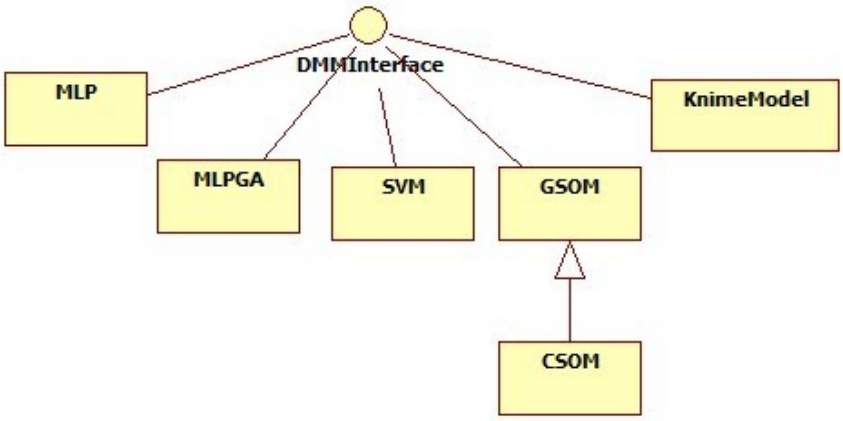

Fig. 44: DMM diagram class

Il DMPlugin KNIME integra un workflow nella Suite. E' utilizzato dal FW per condurre un esperimento eseguito attraverso il KNIME kernel. Il DMPlugin è implementato in Java. Per la corretta integrazione della funzionalità nella Suite, il plug-in implementa l'interfaccia comune a tutti i modelli integrati nel DMPlugin (AbstractDMPlugin). Inoltre implementa una serie di metodi ausiliari per l'impostazione dei parametri del plug-in e dei parametri da modificare nel workflow. In particolare implementeremo un plug-in per fare clustering attraverso l'algoritmo K-means.

2. La comunicazione

Per mostrare in che modo avviene la comunicazione tra i componenti di DAME-KAPPA per l'esecuzione di un esperimento eseguito sotto KNIME, ripercorreremo passo dopo passo il ciclo di vita di un DMPlugin che esegue un workflow KNIME.

2.1. Le ipotesi iniziali

Per l'utilizzo di un DMPlugin attraverso il KnimeModel si presuppone che tutti i workflow relativi al plug-in (per tutti i casi d'uso previsti dalla funzionalità) siano raccolti in modelli di progetto KNIME e memorizzati nella locazione predefinita (locazione dei ZIP).

2.2. L'interfaccia di KNIME in DAME per l'esperimento

L'utilizzo di KNIME in DAME è reso trasparente all'utente, tutte le operazioni di creazione, configurazione ed esecuzione di un esperimento sono effettuate attraverso la GUI DAME.

Il DMPlugin di una funzionalità eseguita sotto KNIME è attivato dalla richiesta da parte del FW di eseguire l'esperimento. Attraverso il parsing del file XML il FW stabilisce il plug-in richiesto, istanzia un oggetto della classe plug-in per l'esperimento scelto e configura l'esperimento attraverso il costruttore della classe, che imposta i parametri di configurazione dell'esperimento prelevati dal parsing dell'XML. Viene poi eseguito il metodo run del caso d'uso richiesto, il quale istanzia un oggetto del modello KnimeModel e, attraverso i metodi messi a disposizione del modello, setta tutti i parametri necessari all'utilizzo. Infine richiama il metodo del caso d'uso richiesto messo a disposizione dal modello. Il plug-in rappresenta quindi l'interfaccia di KNIME in DAME.

2.3. La comunicazione tra KNIME e DAME

Con la chiamata al caso d'uso del modello, il controllo dell'esecuzione passa all'istanza

della classe KnimeModel. Nel corpo del caso d'uso richiama il metodo createKnimeProject, che si occupa della creazione, configurazione ed esecuzione di un modello di progetto KNIME.

2.4. L'esecuzione

Il metodo createKnimeProject dopo aver eseguito opportuni controlli crea la linea di comando e la esegue attraverso il metodo “inviaComando”, che si occupa dell'esecuzione della commandline. Il metodo restituisce al metodo createKnimeProject un booleano che rappresenta l'esito dell'esecuzione. Il metodo createKnimeProject restituisce al caso d'uso la directory dove è stato memorizzato l'esperimento. I casi d'uso del DMM non ritornano alcun valore.

2.5. Interfaccia output di KNIME con DAME

Il controllo ritorna quindi al DMPlugin, che si occupa di registrare e mostrare il file di log, il file di errore e i file di output. I file risultanti ottenuti sono memorizzati in locale, nella cartella utente.

2.6. Monitoraggio dell'esecuzione

Le istruzioni eseguite da linea di comando dal kernel KNIME mostra lo stato d'avanzamento dell'esecuzione dell'esperimento in percentuale, attraverso le quali è possibile effettuare il monitoraggio dell'esperimento. Quindi redirezionando l'output della linea di

comando al file log si ottiene il monitoraggio dell'esperimento.

3. I dataset di I/O

Tutti i dati riguardanti gli esperimenti di un utente sono salvati in locale nella cartella utente. Il software crea una cartella per ogni utente in una locazione prestabilita. La cartella dell'utente sarà denominata con l'username del proprietario. Una generica cartella utente conterrà due cartelle una per la raccolta degli esperimenti KNIME (workspace) ed una per i file di output (FileOut).

Tutti i dataset di output e i file provvisori sono salvati in locale nella cartella denominata con il nome dell'esperimento nella cartella FileOut dell'utente. I file provvisori sono raccolti in una cartella denominata Temp in FileOut.

Tutti gli esperimenti sono raccolti nella cartella workspace della cartella utente. L'esperimento è denominato con il nome assegnato dall'utente all'esperimento.

. L'implementazione

Di seguito sono documentate tutte le fasi di implementazione per[15]:

1. il nodo StilReader;
2. il nodo StilWriter;
3. il modello KnimeModel;
4. il DMPlugin K-means;

15 Vedi RIF. 11

Inoltre nel paragrafo 5.4.1 sono descritti i requisiti che deve soddisfare un generico DMPlugin che utilizza il modello KnimeModel.

1. Il nodo StilReader

Il nodo StilReader è stato sviluppato secondo le caratteristiche definite da KNIME per la corretta integrazione nella repository. L'interfacciamento del nodo con l'utente avviene attraverso KNIME, come per gli altri nodi sarà possibile configurarne i parametri e eseguirlo. Il nodo richiede l'importazione della libreria Stil e le classi dameError (Fig. 4), dataset e datasetException (Fig. 3). In particolare, questa classe importa a sua volta una gerarchia di classi appartenenti al pacchetto del progetto DAME.

1.1. La creazione del plug-in StilReader

Il plug-in Eclipse è generato automaticamente dal wizard (descritto nel Cap. 3 Par. 6.3). Il plug-in è configurato come da figura (Fig. 45).

1.2. Classi e librerie importate

Le classi del plug-in nella loro implementazione utilizzano una serie di classi ereditate da DAME (Fig. 46 - 47) e la libreria Stil, opportunamente importate dal plug-in.

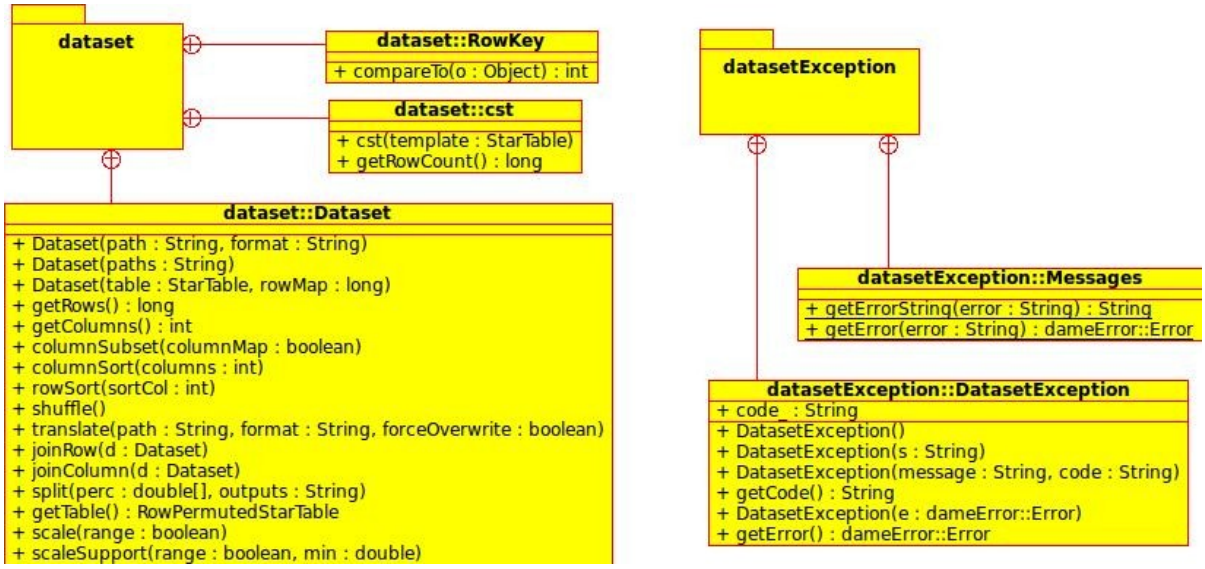

Fig. 46: Package "dataset" e "datasetException"

1.3. Implementazione delle classi

Una volta creata la struttura (Fig. 48) bisogna implementare le classi generate per fare in modo che eseguano la corretta lettura dei dataset in ingresso. Il nostro plug-in richiede l'implementazione delle classi StilReaderNodeDialog per impostare la finestra di dialogo e StilReaderNodeModel per lo svolgimento della funzionalità.

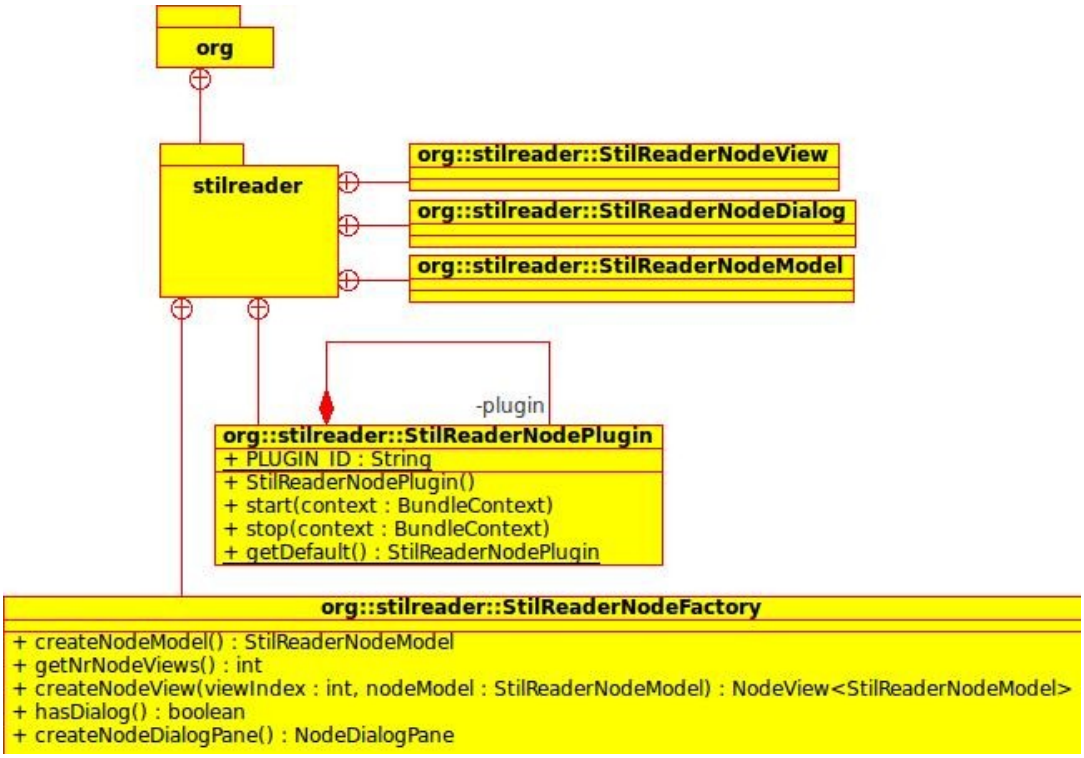

Fig. 48: Package StilReader

1.3.1 La classe StilReaderNodeModel

La classe StilReaderNodeModel importa una serie di classi di KNIME, la classe Dataset della Suite (Fig.3-4) e la libreria Stil. Definisce inoltre due chiavi di impostazione, utilizzate per recuperare e memorizzare le impostazioni dalla finestra di dialogo o da un file di impostazione. Esse hanno visibilità di pacchetto per renderle utilizzabili dalla finestra di dialogo.

```
static final String CFGKEY_IN_FILE = "inFile";
static final String CFGKEY_IN_FORMAT = "inFormat";
```

Utilizza inoltre due variabili di tipo SettingsModelString, riempite dalla finestra di dialogo e utilizzate nel metodo di esecuzione. I valori di default sono impostati mediante il costruttore "SettingsModels".

```
private final SettingsModelString m_inFile;
private final SettingsModelString m_inFormat;
```

Per l'URI del dataset è utilizzata la chiave di impostazione *CFGKEY_IN_FILE* e la variabile `m_inFile`. Per il formato del dataset è utilizzata la chiave di impostazione *CFGKEY_IN_FORMAT* e la variabile `m_inFormat`.

La classe contiene un costruttore e sette metodi. Il costruttore StilReaderNodeModel della classe non richiede parametri di ingresso e mediante una chiamata alla superclasse costruisce un nodo senza porte di input ed una porta di output.

Il metodo “execute” si occupa dello svolgimento della funzionalità del nodo, l'algoritmo utilizzato per poter portare a termine la lettura del dataset è il seguente:

- crea un oggetto di tipo Dataset denominato d, mediante la classe Dataset importata

dalla Suite, utilizzando il costruttore che prende in input l'URI ed il formato del dataset impostato dall'utente nella finestra di dialogo;

- mediante il metodo “translate” della classe Dataset della Suite converte il dataset creato al passo precedente in un file dataset in formato CSV;
- utilizzando i metodi della classe Stil ricaviamo il numero di righe ed il numero di colonne del dataset;
- dichiariamo un oggetto StarTable a cui assegniamo il dataset d;
- dichiariamo un oggetto ColumnInfo, che farà riferimento alla struttura contenente le informazioni relative ad una colonna ;
- definisco una struttura allColSpec di tipo DataColumnSpec, in cui il numero di colonne è specificato dal numero di elementi contenuti nella struttura (array unidimensionale) e il tipo di valori che conterranno specificandolo negli elementi dell'array;
- definisce i tipi delle colonne tramite un ciclo, che per ogni colonna del dataset ne estrae le informazioni, ne definisce la classe si appartenenza ed imposta nell'array il valore opportuno;
- definisce la tabella di output attraverso due cicli innestati, di cui il primo è eseguito sulle righe, il secondo sulle colonne. Il corpo del secondo ciclo contiene l'assegnazione al valore estratto dalla tabella.

I metodi “reset”, “loadInternals” e “saveInternals” non hanno corpo. Il metodo “configure” si occupa dei controlli sulle variabili lette dalla finestra di dialogo, in particolare controlla che il valore di `m_inFile` sia immesso e che il file esista.

Il metodo “validateSetting” si occupa del controllo dei valori attuali delle variabili, nel nostro caso i controlli sono delegati al metodo validateSettings della classe SettingsModels.

Il metodo “loadValidateSettingFrom” si occupa del caricamento delle informazioni nei campi locali. Quando viene invocato questo metodo le informazioni sono già validate dal metodo validateSetting. Nel nostro caso il caricamento è delegato al metodo loadSettingFrom della classe SettingsModels.

Il metodo “saveSettingTo” si occupa di scrivere i campi locali nelle impostazioni in modo che la finestra di dialogo visualizzi i valori correnti. Nel nostro caso la scrittura è delegata al metodo saveSettingTo della classe SettingsModels.

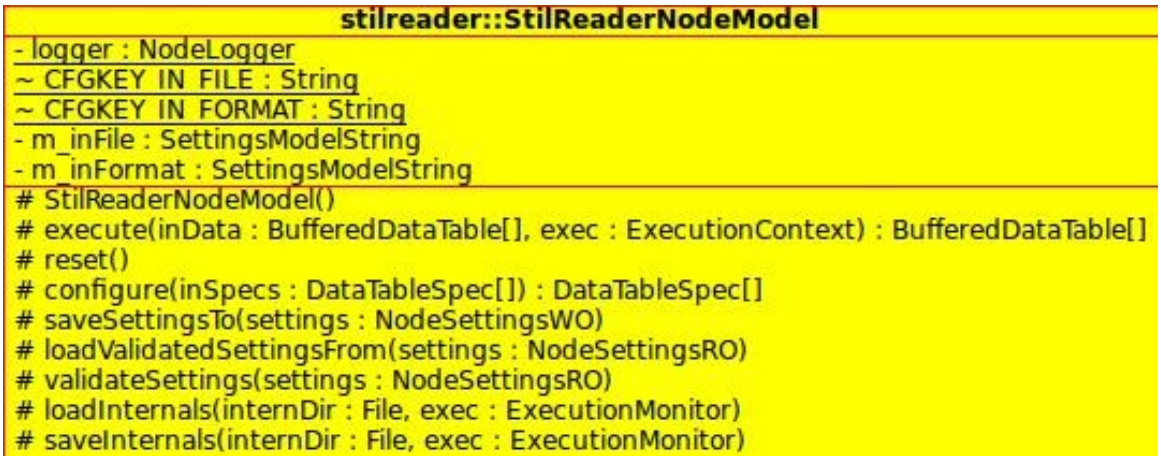

Fig. 49: StilReaderNodeModel class

1.3.2 La classe StilReaderNodeDialog

La classe StilReaderNodeDialog importa una serie di classi di KNIME. La classe contiene un costruttore, StilReaderNodeDialog, il quale non richiede parametri di ingresso e mediante una chiamata alla superclasse costruisce un oggetto finestra di dialogo. Esso definisce inoltre gli oggetti per l'acquisizione delle informazioni necessarie alla configurazione, quindi per l'URI del dataset ed il formato del Dataset.

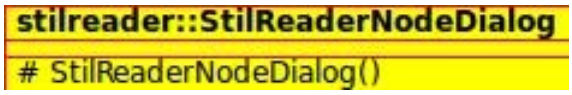

Fig. 50: StilReaderNodeDialog class

1.3.3 Le altre classi

Le classi StilReaderNodeFactory (Fig. 51), StilReaderNodeView (Fig. 52) e StilReaderNodePlugin (Fig. 53) non sono state modificate rispetto al modello generato dal wizard.

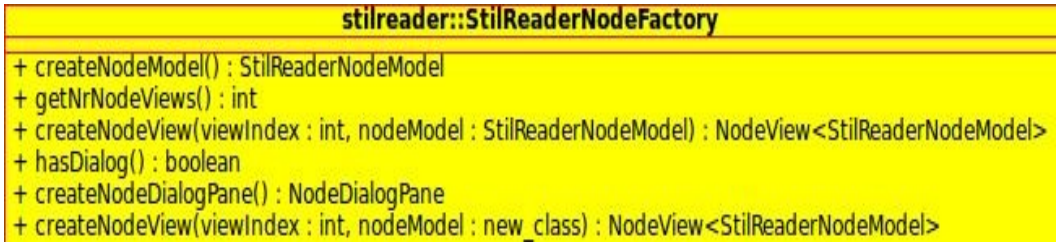

Fig. 51: StilReaderNodeFactory class

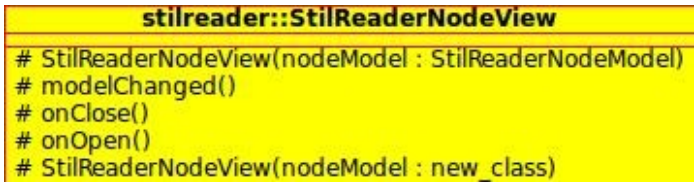

Fig. 53: StilReaderNodeView class

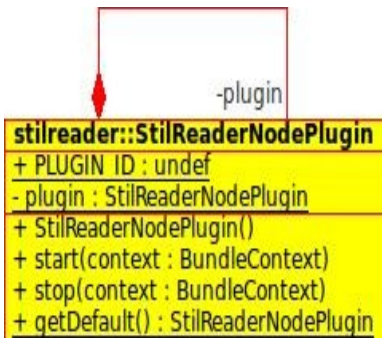

Fig. 52: StilReaderNodePlugin class

1.4. L'integrazione nella repository

Per l'integrazione del nuovo nodo bisogna esportare e distribuire il nodo attraverso il procedimento definito nel Cap. 3 Par. 6.4.

2. Il nodo StilWriter

Il nodo StilWriter è stato sviluppato secondo le caratteristiche definite da KNIME per la corretta integrazione nella repository. L'interfacciamento del nodo con l'utente avviene attraverso KNIME. Come per gli altri nodi, sarà possibile configurarne i parametri e eseguirlo. Il nodo richiede l'importazione della libreria Stil e le classi dameError (Fig. 47), dataset e datasetException (Fig. 46). Inoltre per il corretto funzionamento del nodo è stato necessario ereditare alcune classi di KNIME nel pacchetto. Queste classi sono originariamente utilizzate per lo sviluppo del nodo CSVWriter e con visibilità a livello di pacchetto, per questo motivo per poter implementare questo nodo è stato necessario recuperare queste classi ed adattarle alle nostre esigenze.

2.1. La creazione del plug-in StilWriter

La creazione del plug-in Eclipse è generato automaticamente dal wizard (descritto nel Cap. 3 Par. 6.3). Il plug

2.2. Classi e librerie importate

Le classi del plug-in nella loro implementazione utilizzano una serie di classi ereditate da DAME (Fig. 46 - 47) e la libreria Stil, opportunamente importate dal plug-in. Utilizza inoltre una classe FileWriterSettings (implementata da KNIME) contenente tutte le impostazioni utilizzate dallo scrittore e i metodi per la lettura e scrittura degli oggetti. Quest'ultimo è utilizzato nel NodeModel per passare le impostazioni per la scrittura del file.

2.3. Implementazione delle classi

Una volta creata la struttura bisogna implementare le classi generate per fare in modo che eseguano la corretta scrittura del dataset ricevuto in input. Il nostro plug-in richiede l'implementazione delle classi StilWriterNodeDialog per impostare la finestra di dialogo e StilWriterNodeModel per lo svolgimento della funzionalità.

2.3.1 La classe StilWriterNodeModel

La classe StilWriterNodeModel importa una serie di classi di KNIME, la classe Dataset della Suite (Fig. 46 - 47) e la libreria Stil. Definisce inoltre due chiavi di impostazione, utilizzate per recuperare e memorizzare le impostazioni dalla finestra di dialogo o da un file di impostazione. Esse hanno visibilità di pacchetto per renderle utilizzabili dalla finestra di dialogo.

```
static final String CFGKEY_OUT_FILE = "outFile";
static final String CFGKEY_OUT_FORMAT = "outFormat";
```

Inoltre, utilizza due variabili di tipo SettingsModelString riempite dalla finestra di dialogo e utilizzate nel metodo di esecuzione. I valori di default sono impostati mediante il costruttore "SettingsModels".

```
private final SettingsModelString m_outFile;
private final SettingsModelString m_outFormat;
```

Per l'URI del dataset è utilizzata la chiave di impostazione *CFGKEY_OUT_FILE* e la variabile `m_outFile`. Per il formato del dataset è utilizzata la chiave di impostazione *CFGKEY_OUT_FORMAT* e la variabile `m_outFormat`.

La classe contiene un costruttore e nove metodi. Il costruttore StilWriterNodeModel della classe non richiede parametri di ingresso e mediante una chiamata alla superclasse costruisce un nodo senza porte di output ed una porta di input.

Il metodo “execute” si occupa dello svolgimento della funzionalità del nodo. L'algoritmo utilizzato per poter portare a termine la scrittura del dataset è il seguente:

1. verifica l'esistenza della directory in cui creare il file. Nel caso in cui non esista la crea;
2. legge attraverso il dataset ricevuto dalla porta di ingresso e lo scrive in formato CSV;
3. crea un oggetto di tipo Dataset denominato d, mediante la classe Dataset importata dalla Suite, utilizzando il costruttore che prende in input l'URI impostato dall'utente nella finestra di dialogo. Il formato del dataset è impostato a CSV;
4. mediante il metodo “translate” della classe Dataset della Suite converte il dataset creato al passo precedente in un file dataset in formato richiesto dall'utente nella finestra di dialogo.

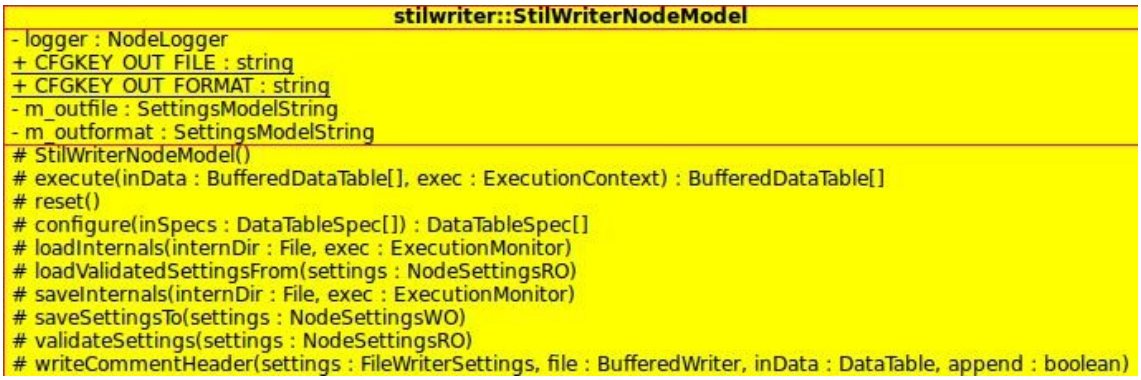

Fig. 55: StilReaderNodeModel class

2.3.2 La classe StilWriterNodeDialog

La classe StilWriterNodeDialog importa una serie di classi di KNIME. La classe contiene un costruttore, StilWriterNodeDialog, il quale non richiede parametri di ingresso e mediante una chiamata alla superclasse costruisce un oggetto finestra di dialogo. Esso definisce inoltre gli oggetti per l'acquisizione delle informazioni necessarie alla configurazione, quindi per

l'URI del dataset ed il formato del Dataset.

Fig. 56: StilReaderNodeDialog class

2.3.3 Le altre classi

Le classi StilWriterNodeFactory (Fig. 57), StilWriterNodeView (Fig. 58) e StilWriterNodePlugin (Fig. 59) non sono state modificate rispetto al modello generato dal wizard.

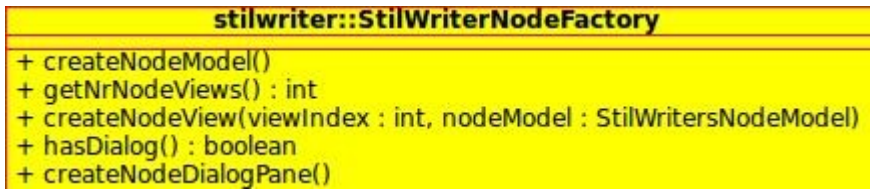

Fig. 57: StilReaderNodeFactory class

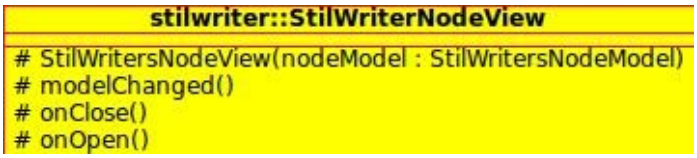

Fig. 59: StilReaderNodeView class

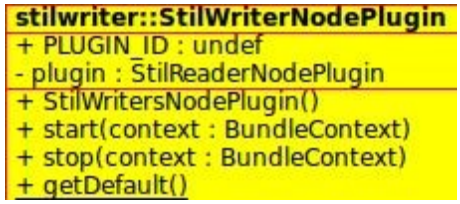

Fig. 58: StilReaderNodePlugin class

2.4. L'integrazione nella repository

Per l'integrazione del nuovo nodo bisogna esportare e distribuire il nodo attraverso il procedimento definito nel Cap. 3 Par. 6.4.

3. Il modello KnimeModel

Il modello incluso nel DMM creato per l'esecuzione tramite il KNIME Kernel e denominato KnimeModel (Fig. 60), gestisce l'esecuzione dei plug-in KNIME (KNIME plug-in). Il modello prevede l'utilizzo di:

- tre costanti;
- cinque variabili;
- nove metodi, oltre ai tre utilizzati per i casi d'uso.

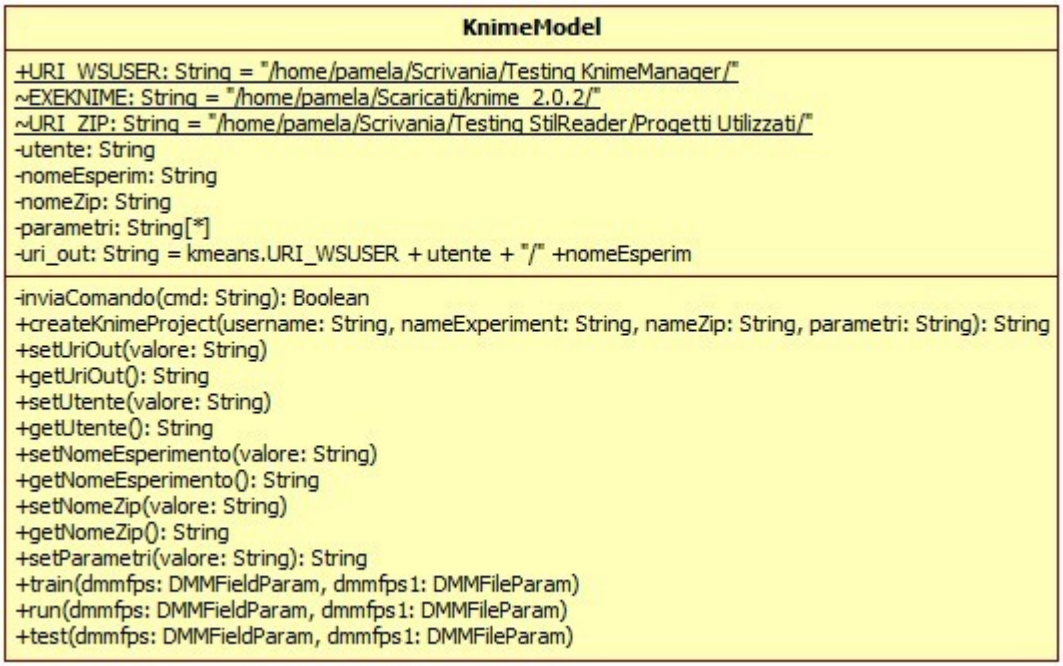

Fig. 60: La classe KnimeModel

3.1. Le costanti

Le costanti utilizzate sono:

- *URI_WSUSER* (URI WorkSpace USER) è una stringa che rappresenta l'URI della directory contenente di tutte le cartelle degli utenti;
- *EXEKNIME* (EXEcutable KNIME) è una stringa che rappresenta l'URI della directory contenente l'eseguibile KNIME;
- *URI_ZIP* (URI ZIP) è una stringa che rappresenta l'URI della directory contenente i modelli di progetto KNIME;

3.2. Le variabili

Le variabili utilizzate sono:

- *utente* è una stringa che rappresenta l'username dell'utente che svolge l'esperimento;
- *nomeEsperim* è una stringa che rappresenta il nome assegnato all'esperimento dall'utente;
- *nomeZip* è una stringa che rappresenta il nome del modello di progetto KNIME da utilizzare;
- *parametri* è un'array bidimensionale in cui sono memorizzati i parametri da configurare nel progetto KNIME;
- *uri_out* è una stringa che rappresenta la locazione in cui memorizzare i file di output;

3.3. I metodi ausiliari

Vedremo ora tutti i metodi ausiliari implementati per realizzare il modello per l'esecuzione di progetti KNIME attraverso il KNIME kernel.

- Il metodo “inviaComando” si occupa dell'esecuzione da terminale della linea di comando, passata come parametro al metodo. Ritorna un booleano che indica l'esito dell'operazione.
- Il metodo “createKNIMEProject” si occupa della creazione ed esecuzione di un modello di progetto KNIME. I parametri richiesti sono:
  - l'username dell'utente;
  - il nome assegnato all'esperimento;
  - il nome del modello di progetto KNIME;
  - i parametri da impostare.

  Per l'esecuzione della linea di comando richiama il metodo inviaComando e ritorna l'URI del progetto.
- Il metodo “setUriOut” si occupa di scrivere il valore della variabile uri_out;

- Il metodo “getUriOut” si occupa di leggere il valore della variabile uri_out;
- Il metodo “setUtente” si occupa di scrivere il valore della variabile utente;
- Il metodo “getUtente” si occupa di leggere il valore della variabile utente;
- Il metodo “setNomeEsperimento” si occupa di scrivere il valore della variabile nomeEsperim;
- Il metodo “getNomeEsperimento” si occupa di leggere il valore della variabile nomeEsperim;
- Il metodo “setNomeZip” si occupa di scrivere il valore della variabile nomeZip;
- Il metodo “getNomeZip ” si occupa di leggere il valore della variabile nomeZip.

3.4. I casi d'uso

Tutti i casi d'uso sono implementati in modo da richiamare il metodo “createKnimeProject”, che si occupa della creazione dell'esperimento.

4. Il DMPlugin per KNIME

4.1. Il generico DMPlugin di KNIME

Un generico DMPlugin per l'esecuzione di workflow KNIME implementa l'interfaccia “AbstractDMPlugin”, per cui deve implementare almeno tutti i casi d'uso. Inoltre per l'esecuzione di un progetto KNIME è necessario che il plug-in abbia i requisiti descritti nei seguenti sotto paragrafi.

4.1.1 Le costanti

Il DMPlugin utilizza due costanti:

- la costante “statusRunning” è una stringa utilizzata per indicare lo stato dell'esecuzione, in particolare indica la terminazione dell'esecuzione;
- la costante “statusError” è una stringa utilizzata per indicare lo stato di errore;

4.1.2 Le variabili

Il DMPlugin utilizza sei variabili:

- la variabile “inputField” è un array di variabili strutturate di tipo DMMFieldParam (una struttura per memorizzare le informazioni sui parametri), utilizzata per memorizzare i parametri del plug-in;
- la variabile “inputFiles” è un array di variabili strutturate di tipo DMMFileParam (una struttura per memorizzare le informazioni sui files di input), utilizzata per memorizzare i parametri del plug-in;
- la variabile “outputFiles” è un array di variabili strutturate di tipo DMMOutputFileParam (una struttura per memorizzare le informazioni sui files di input), utilizzata per memorizzare i parametri del plug-in;
- una variabile di tipo DMMOutputFileParam per ogni file di output;
- una variabile di tipo stringa per l'URI della cartella in cui salvare i file di output;
- una variabile “param” è un array bidimensionale di stringhe, in cui verranno memorizzate tutte le informazioni necessarie sui parametri da modificare nel

workflow.

4.1.3 I costruttori

I costruttori della classe sono generati in automatico dal wizard.

4.1.4 I metodi ausiliari

I metodi ausiliari da implementare sono due:

- il metodo “getFile” che si occupa di settare gli URI dei file di I/O;
- il metodo “setParametri” che si occupa di impostare i parametri da modificare nel workflow.

4.1.5 I casi d'uso

I metodi dei casi d'uso da implementare sono relativi ai casi d'uso previsti dal modello. Il corpo del caso d'uso deve:

1. creare un oggetto della classe KnimeModel;
2. settare i parametri necessari all'esecuzione: username utente, nome dell'esperimento, nome del modello di progetto KNIME e l'URI della cartella in cui salvare i file di output;
3. richiamare il metodo “getFile” per settare gli URI dei file di I/O e effettuare il download del dataset di input;

4. richiamare il metodo del modello per il caso d'uso richiesto;

5. riportare al termine dell'esecuzione lo stato i risultati ottenuti.

4.2. Il DMPlugin K-means

In questo paragrafo è descritta la creazione di un nuovo plug-in per il clustering su un dataset, mediante il nodo K-means di KNIME. In particolare la funzionalità fornisce le classi di cluster per un numero predefinito di cluster. Le informazioni di configurazione necessarie per lo svolgimento dell'esperimento sono:

- il numero di clusters da creare;
- il numero massimo di iterazioni;
- il formato del dataset in input.

In ingresso richiede:

- un dataset;

In uscita produce:

- un dataset in formato CSV;
- un modello in formato PMML;
- un file di log in formato TXT;
- un file di errore in formato TXT.

4.2.1 La creazione del DMPlugin

La creazione del DMPlugin per K-means avviene attraverso il wizard. In primo luogo compiliamo il form di configurazione (Fig. 61) con le informazioni sul plug-in, fornitore e modelli di esecuzione, come segue:

- il nome del plug-in, il riferimento alla documentazione, la versione del software e il dominio, nella sezione “Plugin information”;
- il nome e la mail del proprietario, nella sezione “Owner information”;
- selezioniamo le modalità di esecuzione previste dal nostro plug-in, nella sezione “Running Modes Information”.

Per il K-means plug-in è prevista solo la modalità “Run”, per la quale configuriamo le componenti “Field”, “Input File” e “Output File” (Fig. 61). Per il plug-in K-means sono necessari i seguenti “Field”:

- *NCluster* - il numero di clusters da creare (Fig. 62);
- *Niteration* - il numero di iterazioni (Fig. 63);
- *inFormat* - il formato del dataset in input.

Per il plug-in Kmeans sono necessari i seguenti “Input Files”:

- *inFile* – file del dataset in input (fig. 64).

Per il plug-in Kmeans sono necessari i seguenti “Output Files”:

- *filename* - file CSV (fig.65);
- *PMMLWriterFile* - modello PMML;
- *L* – file di log;
- *E* – file di errore;

Completata la configurazione del plug-in generiamo il relativo codice (Fig. 66), che consiste in una classe Java in cui verrà integrato il codice per la modalità d'esecuzione prevista.

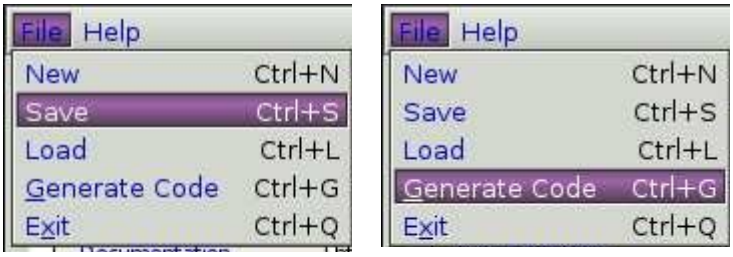

Fig. 66: Menù wizard - Save e Generate code

4.2.2 Implementazione della classe Kmeans

Il plug-in K-means è implementato con le caratteristiche di un KNIME plug-in (Par. 5.4.1).

Il plug-in esegue attraverso KNIME il workflow mostrato il figura (Fig. 67).

4.2.2.1 I metodi ausiliari

Vediamo come abbiamo implementato i metodi “getFile” e “setParametri”. Il metodo “getFile” svolge le seguenti operazioni:

- crea un nuovo oggetto per ognuno degli array “inputField”, “inputFiles” e “outputFiles”;
- legge i parametri configurati e li memorizza nell'array “inputField”, settandone il nome;
- legge la locazione del file di input, setta l'URI e effettua il download, memorizzandolo nell'URI settato;
- per ogni file di output setta l'URI in cui verrà memorizzato.

Il metodo “setParametri” svolge le seguenti operazioni:

- crea un nuovo oggetto array “param”;
- Viene impostata la prima riga dell'array con i valori per l'impostazione del file di input, assegnando:
  - alla prima colonna l'identificativo del nodo di lettura “1”;
  - alla seconda colonna il nome del parametro, letto in automatico dall'array inputFile;
  - alla terza colonna il valore che il parametro assumerà, letto in automatico dall'array inputFile;
  - alla quarta colonna il tipo “String”;
- Viene impostata la seconda riga dell'array con i valori per l'impostazione del formato di input, assegnando:
  - alla prima colonna l'identificativo del nodo di lettura “1”;
  - alla seconda colonna il nome del parametro, definito manualmente a “inFormat” poiché tutti i file sono sempre convertiti a CSV;
  - alla terza colonna il valore “CSV”;
  - alla quarta colonna il tipo “String”;
- imposta attraverso un ciclo tutti i parametri da modificare. Essendo i parametri (numero di cluster e numero di iterazioni) tutti relativi al secondo nodo , i valori della prima colonna saranno tutti sono tutti impostati a “2”. Il nome dei parametri del plug-in avendo gli stessi nomi e i valori dei parametri in KNIME, sono letti in automatico dalla struttura DMMFileParam. Essendo i parametri entrambi interi la quarta colonna è impostata a “Int”.

- Imposta in modo automatico i parametri di output leggendoli dall'array outputFile, effettuando un controllo per l'assegnazione dell'identificativo del nodo.

Il metodo ritorna il riferimento all'array creato.

4.2.2.2 I caso d'uso run

Il metodo per l'esecuzione del caso d'uso svolge le seguenti operazioni:

1. creare un oggetto della classe KnimeModel;
2. settare i parametri necessari all'esecuzione: username utente, nome dell'esperimento, nome del modello di progetto KNIME e l'URI della cartella in cui salvare i file di output;
3. richiamare il metodo “getFile” per settare gli URI dei file di I/O e effettuare il download del dataset di input;
4. richiamare il metodo del modello per il caso d'uso richiesto;
5. riportare al termine dell'esecuzione lo stato ed i risultati ottenuti attraverso il metodo reportStatus, che visualizza il contenuto dei file di output, del file di log e del file di errore.

4.2.3 L'integrazione nella Suite

La funzionalità K-means è integrata in DAME come una classe Java nel DMPlugin. Il seguente diagramma delle classi mostra dove è inserita (Fig. 68).

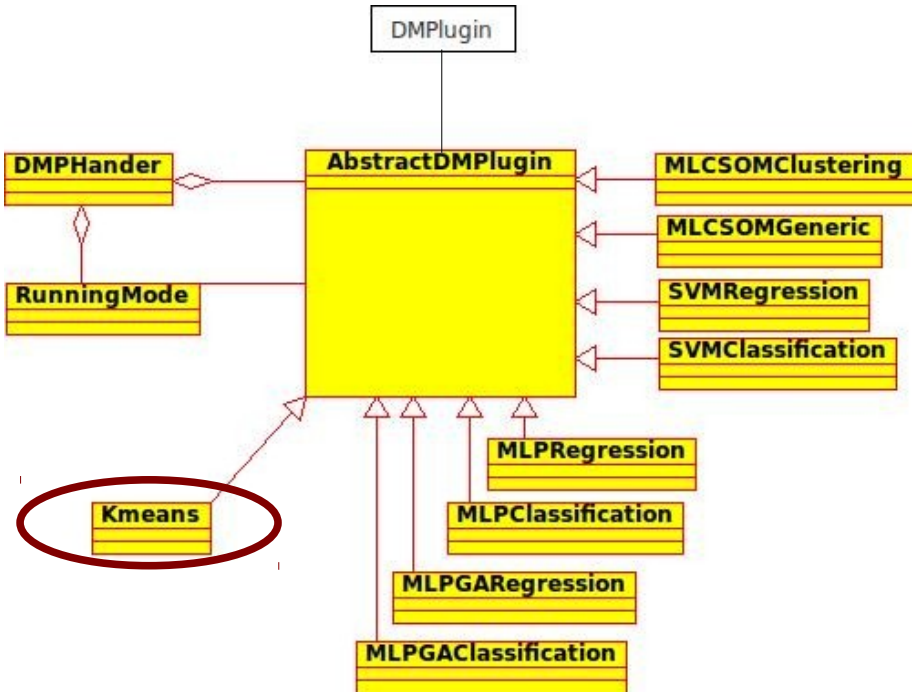

Fig. 68: Funzionalità del DMPlugin package

. I vincoli

I file di output dell'esperimento di default assumono il nome dell'esperimento. I file relativi ai casi d'uso assumono il nome dell'esperimento concatenato al caso d'uso che lo ha generato.

CAPITOLO 5

LA FASE DI TESTING

Questo capitolo racchiude le informazioni relative alla fase di verifica del corretto funzionamento dei componenti implementati. In prima istanza i componenti sono stati valutati singolarmente, successivamente abbiamo testato i componenti in modo congiunto.

In particolare sono stati testati prima i nodi StilReader e StilWriter. Il test del funzionamento del nodo StilReader (per la lettura, caricamento e conversione dei file dataset nei vari formati di uso astrofisico) (Par. 1) è articolato come segue:

1. test di lettura, caricamento e conversione di un dataset in formato ASCII;
2. test di lettura, caricamento e conversione di un dataset in formato FITS;
3. test di lettura, caricamento e conversione di un dataset in formato VOTABLE;
4. test di lettura, caricamento e conversione di un dataset in formato CSV;

Il test del funzionamento del nodo StilWriter (per caricamento, conversione e scrittura dei file dataset nei vari formati di uso astrofisico) (Par. 2) è articolato come segue:

1. test di caricamento, conversione e scrittura di un dataset in formato ASCII;
2. test di caricamento, conversione e scrittura di un dataset in formato FITS;

3. test di caricamento, conversione e scrittura di un dataset in formato VOTABLE;

4. test di caricamento, conversione e scrittura di un dataset in formato CSV;

Successivamente si è proceduto al test di funzionamento del modello KnimeModel (Par. 3). Seguono i risultati del test attraverso l'uso del plugin DAME per il workflow relativo al modello di clustering Kmeans (Par. 4). Il test del plugin è stato realizzato su dataset nei formati ASCII, FITS, VOTABLE e CSV. Riportiamo nel seguito i dettagli dei test su citati.

. Il nodo StilReader

Il testing del nodo StilReader è stato realizzato creando dei workflow attraverso la GUI di KNIME, allo scopo di verificare se il nodo esegue correttamente la lettura del dataset e se incapsula correttamente i dati nel formato usato da KNIME per il trasporto dei dati.

1. Testing su un dataset in formato ASCII

Il primo testing del nodo StilReader è stato effettuato sulla versione SDK KNIME su SO Linux Ubuntu.

1.1. Descrizione del testing

Dopo aver creato attraverso la GUI il workflow oggetto del testing (Fig. 69), è stato configurato per la lettura di un file ASCII (Fig. 70).

Per configurare i nodi successivi a StilReader è necessario che quest'ultimo sia eseguito. In seguito controlliamo nella cartella provvisoria se è stato creato il file temporaneo ottenuto dalla conversione del dataset. Nella cartella notiamo che il file temporaneo è stato creato correttamente (Fig. 71).

Per verificare se la conversione è andata a buon fine controlliamo il contenuto del file temporaneo (Fig. 72) ed il contenuto del file originario (Fig. 73).

La conversione è avvenuta con successo, possiamo notare inoltre che nel file creato il metodo translate ha creato un header di colonna. I nodi successivi sono configurati come da figure (Fig. 74, 75, 76).

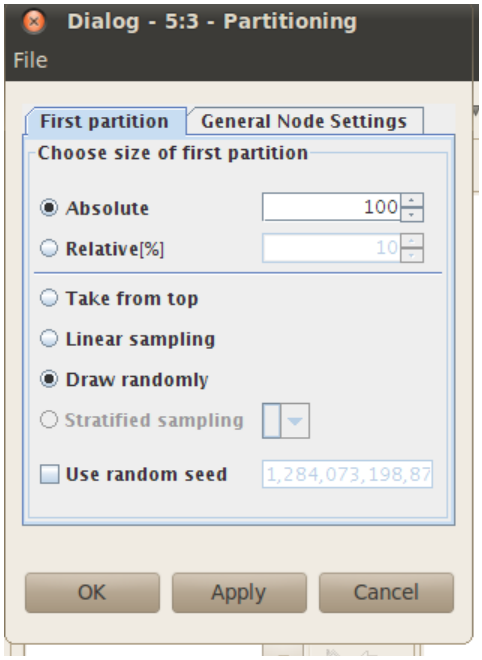

Fig. 74: Configurazione del nodo Partitioning

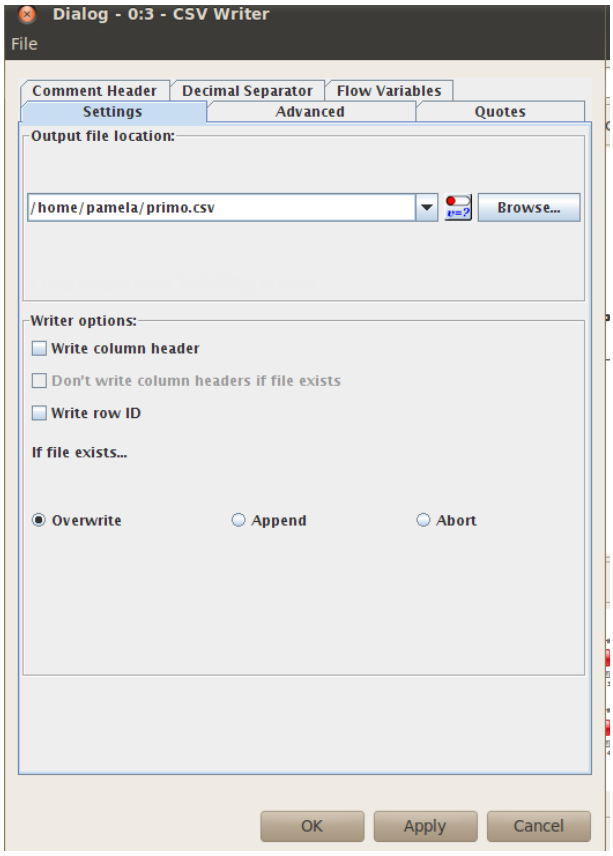

Fig. 75: Configurazione del nodo CSVWriter in alto

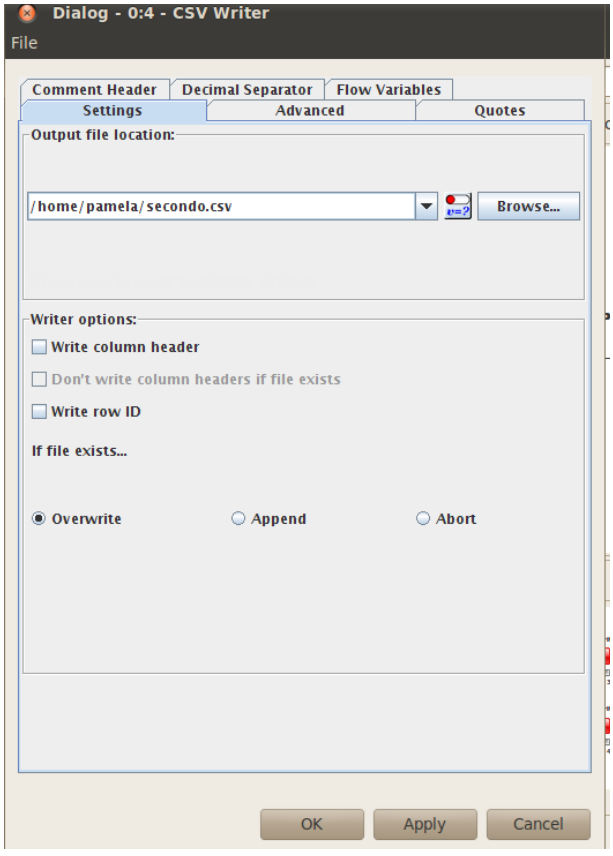

Fig. 76: Configurazione del nodo CSVWriter in basso

1.2. Risultati

In conclusione il nodo è stato eseguito correttamente quindi non ci sono problemi di lettura del formato interno. Ulteriore verifica si ha dall'esecuzione dei due nodi CSV, che creano correttamente i due file (Fig. 77) mostrando la corretta lettura dei dati (Fig. 78 – 79).

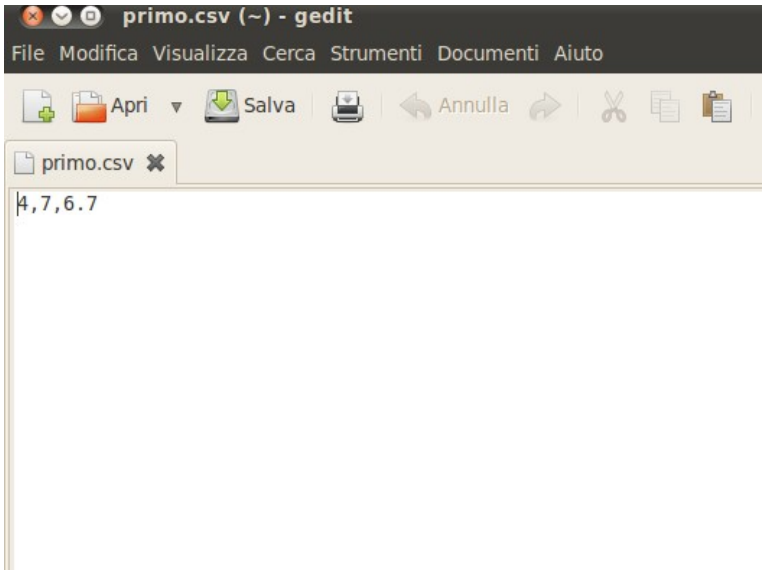

Fig. 78: Contenuto del file primo.csv

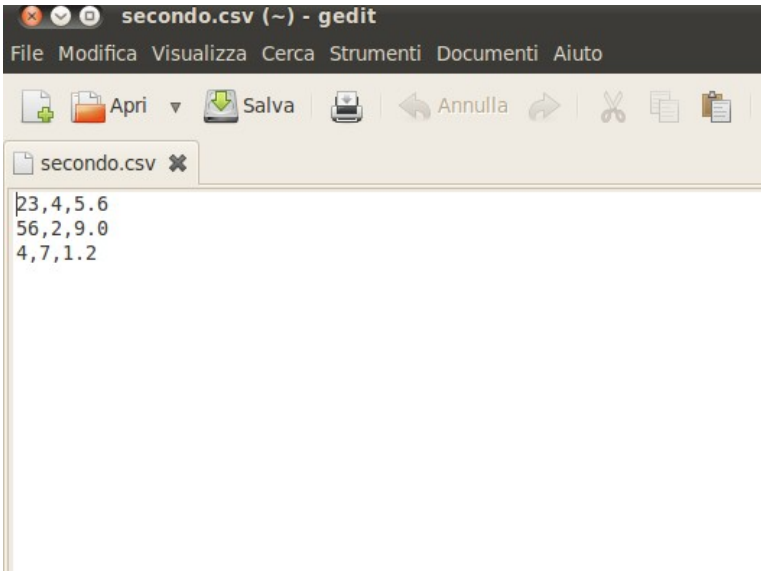

Fig. 79: Contenuto del file secondo.csv

Il testing del nodo StilReader per la lettura di dataset ASCII è andato a buon fine.

2. Testing su un dataset in formato FITS

Il secondo testing del nodo StilReader è stato effettuato sulla versione Desktop KNIME ver. 2.1.2 (32 bit) su SO Linux Ubuntu.

2.1. Descrizione del testing

Dopo aver creato attraverso la GUI il workflow oggetto del testing (Fig. 67), è stato configurato per la lettura di un file FITS (Fig. 80).

Per configurare i nodi successivi al StilReader è necessario che quest'ultimo sia eseguito. In seguito controlliamo nella cartella provvisoria se è stato creato il file temporaneo ottenuto dalla conversione del dataset. Nella cartella notiamo che il file temporaneo è stato creato correttamente. Vediamo ora il contenuto di del file temporaneo (Fig. 81) e del file originale (Fig. 82) per verificare il buon esito della conversione.

La configurazione dei nodi K-means (Fig. 83), CSV Writer (Fig. 84) e PMML Writer (Fig. 85) sono configurate come dalle successive figure.

2.2. Risultati

In conclusione il nodo è stato eseguito correttamente, infatti dall'esecuzione del workflow sono stati prodotti i file previsti.

3. Testing su un dataset in formato VOTABLE

Il terzo testing del nodo StilReader è stato effettuato sulla versione SDK KNIME su SO Linux Ubuntu.

3.1. Descrizione del testing

Il workflow oggetto del testing (Fig. 67), è stato configurato per la lettura di un file VOTABLE (Fig. 86).

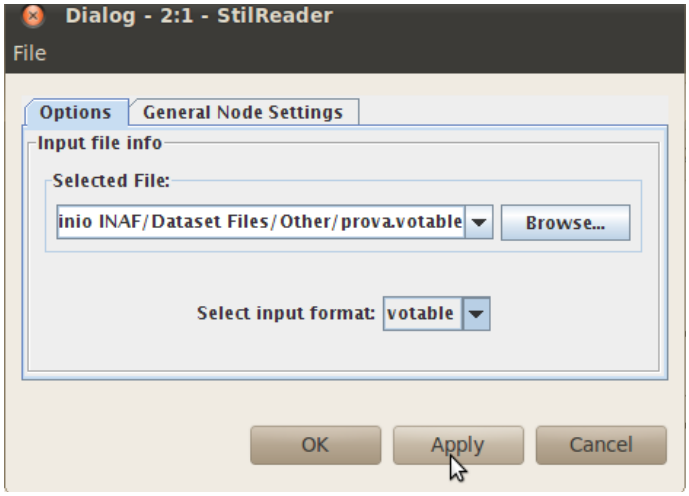

Fig. 86: Configurazione del nodo StilReader

La configurazione dei nodi Kmeans (Fig. 83), CSV Writer (Fig. 84) e PMML Writer (Fig. 85) sono rappresentate nelle figure seguenti. In seguito all'esecuzione di StilReader controlliamo nella cartella provvisoria se è stato creato il file temporaneo ottenuto dalla conversione del dataset. Nella cartella notiamo che il file temporaneo è stato creato correttamente (Fig. 87).

Verifichiamo il contenuto del file temporaneo (Fig. 88) e il contenuto del file originario (Fig. 89) per verificare se la conversione è avvenuta correttamente. Il contenuto del file temporaneo è corretto.

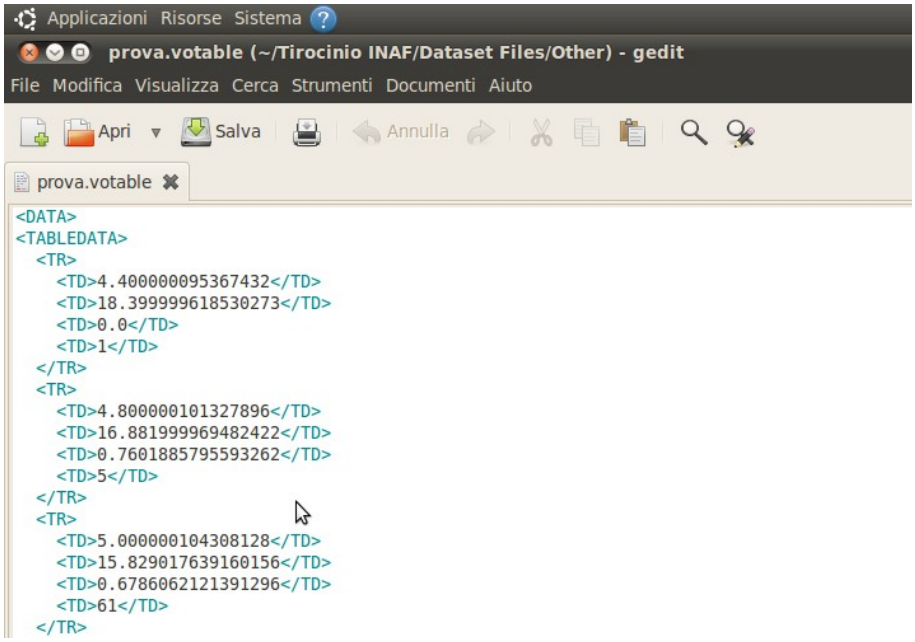

Fig. 89: Contenuto del file originario

3.2. Risultati

In conclusione il nodo è stato eseguito correttamente. Ulteriore verifica si ha dall'esecuzione del nodo CSV e del nodo PMML, creando correttamente i due file.

4. Testing su un dataset in formato CSV

Il secondo testing del nodo StilReader è stato effettuato sulla versione Desktop KNIME ver. 2.1.2 (32 bit) su SO Linux Ubuntu.

4.1. Descrizione del testing

Il progetto utilizzato è mostrato nell'immagine (Fig. 69). Il progetto è configurato per la lettura di un file CSV (Fig. 90).

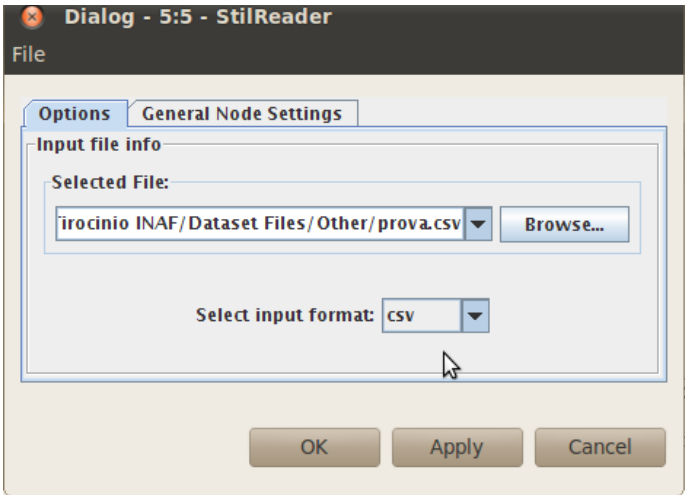

Fig. 90: Configurazione del nodo StilReader

In seguito all'esecuzione di StilReader controlliamo nella cartella provvisoria se è stato creato il file temporaneo ottenuto dalla conversione del dataset. Nella cartella notiamo che il file temporaneo è stato creato correttamente. Controlliamo il contenuto del file temporaneo (Fig. 91) e il contenuto del file originario (Fig. 92).

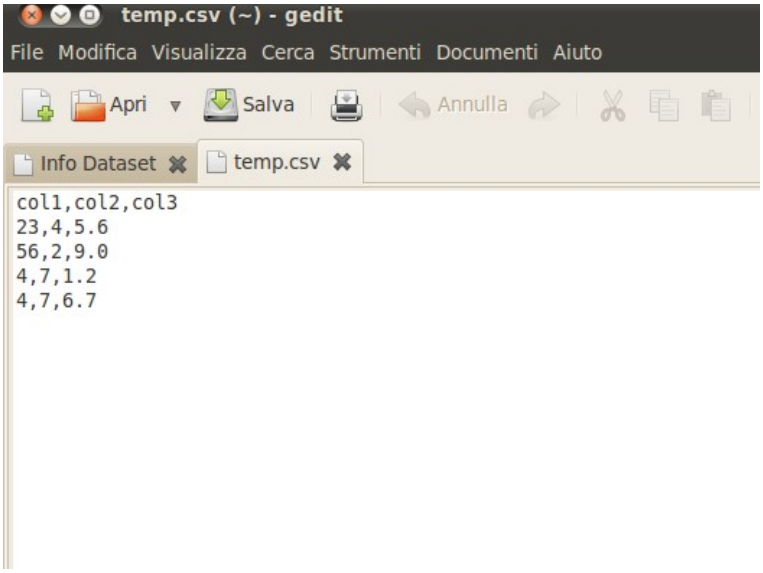

Fig. 91: Contenuto del file temporaneo

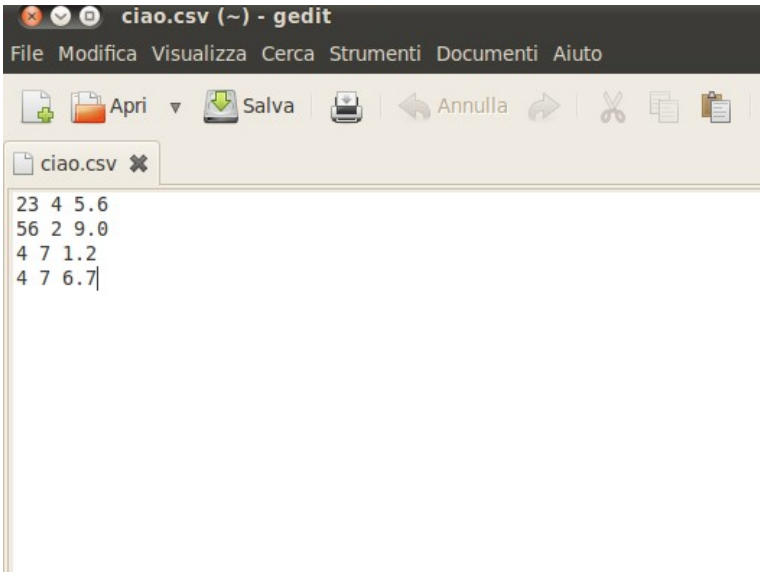

Fig. 92: Contenuto del file originario

4.2. Risultati

In conclusione il nodo è stato eseguito correttamente, il testing ha quindi avuto esito positivo.

. Il nodo StilWriter

Il testing del nodo StilWriter è stato realizzato creando il workflow attraverso la GUI di KNIME, allo scopo di verificare se il nodo esegue correttamente la scrittura del dataset. Il workflow composto legge un dataset attraverso il FileReader e lo scrive con StilWriter (Fig. 93).

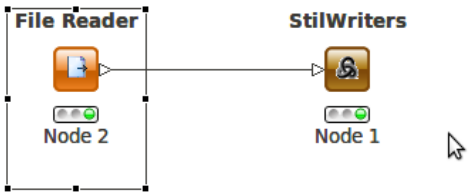

Fig. 93: Il workflow del testing di StilWriter

Il testing è stato realizzato attraverso la versione SDK KNIME su SO Linux Ubuntu. Il dataset utilizzato è rappresentato nella seguente figura (Fig. 94).

1. Testing su un dataset in formato ASCII

1.1. Descrizione del testing

Configurando il nodo StilWriter (Fig. 95) per la scrittura di un dataset ASCII otteniamo i risultati attesi (Fig. 96).

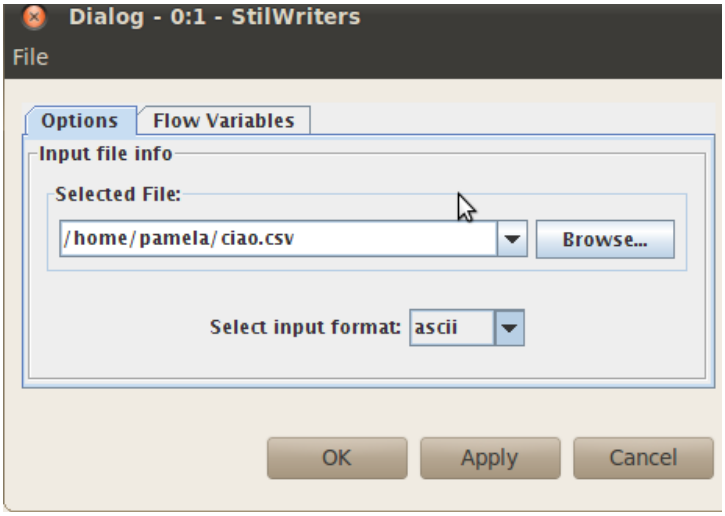

Fig. 95: Configurazione StilWriter ad un ASCII dataset

1.2. Risultati

La verifica è terminata con successo.

2. Testing su un dataset in formato CSV

2.1. Descrizione del testing

Configurando il nodo StilWriter (Fig. 97) per la scrittura di un dataset CSV otteniamo i risultati attesi (Fig. 98).

2.2. Risultati

La verifica è terminata con successo.

3. Testing su un dataset in formato FITS

3.1. Descrizione del testing

Configurando il nodo StilWriter (Fig. 99) per la scrittura di un dataset FITS otteniamo i risultati attesi (Fig. 100).

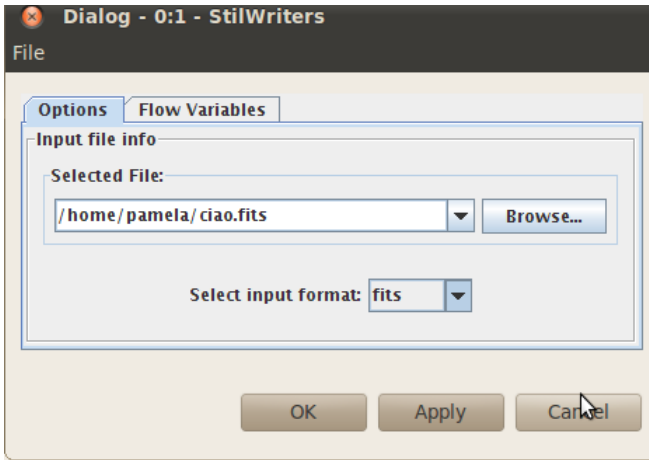

Fig. 99: Configurazione StilWriter ad un FITS dataset

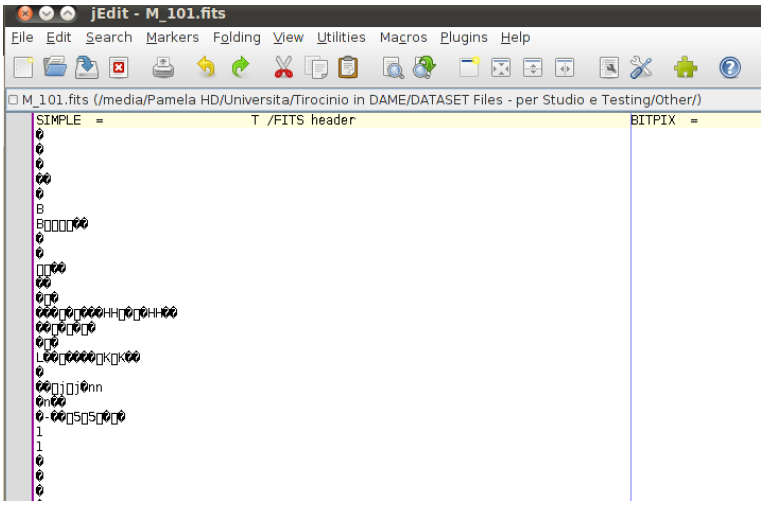

Fig. 100: Il dataset risultante

3.2. Risultati

La verifica è terminata con successo.

4. Testing su un dataset in formato VOTABLE

4.1. Descrizione del testing

Configurando il nodo StilWriter (Fig. 101) per la scrittura di un dataset VOTABLE otteniamo i risultati attesi (Fig. 102).

4.2. Risultati

La verifica è terminata con successo.

. Il modello KnimeModel

1. Descrizione del testing

Per testare il modello KnimeModel abbiamo creato un main che importa il modello e attraverso un main utilizza il metodo createKnimeProject per verificare il corretto funzionamento del modello. Abbiamo quindi impostato i parametri da passare alla funzione come segue:

```
String utente = "pamela";
String nomeEsperim = "Kmeans_Pamela";
String nomeZip = "KmeansProject.zip";
String [][]param = new String[6][4];

param[0][0]="3";
param[0][1]="filename";
param[0][2]="/home/pamela/Scrivania/Testing KNIMEManager/Dataset/
primo.csv";
param[0][3]="String";
param[1][0]="2";
```

```
param[1][1]="nrClusters";
param[1][2]="";
param[1][3]="Int";
param[2][0]="2";
param[2][1]="maxNrIterations";
param[2][2]="";
param[2][3]="Int";
param[3][0]="1";
param[3][1]="inFile";
param[3][2]="";
param[3][3]="String";
param[4][0]="1";
param[4][1]="inFormat";
param[4][2]="";
param[4][3]="String";
param[5][0]="4";
param[5][1]="PMMLWriterFile";
param[5][2]="/home/pamela/Scrivania/Testing KnimeManager/Dataset/
secondo.pmml";
param[5][3]="String";
```

2. Risultati

Richiamando la funzione con questi parametri si ottiene il risultato desiderato, ossia il progetto è eseguito correttamente e il salvataggio dei file di output avviene correttamente nella locazione specificata. Il metodo crea nella cartella pamela/workspace il progetto “Kmeans_Pamela”, e nella cartella utente i file di output. Il metodo funziona correttamente.

. Il plugin K-means

Per valutare il funzionamento del plugin K-means, abbiamo eseguito esperimenti configurando opportunamente i parametri. In particolare, gli esperimenti condotti sono stati su dataset nei formati astronomici (FITS, VOTABLE, CSV e ASCII).

1. Testing su un dataset in formato FITS

1.1. Descrizione

Per eseguire l'esperimento su un dataset FITS abbiamo utilizzato il dataset in Fig. 82. I parametri dell'esperimento sono stati configurati configurati come segue:

- “inFormat” a “fits” ;
- “Ncluster” a 3;
- “Niteration” a 20.

1.2. Risultati

La seguente tabella riassume completamente tutti i parametri configurati ed i risultati ottenuti:

Nome Esperimento	Kmeans_Pamela_1
inFile	/home/pamela/Tirocinio INAF/Dataset Files/Other/M_101.fits Fig. 82
inFormat NCluster Niteration	fits 3 20
filename	/home/pamela/primo Fig. 103
PMMLWriterFile	/home/pamela/secondo Fig. 104
L	/home/pamela/Kmeans_Pamela_1.log WARN CSV Writer Selected output file exists and will be overwritten! Workflow Manager 0:2: CONFIGURED (start) StilReader 0:2:1 (EXECUTED) k-Means 0:2:2 (CONFIGURED) CSV Writer 0:2:3 (CONFIGURED) PMML Writer (BETA) 0:2:4 (CONFIGURED) Workflow Manager 0:2(end) WARN CSV Writer Selected output file exists and will be overwritten! Finished in 1 sec (1337ms)
E	/home/pamela/Kmeans_Pamela_1.error

Possiamo quindi affermare che l'esperimento si è concluso con successo.

2. Testing su un dataset in formato VOTABLE

2.1. Descrizione

Per eseguire l'esperimento su un dataset VOTABLE abbiamo utilizzato il dataset in Fig. 105. I parametri dell'esperimento sono stati configurati configurati come segue:

- “inFormat” a “votable” ;
- “Ncluster” a 5;
- “Niteration” a 26.

2.2. Risultati

La seguente tabella riassume completamente tutti i parametri configurati ed i risultati ottenuti:

Nome Esperimento	Kmeans_Pamela_2
inFile	/home/pamela/Tirocinio INAF/Dataset Files/Other/prova.votable Fig. 105
inFormat NCluster Niteration	votable 5 26
filename	/home/pamela/primo Fig. 106
PMMLWriterFile	/home/pamela/secondo Fig. 107

L	/home/pamela/Kmeans_Pamela_2.log
	
	Workflow Manager 0:2: EXECUTED (start)
	  StilReader 0:2:1 (EXECUTED)
	  k-Means 0:2:2 (EXECUTED)
	  CSV Writer 0:2:3 (EXECUTED)
	  PMML Writer (BETA) 0:2:4 (EXECUTED)
	Workflow Manager 0:2(end)
	Finished in 0 secs (572ms)

E	/home/pamela/Kmeans_Pamela_2.error

Possiamo quindi affermare che l'esperimento si è concluso con successo.

3. Testing su un dataset in formato CSV

3.1. Descrizione

Per eseguire l'esperimento su un dataset CSV abbiamo utilizzato il dataset in Fig. 108. I parametri dell'esperimento sono stati configurati configurati come segue:

- “inFormat” a “csv” ;
- “Ncluster” a 4;
- “Niteration” a 50.

3.2. Risultati

La seguente tabella riassume completamente tutti i parametri configurati ed i risultati ottenuti:

Nome Esperimento	Kmeans_Pamela_4
inFile	/home/pamela/Tirocinio INAF/Dataset Files/Other/prova.csv Fig. 108
inFormat NCluster Niteration	CSV 4 50
filename	/home/pamela/primo Fig. 109
PMMLWriterFile	/home/pamela/secondo Fig. 110

L	/home/pamela/Kmeans_Pamela_4.log Workflow Manager 0:2: EXECUTED (start) StilReader 0:2:1 (EXECUTED) k-Means 0:2:2 (EXECUTED) CSV Writer 0:2:3 (EXECUTED) PMML Writer (BETA) 0:2:4 (EXECUTED) Workflow Manager 0:2(end) Finished in 0 secs (537ms)
E	/home/pamela/Kmeans_Pamela_4.error

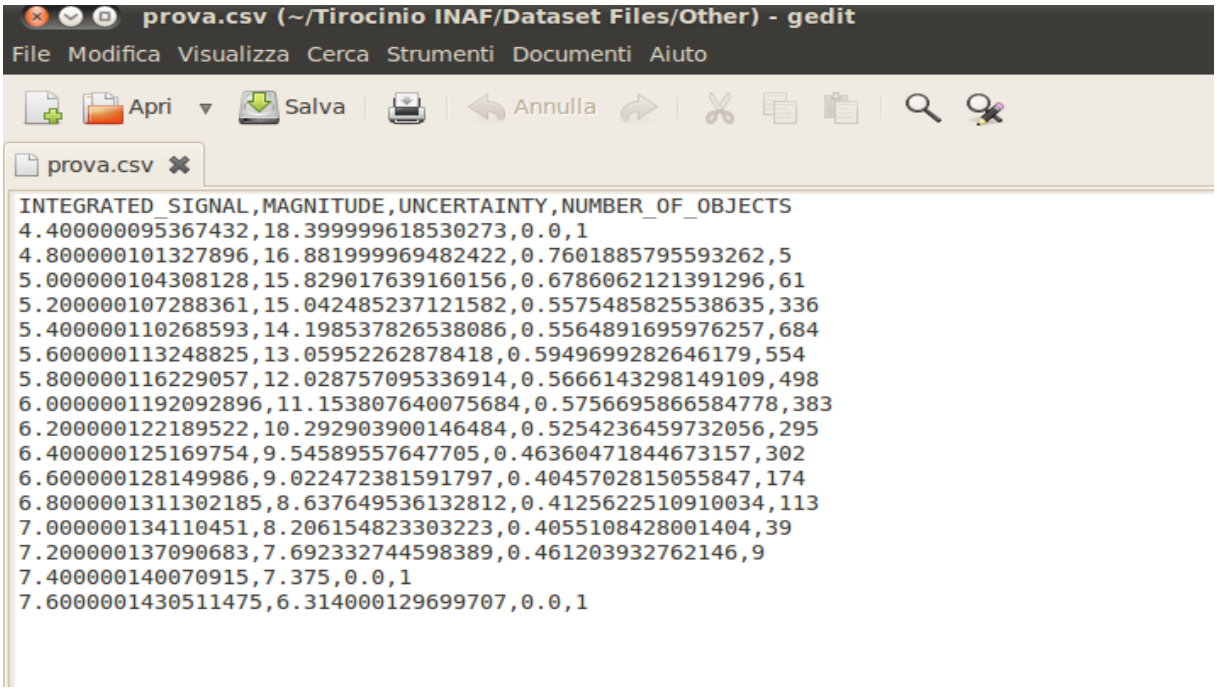

Fig. 108: File di input al testing 4 Kmeans

Possiamo quindi affermare che l'esperimento si è concluso con successo.

4. Testing su un dataset in formato ASCII

4.1. Descrizione del testing

Per eseguire l'esperimento su un dataset ASCII abbiamo utilizzato il dataset in Fig. 111. I parametri dell'esperimento sono stati configurati configurati come segue:

- “inFormat” a “ascii” ;
- “Ncluster” a 4;
- “Niteration” a 50.

4.2. Risultati

La seguente tabella riassume completamente tutti i parametri configurati ed i risultati ottenuti:

Nome Esperimento	Kmeans_Pamela_5
inFile	/home/pamela/Tirocinio INAF/Dataset Files/Other/prova.csv Fig. 111
inFormat NCluster Niteration	ascii 4 50
filename	/home/pamela/primo Fig. 112
PMMLWriterFile	/home/pamela/secondo Fig. 113

L	/home/pamela/Kmeans_Pamela_5.log
	
	Workflow Manager 0:2: EXECUTED (start)
	  StilReader 0:2:1 (EXECUTED)
	  k-Means 0:2:2 (EXECUTED)
	  CSV Writer 0:2:3 (EXECUTED)
	  PMML Writer (BETA) 0:2:4 (EXECUTED)
	Workflow Manager 0:2(end)
	Finished in 0 secs (520ms)

E	/home/pamela/Kmeans_Pamela_5.error

Possiamo quindi affermare che l'esperimento si è concluso con successo.

. Testing su errore

1. Testing su un dataset in formato FITS

1.1. Descrizione

Per valutare il comportamento del DMPlugin in caso di errore abbiamo configurato come dataset di input un file in formato FITS e il parametro “inFormat” a “votable”. In tal caso il plug-in dovrà segnalare un errore.

1.2. Risultati

La seguente tabella riassume completamente tutti i parametri configurati ed i risultati ottenuti:

Nome Esperimento	Kmeans_Pamela_3
inFile	/home/pamela/Tirocinio INAF/Dataset Files/Other/M_101.fits Fig. 82
inFormat NCluster Niteration	votable 3 20
filename	/home/pamela/primo
PMMLWriterFile	/home/pamela/secondo
L	/home/pamela/Kmeans_Pamela_3.log
E	/home/pamela/Kmeans_Pamela_3.error

	ERROR StilReader Execute failed: dameError.Error@439b37 Can't open /home/pamela/Tirocinio INAF/Dataset Files/Other/M_101.fits as VOTable (XML parse error (Content is not allowed in prolog.))

Possiamo quindi affermare che l'esperimento si è concluso con successo.

CAPITOLO 6

CONCLUSIONI

Attraverso l'integrazione tra il kernel della piattaforma per l'analisi dei dati di KNIME e la Suite su calcolo distribuito per il data mining di DAME, si è riusciti a garantire la piena interoperabilità tra i due servizi, per definizione complementari sia in termini di architettura che di obiettivo scientifico, a tutto vantaggio della comunità di e-learning (in particolare quella astrofisica), in grado di poter sfruttare appieno le qualità delle moderne tecnologie basate su Web per il trattamento (analisi ed esplorazione) di Massive Data Sets su infrastrutture di calcolo ibride (CLOUD, GRID, Stand-Alone).

Il presente lavoro di tesi ha raggiunto il principale obiettivo di innescare questo processo d'integrazione tra le due piattaforme, che culminerà nel prossimo futuro con la fruizione dei servizi integrati direttamente attraverso la GUI della Suite DAME.

Tale obiettivo è stato perseguito prestando particolare attenzione a non causare modifiche strutturali a entrambe le piattaforme. Di conseguenza la difficoltà maggiore incontrata in fase di progettazione è consistita nel riuscire a conciliare entrambe le esigenze relative agli standard di rappresentazione dell'informazione, ai protocolli di comunicazione ed alle specifiche architetturali.

L'interoperabilità tra due strutture complesse, basate sulle nuove tecnologie Web, su architetture di calcolo orientate ai servizi ed alle risorse, in grado di poter gestire voluminosi database distribuiti, mettendo a disposizione potenti strumenti ed algoritmi per l'analisi,

l'esplorazione ed il data mining, ha permesso dunque di sviluppare un servizio per utenti esterni pienamente efficiente ed in linea con la rivoluzione in atto nel settore della ricerca scientifica e tecnologica, basata sul quarto paradigma della scienza, dopo teoria, sperimentazione e simulazione: l'e-science.

CAPITOLO 7

RIFERIMENTI BIBLIOGRAFICI

[1] Brescia M., *DMPlugin Component Description*, 2010, DAME Project Official Website, Napoli, http://voneural.na.infn.it/dmplugin.html

[2] Berthold M. R., et al., KNIME: *The Konstanz Information Miner*, 2007, University of Konstanz (Konstanz, Germany), http://www.unikonstanz.de/

[3] Fiore M., *Il Componente Framework del progetto Dame*, A.Y. 2007/08, M. Sc. Computer Science, Università degli studi Federico II di Napoli, http://voneural.na.infn.it/documents.html

[4] Manna F., *Una Web Application su infrastruttura GRID per il progetto DAME,* A.Y. 2008/09, M. Sc. Computer Science, Università degli studi Federico II di Napoli, http://voneural.na.infn.it/documents.html

[5] Ochsenbein F., *IVOA VOTable*, 2009, IVOA Recommendation, http://ivoa.net/Documents/VOTable/20091130/REC-VOTable-1.2.html

[6] FITS Working Group , *Definition of the Flexible Image Transport System (FITS)*, 2008, http://fits.gsfc.nasa.gov/standard30/fits_standard30.pdf

[7] Taylor M., *Starlink Tables Infrastructure Library (STIL)*, 2009,

http://www.star.bris.ac.uk/~mbt/stil/sun252.html

[8] Shin S., *GWT – Google Web Toolkit*, http://javapassion.com/ajax/GWT.pdf

[9] Esposito P., *KNIME in DAME Project Description Document*, 2010, DAME Project Technical Documentation, VONEURAL-PDD-NA-0003-Rel1.6

[10] Esposito P., *DAMEKappA software requirement specifications*, 2010, DAME Project Technical Documentation, DAME-SRS-NA-0007-Rel1.0

[11] Esposito P., *DAME-KappA Software Design Description*, 2010, DAME Project Technical Documentation, DAMESDD-NA-0014-Rel0.1

[12] Esposito P., *Internal Lecture on KNIME*, 2010, DAME Project Technical Documentation, DAME-NA-PRE-0024

[13] Esposito P., *Internal Lecture on KNIME - Kernel functionality study - part II*, 2010, DAME Project Technical Documentation, DAME-NA-PRE-0026

[14] Fiore M., Cavuoti S., *DMPlugin end user manual*, 2010, DAME Project Technical Documentation, VONEURAL-MAN-NA-0004-Rel1.3

[15] Fiore M., Cavuoti S., *DMPlugin administrator manual*, 2010, DAME Project Technical Documentation, VONEURAL-MAN-NA-0005-Rel1.6

[16] Cavuoti S., *Data Mining Models component Software Requirement Specifications*, 2009, DAME Project Technical Documentation, VONEURAL-SRS-NA-0005-Rel0.4

[17] Di Guido A., Cavuoti S., *Test report of DMM component in DAME Suite*, 2009, DAME Project Technical Documentation, VONEURAL-TRE-0012-Rel.0.2